Automated Analysis of Computer-Generated Software Usage Protocols: An Exploratory Study

TR91-052

December, 1991

John Q. Walker, II

The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175
919-962-1792
jbs@cs.unc.edu

A TextLab/Collaboratory Report

JOHN QUILLIAN WALKER II. Automated Analysis of Computer-Generated Software Usage Protocols: An Exploratory Study (Under the direction of John Bristow Smith.)

ABSTRACT

Highly-interactive computer software can potentially help users think and work more effectively. To realize this potential, software developers should understand the cognitive processes involved in the tasks being performed and the ways users interact with the software to accomplish the tasks.

Gathering data about software usage—called protocols—is costly in several ways, including preparing representative test scenarios, finding suitable subjects, training personnel to administer the tests and record the protocols, and collecting and coding the protocols. Similarly, analyzing protocols can be tedious, often done manually by skilled researchers. Because of their high costs, protocol studies frequently consist of a few subjects tested while performing synthetic tasks in an artificial setting. The value of these studies is limited both for software developers and researchers in human-computer interaction.

This paper describes a method used to collect and analyze the protocols of a large number of subjects performing tasks in a naturalistic setting. An interactive computer program was developed as a testbed for this study. It contained an automatic tracker that unobtrusively collected protocol records of users' interactions with the program. Users' strategies in working with this program were modeled as a formal grammar, and a parser was devised, based on the grammar, to analyze protocol records produced by the program. Users' behaviors and strategies of working with the program were examined and characterized, based upon the parsed protocol data.

A graphical structure editor was created as the testbed for this study; it assists in expository writing, such as technical journal articles, with special emphasis on the exploratory and organizational phases of the writing process. This paper discusses lessons learned in devising the grammar to model users' writing sessions with the editor, in building and refining the parser, and in analyzing the protocol records for 112 sessions collected from 29 subjects.

Dissertation Thesis Statement

A practical methodology can be developed for automating the collection and analysis of large amounts of protocol data from users of interactive software in their natural working environment.

ACKNOWLEDGEMENTS

Working full-time and writing a dissertation took a lot of help!

John B. Smith was the best advisor I could ask for. He's creative and inspiring; I continue to learn a lot from him. I appreciate the innumerable hours he's invested in this project.

What a dissertation committee!—alphabetically: David Beard, Fred Brooks, Marcy Lansman, and Steve Weiss. It's the best committee I can imagine; their contributions to my education are deeply appreciated.

Members of the Textlab group were always available to exchange ideas on the topics discussed here; we've enriched each others' projects: Jay Bolter, Gordon Ferguson, Bobby Stam, Mark Rooks, Irene Weber, Valerie Kierulf, Greg Berg, and Katie Clapp. I also acknowledge the authors of the original PROSE program: Nancy Donaldson, Tom Morris, Lane Price, and Jim Shoaf. Finally, I shared an office at UNC with some enlightened fellow students: Will Partain and Phil Amburn.

IBM and my excellent managers there allowed me the luxury to pursue this degree; I'll assume they're starting to see the payback. Over more than ten years it's become a long list; thanks team, you're great! My direct management team has included: Otis Bradley, Herb Pelnar, Nishan Bouroudjian, Frank Corr, Joel Webb, Jim Staton, Jane Munn, Fred McGriff, Denise Kanyuh, Bob Steen, Don Haile, John Hunter, and Rick McGee.

There's a lot of other support I got from IBM; in particular I thank the subjects of this study for helping me improve *Prose II* and for bothering to send me their interesting recordings. I used a number of world-class IBM tools; thanks to their authors—I hope I am now returning the privilege by supporting my own set. Key IBM software I used were VNET, BookMaster, almcopy, Personal Editor II, FileCommand, gfind, and gsee. Thanks to Margie Weaver and June Turner, who administered the IBM Ph.D. Resident Study and Graduate Work-Study programs.

I've been in three departments here in IBM RTP: E75, E64, and E42. Thanks to the terrific friends I've had there, many of whom have moved on far from here. I can't have asked for a better place to work. My colleagues have exhibited amazing patience with my pursuit of this "second job."

I've had some wonderful friends who've joined me in the good days and the frustrating days of this project. Terry Carns, Jane Troller, and Jay Whitehead put up with my habits. Thanks always to Trileigh, Suzanne, Stephanie, and Rita. Other enduring friends know how they've been frequent help and inspiration: Bob and Linda Gibson, Jim and Becki Beck, Bill Hogan, Jay Kurtz, Don Andrews and Amy Svoboda and Georgia, John and Alice Pierce, Bob and Nancy Donnan, Mark and Sherrie Nickels, Peter and Suzanne Schwaller, Steve and Julie Joyce, and Leonard and Marlene Van Camp.

I'm sure my wonderful sisters and their families will be glad to see this completed: Kathy and John Wyrwich, and Frances and Bruce Walker. My grandmother, Lucille Maloney, would have been surely glad to see it done. Finally, thanks Mom and Dad for everything—finally!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ANSI	American National Standards Institute
ASCII	American Standard Code for Information Exchange
ATN	augmented transition network
BNF	Backus-Naur form
CISC	complex instruction set computer
CPU	central processing unit
DOS	disk operating system
FFT	fast Fourier transform
FIFO	first-in-first-out
FSM	finite state machine
GML	generalized markup language
GOMS	goals, operators, methods, and selection rules
GUI	graphical user interface
IBM	International Business Machines Corporation
I/O	input/output
Kbytes	kilo-bytes
LEX	lexical scanner generator
LTM	long-term memory
MIT	the Massachusetts Institute of Technology
ms	milli-second
μs	micro-second
OS/2	Operating System/2
PC	personal computer
PIF	program information file
RAM	random access memory
RISC	reduced instruction set computer
stdin	standard input
stdout	standard output
tde	transition diagram editor
UNC	the University of North Carolina at Chapel Hill
VAX	Virtual Address eXtension
VLSI	very large scale integration
VM	Virtual Machine
VNET	VM network
WE	Writing Environment
WM	working memory
YACC	yet another compiler-compiler

CHAPTER 1. AN INTRODUCTION TO THIS PROJECT

Recent computer software is highly interactive. It promotes a continual dialogue between the computer system and its human users. This software is often designed to help people carry out intellectual tasks more efficiently and more effectively, that is, to help users think better with respect to a given task.

As an example, recent interactive software for writing and planning offers users support for several kinds of thinking:

- remembering and organizing thoughts,
- exploring relationships among ideas,
- developing clear and consistent organizational structures, and
- deducing unexpected results.

Feedback to software designers is increasingly important if they are to match the software tools they create to the complex mental tasks the software is designed to support. Yet as the interaction between human and computer becomes more tightly coupled, software designers generally have little understanding of how their software is actually used. Why? One reason is that collecting the necessary information is expensive and obtrusive, and its analysis is inconsistent and inconclusive. To make such feedback inexpensive, consistent, and readily available, powerful tools are needed to automate its collection and analysis.

One source of help is the techniques already employed for examining human-computer interaction. Card, Moran, and Newell have initially surveyed the study of computer usage in their landmark book, *The Psychology of Human-Computer Interaction* (1983). They summarized a large body of low-level experimental data, presenting a model of humans as rational information processors, much like computers. From this basis, Card, Moran, and Newell developed a method for dividing structured tasks into smaller subtasks, to form a task hierarchy. The subtasks are small enough so that humans (with experience) can do them automatically; for example, moving a cursor on a computer screen to a particular location. Although much of their book highlights such low-level interaction, it discusses one extended example of how software usage in accomplishing a complex task might be analyzed.

In the research of Card, Moran, and Newell, as in the research of others, observations of users and analyses of protocol data are done primarily by hand. This approach is labor-intensive, making it hard to study many subjects and obtain consistent results. Subjects in studies of human-computer interaction are conscious of "being watched," since they are studied in controlled settings. Further, testing of human-computer interaction frequently focuses on how a system is first learned and how its concepts are grasped. There is little knowledge of how users' patterns change as they become familiar with a given system in the settings where it is actually used.

The project described in this paper builds on the work of Card, Moran, and Newell and others. In particular, the project addresses several problems with existing techniques for understanding software usage; the intent is to make protocol data less expensive, less obtrusive, less inconsistent, and thus more timely, more accurate, and more useful. The project consists of a cycle of designing and implementing a software system, developing tools to automate the collection and analysis of software usage data, conducting a study in which a large amount of software usage data was collected and analyzed, and reflecting on the lessons learned in this exercise.

Overall targets for the project were the following:

- to clarify the issues encountered,
- to provide a firmer foundation of repeatable techniques,
- to provide insight on how to build interactive software and tools of this type,
- to discuss interesting findings in the data, and
- to give directions for future research and controlled studies.

Finally, this project was exploratory; it was not intended to provide a definitive solution. It used a novel set of tools to explore human-computer interactions in a way not otherwise practical. Its experimental study of actual users was open-ended, with few controls. Thus, it represents a first try at a process that could be refined in future iterations.

1.1 Project Overview

This project was designed for one cycle of a software design and feedback process. It included the following steps:

1. Select and understand a specific task that contained a variety of types of human-computer interaction.
2. Build a testbed software system that helps accomplish the elements of the task.
3. Build tools to automate the collection and analysis of data that describe users' interaction with the system.
4. Run an experimental study of many subjects using the system in their actual settings. Collect and analyze their software usage data.
5. Reflect on the results of the study and the analysis.

The specific task that was examined was writing, in particular, its early organizational stages. I built a software system to assist writers in organizing and writing about their ideas. This software included an internal tracker that automatically recorded key elements of the human-computer interaction. The analysis of these recordings was automated, using techniques developed by Card, Moran, and Newell (1983) in their manual analysis of a single session. To test the feasibility of this approach, I collected and analyzed a large amount of protocol data in a field study of 29 subjects.

The details of these five steps are further summarized in the remainder of this section. These topics are covered in more detail in the later chapters of this paper. In particular, Chapters 5, 6, and 7 present three sets of observations. The first set describes the results of the protocol analysis for this writing system: what was learned about the cognitive strategies employed by writers, and how the structure editor was used to accomplish the writing tasks. The second set of observations centers on the meta-issues involved in doing automated protocol analysis, such as the power and viability of specifying the strategy model as a grammar and then using the associated parser for protocol analysis. The third set of observations concerns the feedback that the analysis provides for the software design process.

A simple analogy is helpful in describing the complexity of this project and the types of results obtained. Suppose you had never played the game of baseball, and you knew nothing about its rules and statistics. Then, suppose you were given recordings of the "uncolored" play-by-play for a baseball team for an entire season. It is your job to infer the rules of the game from these recordings. It is also up to you to invent meaningful statistics to be used in evaluating the team and the individual players. These tasks are roughly analogous to the steps in this project, but they do not cover the entire project.

To complete the analogy... a first set of observations involves statistics: given the possible baseball statistics you can invent, which are most meaningful for comparing players, games, and overall strategy? A second set of observations involves the process of working with baseball statistics: what is the best way to collect, categorize, and generate these statistics? A third set of observations addresses using these statistics; what is the best way to use these results to coach individual players, to build better bats or gloves, or to improve the rules and regulations of the game, itself?

The following subsection introduces the components of this project. The mental task being studied is technical writing. A testbed software system was implemented to assist in elements of the writing task, namely its exploratory and organizational phases. To see how people used this software for writing, a recorder was built to capture user commands, and software tools were constructed to automate the analysis of the recordings. To test the efficacy of this approach, an experimental study was conducted to collect usage data from users in the field. Major results of the project are summarized in the next section.

1.1.1 The Task: Writing

Writing is a fundamental activity of technical professionals. For example, a recent report estimated that design engineers spend 30% of their time writing documentation, twice the time they spend in other technical design (Business Week, 1986). They write technical reports and summaries, system documentation, strategic plans, training materials, letters, and memoranda. For many, writing is difficult and unpredictable in terms of both time and results.

Writing involves several different kinds of thinking: some are at a low level, such as spelling and text editing, and some are at a high level, such as organizing how ideas are related. Writing also involves strategies for moving among these levels, and tactics that can be used when problems arise. An understanding of these kinds of thinking, along with the strategies and tactics, would be useful for software developers who could design their systems to be consistent with these patterns of thinking.

Cognitive psychologists have devoted much research to understanding how people represent the external world internally. In studying writing, the problem is reversed. Writers must represent their internal thoughts externally. At the initial, exploratory stage of writing a document, a writer is likely to have in his or her head a loosely-connected set of ideas or concepts relevant to the topic at hand. To convert these concepts into a document, a writer must organize them into a coherent structure. Writers play with their ideas and concepts, in their head and sometimes on paper, until the concepts are connected into a network of relationships and associations. This network must eventually be transformed into the hierarchical form familiar in articles and manuscripts (that is, chapters, sections, and paragraphs), and then into a linear sequence of text, with the desired "connecting tissue."

The testbed system used in this project assists these activities. It is intended to help writers represent and build a structure of concepts during the exploratory and organizational stages of preparing a document.

1.1.2 The Testbed Software System: A Tool to Help Writers Organize

The results of a literature survey by the Textlab group at UNC (Smith and Lansman, 1988) suggest that a tool that allows writers to create visual images of the relationships between their ideas can facilitate the organizational process (for example, see Shepard, 1978). In addition, they also assert that the tool would be most effective for expository writing if it encourages a hierarchical representation of ideas. One reason is evidence from reading comprehension studies that hierarchically-organized, expository documents are easier to understand than documents with other structures. Other reasons have to do with current cognitive theories on the internal mental organization of knowledge and our perception of visual images. Smith and Lansman (1988) further describe the motivation for designing such a system. Many of their ideas are embodied in the software used in this project, as well as in their *Writing Environment (WE)* system.

The software built for this project is a graphical structure editor writers can use to help them explore and organize their ideas. With this system, they can create and label nodes, where each node represents a single idea. Thus, in the earliest stages of writing, writers might use the system as an aid to memory. They can then move and rearrange ideas, clustering on the screen ideas that they see as related conceptually to one another. Nodes can also be linked with one another in a hierarchical manner, and the resulting tree structures can, in turn, be moved and joined with other trees.

Any time after a node is defined, writers can write text to be associated with that node. They may select a node and a text-input option, normally a conventional text editor. If this node appears at an upper level of the hierarchy for the document, the text is likely to be an introductory statement or an overview of a section; on the other hand, text within a leaf node is likely to be a group of concrete paragraphs expressing the basic content of the document.

Structures of nodes often can grow too large to be seen on one screen. The software includes tools that provide a schematic overview of the structures, along with the capability to roam and zoom in the conceptual space. Beard and Walker (1987) describe these navigational techniques.

The target users for this software system are technical professionals. To meet their needs and expectations, the software has fast response time and operates with a common graphical user interface that can be found on many current desktop microcomputer systems—the Microsoft *Windows* graphical environment.

1.1.3 The Tools for Data Collection and Analysis

A protocol, in cognitive psychology, is a report of the steps performed by a subject in attempting some task. In psychological testing, protocols are usually verbal reports given by the subjects themselves, with additional annotation consisting of observations by the tester(s). The verbal reports are either recorded at the time of the test as the subjects speak aloud their thoughts and intentions, or recorded retrospectively—that is, at some time after the test—based on some prompting by the tester. Audio and video recordings are also used to capture many aspects of an interaction with greater fidelity. More recently, the interactions between a human and some computer systems have been recorded and stored by the computer in its memory.

For this project, an automated tracker recorded writers' actions in a separate file concurrently with the operation of the structure editor. For each action, the tracker recorded the particular action, its essential parameters, and a timestamp. Two types of trackers have appeared for commercial interactive computer systems: 1) "application-aware" trackers, such as those in spreadsheets and word-processing software packages, that record information specific to the capabilities of the software, and 2) "application-independent" trackers, that capture keystrokes and mouse movement without knowledge of their effect. The tracker created for this study was "application-aware," with these differences from the macro recording facilities available in some spreadsheet and word-processing packages:

- the recorded actions are at a higher level of granularity than basic keystrokes,
- additional parameters, such as time and spatial location for actions, are recorded, and
- all the actions for an entire session are recorded.

An example of a portion of a protocol record is shown in Figure 1 on page 5.

```
+--------------------------------------------------------------------------+
| Start     Stop      Time  Operator         Parameters                    |
|min:sec   min:sec    sec                                                  |
+--------------------------------------------------------------------------+
03:46.52  03:47.46   0.94  PAUSE
03:47.46  03:47.62   0.16  CreateNode       ID(1) StartPt(436, -225)
03:47.62  03:50.09   2.47  PAUSE
03:50.09  03:50.26   0.17  CreateNode       ID(2) StartPt(129, -53)
03:50.26  03:52.62   2.36  PAUSE
03:52.62  03:57.67   5.05  EditLabel        ID(2) NewText(`xx xxxx`)
03:57.67  04:00.37   2.70  PAUSE
04:00.37  04:03.50   3.13  EditLabel        ID(1) NewText(`xxxx`)
04:03.50  04:11.57   8.07  PAUSE
```

Figure 1. A protocol record generated by the tracker used in this study. This portion of a protocol record shows two nodes being created and labeled. The nodes here have identifiers (IDs) 1 and 2, and the coordinates of the nodes' centers is shown. The text of the labels is x'd out by the tracker for anonymity.

A grammar was formulated to analyze the protocol data of a subject's session with the testbed system. An analytic model, embodied in the grammar, views each session as being divided into phases. Each phase lasts several minutes; a single kind of activity predominates in a phase. A phase consists of a sequence of cognitive tasks, called episodes. A cadence of alternating work-and-rest occurs within a phase: episodes of constructive work alternate with periods of reflection and housekeeping. These intervening periods delimit the constructive episodes. At a lower level, episodes are composed of sequences of individual user commands and pauses. Each of these lasts but a few seconds or less. Thus, these four elements—sessions, phases, episodes, and commands—can be viewed as forming a hierarchy of user behavior.

The protocol elements collected by the tracker were specified as terminals in the grammar. In this project, these elements are the basic system actions used to manipulate nodes, trees, and subtrees in a two-dimensional display space. These include user interface commands for creating a new node at the currently-specified coordinates, moving a node, deleting a node, linking a pair of nodes, and editing a node.

The grammar was expressed as an augmented transition network (ATN). ATN grammars were developed by Woods (1970, 1980) as a framework for parsing natural language. ATNs augment finite state machines by maintaining a set of registers that can store information in addition to state and stack information. Also, arbitrary computational tests and actions may be associated with the state transitions. The actions may extend to any function, providing Turing Machine equivalency (*i.e.*, Type 0 power in the Chomsky hierarchy).

The grammar was represented as a set of rules that could be executed as a computer program. This computer program, or parser, automates the kind of analysis performed by Card, Moran, and Newell. The result of running the parser with one protocol record was a parse tree that hierarchically organized the major sequences of human-computer interactions that occurred during a single session. An example of a parse tree created by the parser for this project is shown in Figure 2 on page 6. Card, Moran, and Newell manually generated a similar parse tree in their single-user study. To test that representative parse trees could be reliably produced for arbitrary sessions, an experimental study was devised with a goal of collecting 100 recordings. This study is described next.

```
Fri Sep 09 09:23:14 1988, File: C: RCD071C.TMP
|  New Workspace [35.87 secs, 9 ops]
|  |  Refocus [35.87 secs, 9 ops]
|  |  |  pause [17.14 secs]
|  |  |  Reset the Drawing [0.05 secs]
|  |  |  pause [0.66 secs]
|  |  |  Reset the Drawing [0.06 secs]
|  |  |  pause [8.18 secs]
|  |  |  Move the Map Window [0.06 secs]
|  |  |  pause [0.16 secs]
|  |  |  Show the Map Window [0.06 secs]
|  |  |  pause [9.50 secs]
|  Exploration [86.18 secs, 4 ops]
|  |  Created a solo node: 1 [0.11 secs, 1 op]
|  |  |  Create node 1 [0.11 secs]
|  |  Help Request [86.07 secs, 3 ops]
|  |  |  pause [7.19 secs]
|  |  |  pause [28.40 secs]
|  |  |  Help: Editing Labels & Nodes [50.48 secs]
|  Define Hierarchies [89.47 secs, 16 ops]
|  |  Created a new tree: (1 -> 2 3)  and ...
|  |  Edited existing nodes [59.32 secs, 13 ops]
|  |  |  Label node 1 [19.39 secs]
|  |  |  pause [4.06 secs]
|  |  |  pause [2.86 secs]
|  |  |  Create node 2 [0.22 secs]
|  |  |  pause [4.66 secs]
|  |  |  Create node 3 [0.11 secs]
|  |  |  Label node 3 [11.81 secs]
|  |  |  pause [2.58 secs]
|  |  |  Label node 2 [5.06 secs]
|  |  |  pause [3.02 secs]
|  |  |  Link nodes (1 -> 2) [1.87 secs]
|  |  |  pause [1.86 secs]
|  |  |  Link nodes (1 -> 3) [1.82 secs]
|  |  Cleanup and Take Stock [30.15 secs, 3 ops]
|  |  |  pause [9.39 secs]
|  |  |  Tidy the workspace [0.05 secs]
|  |  |  pause [20.71 secs]
|  Exploration [17.25 secs, 3 ops]
|  |  Created a solo node: 4 [9.56 secs, 2 ops]
|  |  |  Create node 4 [0.22 secs]
|  |  |  Label node 4 [9.34 secs]
|  |  Pause [7.69 secs, 1 op]
|  |  |  pause [7.69 secs]
|  Top Down Construction [24.99 secs, 4 ops]
|  |  Hooked existing nodes to a tree: (1 -> 4)  [0.71 secs, 1 op]
|  |  |  Link nodes (2 -> 4) [0.71 secs]
|  |  Cleanup and Take Stock [24.28 secs, 3 ops]
|  |  |  pause [4.07 secs]
|  |  |  Tidy the workspace [0.10 secs]
|  |  |  pause [20.11 secs]
```

Figure 2. An example parse tree. The rightmost column shows the commands and pauses. Adjacent pauses, as occur twice above, can occur when a user presses a key or mouse without completing any command. The 2nd column from the right shows the grouping of the commands and pauses into episodes. The 3rd column from the right shows the phases of activity. The leftmost column shows the session's date and time.

1.1.4 The Experimental Study: Examine 100 Sessions of System Usage in Actual Settings

The idea of automated collection and analysis of software protocols needed to be tested using real users to see whether it was feasible and provided useful information. We designed the study carried out in this project to collect and analyze approximately 100 sessions from about 20 subjects. This study had three goals:

1. To broaden the knowledge of writers and their cognitive habits,
2. To provide insight to future builders of highly-interactive software systems, and
3. To consider the apparatus and procedures used to accomplish the first two goals.

Automated tracking allowed me to distribute the testbed system to subjects for use in their actual field settings. After using the software for writing tasks, the subjects returned the protocol records of their sessions. As an IBM employee, I was able to distribute the software to IBM employees worldwide via the corporate communications network. The subjects were thus adult professionals who routinely do expository writing. This afforded a relatively homogenous group of subjects representative of the professionals to which such a writing system would be targeted. The subjects already had the hardware and software required to use the system. I provided a user's guide that itself served as a demo of how to use the system.

The study was exploratory in nature, with few experimental controls.

- Although the testbed system generated protocol records automatically, subjects were not compelled to return these recordings.
- Subjects had no prescribed topics to write about in their documents. The documents did not have to fit a designated format, *e.g.*, a memo, short story, or technical article.
- Subjects returned their protocol records, but not the actual documents they were writing. There was no knowledge of the content of the actual documents being produced. There was not even a requirement that actual work be done in a session.
- There was no restriction on the knowledge and experience of the subjects. All subjects were adult professionals. Otherwise, the study was blind to matters of age, sex, race, and nationality.
- There was no restriction on the computer hardware used by a subject. The speed of a subject's computer and its screen size were two of the many variables that varied among subjects.
- There was no control on the time duration for a session or on how the time was spent in a session. Subjects may be simultaneously working on any number of other tasks.
- There was no restriction on the number of sessions per subject, on the number of documents per subject, on the number of documents per session, or on the number of sessions per document.

At the end of the study period, I had collected 112 session protocols from 29 subjects. The focus of the data analysis was on descriptive statistics to discover patterns. This is in contrast to rigorous tests to make fine distinctions, which, at this point, are premature. This project is an early step in understanding the problems and issues outlined in this introduction. The results suggest areas where controlled, follow-up studies would be beneficial.

1.2 Research Issues

The research issues that we addressed cover the main areas of the project. What can we learn about human behavior with this software? What can we learn about the tools and methods we used to examine that behavior? What can we learn about improving the design of a given software system? As an exploratory project, what topics are good candidates for future lines of research? A list of the particular issues we pursued follows, divided into the three main areas:

Human Behavior and Patterns of Use

- How do people use this kind of writing system? What patterns are revealed?

Methodology for Studying Subjects and Their Software Usage

- ***Protocol Collection***
  - Can machine-recorded protocols provide rich and interesting data?
  - What is the right level of granularity for these data?
- ***Analysis and Categorization***
  - Can a grammar be described to capture the essential elements of the human-computer interaction?
  - What power of grammar is needed to describe the session protocols?
  - Can parsing tools be built so that an analyst is not buried in data?
  - What do the tools address that we can't get at otherwise? What kinds of questions can be answered using these protocol analysis methods?
  - How robust is the parser? How is robustness to be measured?
  - What can we see in the analyzed data versus the raw data? How do we know if we find something interesting?
  - How can the grammar be validated?
- ***Experimental Study***
  - What did we learn about this kind of study and future similar studies?

Feedback for the Software Development Process

- What did we learn about building this kind of system?
- What does the data confirm about the system?

The next section highlights major results from each area of the project. Later chapters, particularly Chapters 5, 6, and 7, provide greater detail.

1.3 Major Results

As an exploratory project, we made a variety of observations on a cross-section of the topics encountered. The following give the general flavor of these observations.

1.3.1 Human Behavior and Patterns of Use

1. Across all the sessions, users' work tended to fall into categories, where the categories describe different kinds of documents and sessions.

    - 38% of the 112 sessions examined were trivial or unproductive. In many sessions, subjects were clearly learning and exploring this new software.
    - In the remaining sessions, subjects frequently used the systems in ways not anticipated.
    - Fewer than 10% of the sessions included extensive writing within the nodes of the trees (aside from labeling of the nodes).

2. In sessions with both planning and writing, these activities were highly intermixed. Only one session showed all planning done before any writing.

3. There was a preponderance of top-down document construction, as opposed to bottom-up construction. In top-down construction, the root of a tree and its children are created and labeled early, with the rest of the document construction consisting of creating new nodes and adding them to the bottom of the evolving tree. In bottom-up construction, nodes are created and labeled before they are linked; small trees are linked into larger trees. The ratio between top-down and bottom-up was about four-to-one.

4. Sessions with extensive writing lasted about an hour. Sessions without writing lasted about 25 minutes.

    About half of all the time in the protocol records was spent in pauses. The median pause time, across all sessions, was three seconds. Two-thirds of all pauses were less than five seconds.

5. Overall, document trees had about 3 to 5 levels and 15 to 30 nodes. Documents constructed across several sessions were smaller than documents constructed in a single session.

6. Half of the commands executed in all the sessions can be accounted for by just three commands: creating, labeling, and linking nodes. The other 36 commands available in the software accounted for the other half.

1.3.2 Methodology for Studying Subjects and Their Software Usage

Protocol Collection

1. It was easy to incorporate a tracker in a software system without degrading performance.

2. The internal tracker used a synchronous interface, which did not perceptibly slow response time and simplified the amount of state information needed to identify each event.

3. The tracker recorded events at the command level, *e.g.*, "Create a node." If every keystroke and mouse movement had been tracked, system performance would have been slowed and the recording file would have been several times longer.

4. Subjects produced about 23 Kbytes or 7 pages of protocol data per hour with this system.

5. Networking proved a powerful tool for software development, distribution, and protocol collection.

    - I was a developer of this Microsoft *Windows* application before there was adequate documentation. I exchanged programming questions with Microsoft *Windows* developers via the *GEnie* network, where Microsoft supported a bulletin board service.

- The testbed software used in this project was distributed for a year via IBM's internal network before protocols were solicited. This allowed me to fix bugs and make improvements based on users' feedback over the network. I was confident the system was stable when I commenced the study.
- The study was distributed to subjects in their actual field setting via the network. The protocols were returned to me over the network.

6. File management became a problem as protocol files with arbitrary names were collected from different user's machines.

   As the tracker was implemented, it invented filenames unique to the user's machines. These names were no longer unique when all the files from all the users were stored on one machine for analysis. Also, users had occasion to change their protocol filenames before returning them to me for analysis.

Analysis and Categorization

1. Managing protocols is an important part of this approach. Collecting and analyzing protocols as done in this project scaled up to about 100, but cataloging and file management then became a predominant problem. Easy categorization of groups of sessions, documents, and subjects is necessary as the number of sessions increases. A database management system is needed to manage large numbers of session protocols.

2. Sessions varied from one to another among many variables. Sessions could be grouped by looking across any of these variables, or they could be characterized by looking across many variables in one session.

3. Generating automatic session summaries and saving these in spreadsheet format simplified much of the analysis.

4. A multi-pass parser allowed intellectual manageability and frequent refinement.

Experimental Study

1. This method of software distribution and protocol collection allowed a large amount of software usage data to be collected in a short period. The costs per session were fixed, and low, after preparing the tools.

2. Of all 210 potential subjects that were given copies of the testbed software, user's guide, and cover letter (printed in Appendix A on page 163), 14% of them returned one or more protocol recordings. The tracker was implemented to avoid writing any binary or confidential information; users could browse through any protocol file with a simple ASCII text editor. I think this gave users confidence that no secret information was being collected about them.

3. Giving users a mechanism for inserting comments into the protocol record proved valuable. Users reported problems and wrote comments in the precise context where they occurred. This feature should be considered for all software.

4. With few experimental controls, many sessions reflect merely training and learning. Much like any harvest, this easy method of protocol collection requires additional methods to filter out the chaff.

- The testbed software used in this project was distributed for a year via IBM's internal network before protocols were solicited. This allowed me to fix bugs and make improvements based on users' feedback over the network. I was confident the system was stable when I commenced the study.
- The study was distributed to subjects in their actual field setting via the network. The protocols were returned to me over the network.

6. File management became a problem as protocol files with arbitrary names were collected from different user's machines.

   As the tracker was implemented, it invented filenames unique to the user's machines. These names were no longer unique when all the files from all the users were stored on one machine for analysis. Also, users had occasion to change their protocol filenames before returning them to me for analysis.

Analysis and Categorization

1. Managing protocols is an important part of this approach. Collecting and analyzing protocols as done in this project scaled up to about 100, but cataloging and file management then became a predominant problem. Easy categorization of groups of sessions, documents, and subjects is necessary as the number of sessions increases. A database management system is needed to manage large numbers of session protocols.

2. Sessions varied from one to another among many variables. Sessions could be grouped by looking across any of these variables, or they could be characterized by looking across many variables in one session.

3. Generating automatic session summaries and saving these in spreadsheet format simplified much of the analysis.

4. A multi-pass parser allowed intellectual manageability and frequent refinement.

Experimental Study

1. This method of software distribution and protocol collection allowed a large amount of software usage data to be collected in a short period. The costs per session were fixed, and low, after preparing the tools.

2. Of all 210 potential subjects that were given copies of the testbed software, user's guide, and cover letter (printed in Appendix A on page 163), 14% of them returned one or more protocol recordings. The tracker was implemented to avoid writing any binary or confidential information; users could browse through any protocol file with a simple ASCII text editor. I think this gave users confidence that no secret information was being collected about them.

3. Giving users a mechanism for inserting comments into the protocol record proved valuable. Users reported problems and wrote comments in the precise context where they occurred. This feature should be considered for all software.

4. With few experimental controls, many sessions reflect merely training and learning. Much like any harvest, this easy method of protocol collection requires additional methods to filter out the chaff.

1.3.3 Feedback for the Software Design and Development Process

1. The system had elements of functional overkill: some of the features were rarely used. In particular, functions without direct mouse/menu interface were rarely used by the subjects. By count, half of all the commands could be accounted for by only 3 of the 39 possible commands.
2. Subjects used the novel "TidyTree" command frequently: this allowed them to place nodes wherever was comfortable, and then later ask the system to make the layout more structured.
3. Subjects rarely saved their work during a session; an ongoing auto-save function could avert disasters for some users. The tracker itself had no checkpointing; if a session was lost, so was its recording.
4. The protocol analysis readily identified help panels that were used frequently, and showed on which helps the most time was spent. The context where help was needed could be easily seen. Frequently-requested helps indicate obvious candidates for functions that need to be simplified.

The preceding set of observations are highlights of the extended descriptions found in Chapters 5, 6, and 7. They suggest that the project was fruitful; it showed many results that were not expected when the software and study were designed. The study met its goals: much was learned about writers and how they used this software, much was learned about how to design and implement these types of tools to automate the study software usage, and feedback was obtained on how to improve the specific software under study.

1.4 Preview of Remaining Chapters

Chapter 2 on page 13 is a review of the literature in the three principal research areas. It discusses the research basis behind the testbed system and the protocol collection and analysis tools. This chapter also provides background information on structure editors and related hypertext systems.

Chapter 3 on page 32 describes how the testbed writing system looks and feels. It introduces each of the commands available in the software.

Chapter 4 on page 49 describes the implementation of the tracker and analysis tools.

Chapter 5 on page 93 tells what we learned about writers and their cognitive behavior by analyzing 112 protocol records.

Chapter 6 on page 129 presents lessons learned in constructing and using the automated analysis tools.

Chapter 7 on page 143 offers some insights gained from this exercise into the development of interactive software.

Chapter 8 on page 151 summarizes the observations and discusses directions for future research.

Appendix A on page 163 contains the cover letter I sent to each of the 210 potential subjects for this study. Three extensive tables summarize the 112 protocol recording files.

Appendix B on page 170 steps through each pass of the parser, starting with an example protocol record and showing the output from each pass and the final parse tree and summary file.

Appendix C on page 205 describes each of the command-line parameters used by each of the six passes of the parser.

CHAPTER 2. RELATED RESEARCH

This project spans multiple areas of research, taking a multi-disciplinary approach to accomplishing the five steps described in Section 1.1. This chapter surveys related research in three areas:

1. Theories and studies of reading and writing.
2. Computer systems that assist writers in organizing and structuring their ideas.
3. Methods for collecting and analyzing protocols of human usage of computer systems.

In its examination of reading and writing, this chapter begins by looking at human memory and theories about its organization. Reading comprehension is seen not simply as interpreting text, but as a process of altering the reader's existing memory organization. Writing research is then examined, looking at basic research in the representation of ideas, the development of ideas, and the writing process itself. A more detailed discussion is provided for one specific aspect of writing theory—cognitive modes—that was developed in our research group and has served as a rich source of ideas—some adopted, some not—for this project.

With this background of the mechanisms employed by writers, the next area reviewed is computer systems designed to assist writers. Specifically examined are tools used to plan and write technical documents. The features of representative systems are surveyed, along with the motivation for designing the new system built as part of this project.

As system builders, we want to test the ideas we incorporate in our systems. Consequently, the final section of this chapter discusses tools and techniques to collect, analyze, and interpret data—called protocols—that reveal users' interactions with computer systems. The section reviews traditional methods of collecting protocols, mentioning their strengths and limitations. It reviews issues of protocol analysis, looking at several models that have been developed. This discussion provides the background and motivation for developing the grammar and other analytic tools used in this project.

Thus, three areas of research are combined in this multi-disciplinary project: reading and writing, writing systems, and protocol analysis. The discussion that follows in this chapter reviews relevant research and concepts used in this project.

2.1 Understanding Reading and Writing

The first body of work to be reviewed is that concerned with the processes of reading and writing. It is relevant for this project because the computer program developed as a testbed system supported the task of writing expository prose.

The discussion begins by reviewing current perspectives on the function and organization of human memory. Next, it shows how these properties affect reading comprehension. Finally, it discusses writing from the perspective of processes and strategies for transforming concepts stored in human memory into documents that can be understood and, in turn, committed to memory by readers.

2.1.1 A Look at Human Memory

Only observed behavior of human memory is considered here; the information-processing model discussed here does not describe the physiological operations of the human neural-motor system. Brain physiology research is presently too low-level and inconclusive to describe the cognitive workings of a writer organizing the elements of a document. Consequently, the model of human memory presented here is based on cognitive theories that are still undergoing development and elaboration in psychological research.

Card, Moran, and Newell (1983) proposed a "human information-processing system" that can be divided into three interacting subsystems: the perceptual system, the cognitive system, and the motor system. They described each subsystem as if it had its own memories and processors, analogous to those in a computer:

- The perceptual system consists of sensors and associated buffer memories, principally brief sensory-image storage for the visual and auditory systems. These hold the output of the sensory system while it is being coded into symbols.
- The cognitive system receives symbolically-coded information from the sensory-image storage and uses previously-stored information to make decisions about how to respond.
- The motor system carries out the response.

A widely-held view of the cognitive system proposes two principal memories: Working Memory, which holds the information under current consideration, and Long-Term Memory, which stores knowledge for future use. Working Memory holds the intermediate products of thinking and the representations produced by the perceptual system. Working Memory is characterized by rapid access times (in the 200 msec range), but small capacity. Long-Term Memory is characterized by much slower access and store times (often on the order of seconds), but essentially unlimited capacity.

Long-Term Memory can be viewed as a structure having two components. In analogy with a computer database system, Long-Term Memory can be said to have both a database and an index. The information in the database, contained in the individual records, is accessed rapidly using the pointers that comprise the index.

The elements of Long-Term Memory can also be seen as an associative structure--a system of concepts interconnected by numerous links (Simon, 1979). Information can be retrieved from Long-Term Memory not only via the index but also by following paths of links from one concept to another through intermediate concepts. Retrieval using the index is called <u>recognition</u>; retrieval using sequences of links is called <u>association</u>. The latter process is considerably slower than the former.

Working Memory is said to consist of elements, called <u>chunks</u>, which may themselves be organized into larger units. The use of chunk as the unit for Working Memory was proposed by Miller (1956) in his "Magical Number Seven" paper. The relevance of this unit for measuring fixation in Long-Term Memory was later affirmed by Simon (1974). Recent research reaffirms the effective capacity of Working Memory as indeed "seven plus-or-minus two" chunks (Card, Moran, and Newell, 1983).

A seven-chunk limit would not seem to support the complex sort of information processing performed, for instance, by chess grandmasters who can play 50 games at once or by writers organizing the many concepts in large documents. By chunking, sets of items are bound together conceptually to form a unique, but abstract, item. For instance, rather than having to memorize the geographic relationships among 11 football players, one can simply say, "They are in a wishbone-T formation." The concept "wishbone T" is a chunk that summarizes a large amount of information about the positioning of different players.

The crucial assumption about chunks is that the contents of Working Memory are symbols that give access to, or point to, corresponding concepts in Long-Term Memory. Thus an act of recognition consists of using the index to retrieve such a symbol and to store it in Working Memory; an act of recall from Long-Term Memory consists of placing in Working Memory the symbol designating a particular concept in Long-Term Memory.

Card, Moran, and Newell observed that storing new chunks in Long-Term Memory requires a fair amount of time and several Long-Term Memory retrievals, because remembering something usually requires building links to existing items. Successful retrieval of a chunk from Long-Term Memory depends upon whether associations to it can be found. There are two reasons the attempt to retrieve a chunk might fail: 1) effective retrieval associations cannot be found, or 2) similar associations to several chunks interfere with the retrieval of the target chunk. Items cannot be added to Long-Term Memory directly; rather, items in Working Memory (possibly consisting of several chunks) have a certain probability of being retrievable later from Long-Term Memory. The more associations the item has, the greater its probability of being retrieved. If someone wants to remember something later, his best strategy is to associate it with items already in Long-Term Memory, especially in novel ways so there is unlikely to be interference with other items.

On the other hand, Long-Term Memory can be accessed on a cognitive-processing cycle of about 70 msec. Thus, the human memory system operates as a fast-read, slow-write system. This asymmetry makes the limited capacity of Working Memory critical for many tasks, because it is not possible in tasks of short duration to transfer much knowledge to Long-Term Memory as a working convenience.

Each concept in Long-Term Memory, then, together with the links connected directly with it, is said to constitute a chunk of information, while the symbol that designates such a concept is said to constitute the corresponding chunk in Working Memory. If Working Memory can hold some fixed number of such symbols, then it will have a fixed capacity measured in chunks. Learning involves both storing new concepts and links in the database portion of Long-Term Memory and elaborating the index to increase its powers of discrimination and recognition.

An important aspect of writing is the process of retrieving or generating the relationship among concepts in memory and presenting them in a form where they can be understood and remembered by readers. The writing system used in this project was designed to assist writers with this task by providing them with tools to externally represent their internal memory organization.

2.1.2 Organizing Concepts in Memory

How do people organize concepts that they have learned? Can they or do they discover inherent relationships in the information and employ this found structure? Do they impose a standard structure on all concepts regardless of their inherent organization? These questions are fundamental for understanding human memory structures and processes. They also have practical significance for deciding how information is to be represented in so it can be well understood by others.

Studies of human memory lead one to conclude that memory is organized so that information can be accurately and quickly retrieved, new concepts can be easily fitted into existing structures, and existing structures can be modified by or accommodated to new experiences (Durding, Becker, and Gould, 1977). For example, several different organizational schemes have been proposed as the basis for providing these high-level capabilities.

- Smith, Shoben, and Rips (1974) have proposed that a structure composed of attribute lists can account for many of the observed phenomena of memory.
- Collins and Quillian (1969; 1972) found that a model based on hierarchical structure accounted for most of their reaction time data.

- Anderson (1972) was able to simulate several aspects of human performance using a model based on associative networks.

Additional experimental support has accrued to each of these proposed models, but there is no consensus that a single model best describes how memory is functionally organized. The type of memory organization people use probably depends on task conditions. Certainly, people can mentally represent the same information in more than one way by using different memory encodings, depending on their purposes (Posner and Warren, 1972).

The writing system designed for this project proposes to assist writers in organizing memory concepts externally. The model chosen for its design incorporates a hierarchical structure, while also accommodating some aspects of an associative network model. This meets the goal of the system, which was to assist in writing expository prose—which necessarily has an underlying hierarchical structure. The research cited here suggests that humans work in more flexible ways than that of a single model; future writing systems should consider designs with the flexibility to accommodate a range of organizational models.

An alternative view is that memory organization is a function of the retrieval strategies used in a particular situation (Landauer, 1972). In this view, the problem solving requirements of a task are the primary determinants of the organizational structures observed by experimenters. This leads to a shift in research emphasis from the structure of memory to the processes that are responsible for organizing the retrieval strategies.

Whereas writing is a process of retrieving items from Long-Term Memory, reorganizing them, and translating them into words, reading is a process of receiving items from an external medium, inserting them into Working Memory and then into Long-Term Memory. To increase the comprehension of their written material, writers should try to trigger the right organizational processes of the reader. We next look more closely at reading, to identify principles that may be used by writers as well as those who build computer systems to help them.

2.1.3 Reading Research

Current theories of reading comprehension suggest that readers mentally encode what they read into a form that is different from the literal text. They incrementally integrate portions of that alternative version into their Long-Term Memory (see, for example, Kintsch and van Dijk, 1978; Lachman and Lachman, 1979a; Schank and Abelson, 1977).

While opinion varies on the precise nature of the encoding, Greeno (1977) has listed three criteria for "good comprehension" that can be applied broadly: coherent, connected representation of content; correspondence between the representation and the text; and connection between the components of the message being comprehended and the reader's general knowledge. As these criteria suggest, comprehension involves several different cognitive processes and takes place on several different levels with respect to memory. For example, during comprehension, a reader remembers units of information that include words, sentences, paragraphs, and so on—up to the entire text—while constructing the encoded version that will be integrated into Long-Term Memory (Voss, Tyler, and Bisanz, 1982). Thus, features that help a reader remember one point while reading another facilitate comprehension.

The theories of propositions advanced by Meyer and Kintsch are particularly relevant (Kintsch, 1974; Meyer, 1975; Kintsch and van Dijk, 1978). These researchers presume that a reader constructs a representation of the text content, called a text base, that is different from the sequence of the printed words. While the semantic content of a discourse is composed of an ordered list of propositions (Meyer, 1975), the content structure is hierarchical. Meyer showed that by selecting a major superordinate idea and then relating subordinate ideas to it, one can construct a tree-diagram representation of the content structure of the text. She and others have also shown that recall of a proposition by readers is significantly affected by the position of that proposition in the hierarchy: propositions high in the tree structure are recalled better than propositions lower in the structure (Meyer, 1975; Kintsch and Keenan, 1973; Britton, Meyer, Hodge, and Glynn, 1980).

The process by which a reader constructs the individual links in a text base has been clarified by Kintsch and van Dijk (1978). A coherent text base can be represented by a connected graph. The reader synthesizes the text base using a step-by-step process in which propositions in a sentence are related to one another. Referents from some propositions carry over from one sentence to the next. Because Working Memory can retain only a few propositions at a time, the reader first attempts to connect a new proposition to one already in Working Memory. If a link is made, the new text being processed is perceived as coherent with the text just read. If not, an inferential bridging process is initiated to locate a similar proposition in Long-Term Memory and place it in Working Memory. Inference and Long-Term Memory searches are costly, since they do not easily fit in the small capacity of Working Memory. As might be expected, the inferential bridging process considerably slows comprehension (Kintsch and van Dijk, 1978).

Several features of a text can contribute to the efficient construction of the text base hierarchy. Thematic titles presented prior to a well-structured text significantly increase free recall of the content of that text (Schwarz and Flammer, 1981). Similarly, advance organizers, or passages containing the main concepts of a text but at a higher level of abstraction, positively affect comprehension (Ausubel, 1963). Texts in which the hierarchical structure is signaled or cued are comprehended more effectively than texts in which the hierarchy is not signaled (Meyer, Brandt, and Bluth, 1980). At the paragraph level, inclusion of a topic- or theme-sentence in the initial position, rather than in an internal position or not at all, results in more accurate comprehension (Kieras, 1980; Williams, Taylor, and Ganger, 1981). Thus, clear signaling of the writer's intended hierarchical structure of concepts through typographic and rhetorical conventions strongly influences the reader's comprehension and the associated process of constructing a hierarchical text base.

Knowledge of the kinds of cues that help readers understand and remember what they read can help developers of writing systems to design their systems so that they encourage writers to produce documents that include such cues. For example, a system intended for writers of expository prose might encourage them to produce hierarchically-structured documents with descriptive headings. Indeed, this was the strategy used for the testbed system described later in this paper.

2.1.4 Writing Research

While rhetorical forms and prescribed structures for writing have been described for hundreds and even thousands of years, most research into how writers compose expository prose dates from the past twenty years. These studies have focused on two major areas:

- how writers internally and externally represent the material they want to present, and
- what mental processes writers employ in producing a completed manuscript.

2.1.4.1 Representation and Writing

In 1980, Hayes and Flower used think-aloud protocols to examine a number of writers from the earliest stages of preparing a document through its completion. These protocols showed articulated plans and goals that never appear in the final text of the document, and in fact soon disappear from the writer's recollection of composing. They found that as writers compose, they create multiple internal and external representations of meaning. Hayes and Flower conjectured that a variety of forms of representation were necessary because the organization of knowledge and meaning within human Long-Term Memory and Working Memory differs considerably from the eventual organization presented by a writer in expository prose.

An important issue raised by these researchers is how people organize concepts, both internally and in the external representation they are constructing. Writers often organize their documents "in their heads." Current cognitive theory on the operation of human memory, however, suggests just two ways of accomplishing this task internally:

1. Writers create in Long-Term Memory a separate cognitive structure for the concepts to be conveyed. This is a slow process, prone to error and omission, and requires a great deal of "mental effort."

2. Writers organize in Working Memory the concepts to be conveyed. A major obstacle with this approach is that writers cannot hold all their concepts in Working Memory at one time. While they are organizing one set of concepts, another set slips out of consciousness.

By organizing externally, writers can potentially overcome these memory limitations and more efficiently convey to their readers the desired set of concepts and their relationships. However, complex ideas and relationships cannot always be represented by a short phrase or even a paragraph. Consequently, writers often struggle to develop alternative ways to express their internal cognitive structures externally.

2.1.4.2 Concepts

Researchers in cognitive structure describe several different types of concepts in the literature. There is general agreement on the central definition of each type and little agreement on the ambiguous areas between the types. Flower and Hayes (1984) observed that writers work with three different types of representation of meaning: verbal, procedural, and imagistic representations. White (1985) similarly described these as propositions, algorithms, and images:

> Propositions are representations in memory of facts or beliefs. Their prominence as the basic unit in many models of cognitive structure is easily explained: they are the basis for writing and speaking; they are a conveniently-sized unit; their existence can be readily tested.
>
> Algorithms and skills are step-by-step procedures for solving a problem or accomplishing some task. West *et al.* (1985) use the term algorithm when they refer to public knowledge and skill when they refer to private understanding: "This is not just a matter of semantics. Books cannot have skills. They can only outline the steps of an algorithm that can be used to perform a particular task. A person who can perform that task, possesses that skill, whether one follows the book algorithm or not."
>
> Images are mental pictures. In recent years there has been debate concerning the existence of separate storages in the brain for images and propositions. Because the ability to form mental pictures is universal and because real pictures have long been seen as a powerful mode of communication, images can well be considered as an independent type of concept.

Concepts must comprise both general knowledge and specific knowledge. For example, the definition of "rock" is a generic definition. The knowledge about specific rocks we have seen is knowledge about particular instances of the general concept. The generic definition applies generally to all rocks, but with some flexibility. Generic definitions describe the typical characteristics of rocks, but any particular individual rock need not follow the generic definition exactly.

Hence, the definition of a concept must have two important properties. First, there is a need for generic information about concepts that provides general knowledge, letting us deduce the properties of instances of the concept, even if we have not experienced those instances. Second, the generic knowledge is prototypical: it specifies typical values, but we are not surprised if particular instances of the concept differ from some of the generic properties. Thus, concepts are not simply organized in relation to one another, they have internal organization as well. Some examples of a concept are more prototypical than others, and typicality greatly affects the speed of recognition and categorization. The meaning of a concept for any person is part of that person's private understanding. Different people use the same concept labels for different internal meanings and relationships. Not only can one label correspond to several concepts, but also some concepts may have no corresponding label.

Writers must thus struggle with two goals: they must externally represent concepts with sufficient information to be meaningful in their context, and they must provide enough information to overcome mismatches between the understanding held by the writer and conflicting understandings already held by potential readers.

2.1.4.3 Representation

Many of the insights discussed in the section above come together in the notion of representation. A representation is an isomorphism, or mapping, between a represented world and a representing world (Palmer, 1978). In writing tasks, the represented world is the writer's multi-dimensional cognitive structure for the document being produced. This represented world changes as the writer progresses from the initial phase of exploration and organization of concepts to the final document structure. The representing world is the medium within which the writer is working, such as pen and paper or a computer display. The goal of any representation is to preserve in the representing world the relationships of importance in the represented world.

Notice that the representing world necessarily reflects a subset of the possible relationships existing in the represented world. For example, the "closeness" of concepts in a writer's associative structure is an example of an aspect that is often included in the subset. Closeness or similarity can be reflected in actual, physical distance relationships in a representation of those concepts. By contrast, the "size" of a concept is often not directly represented in most computer writing systems, although it could be.

Similarly, not all aspects of the representing world model some aspect of the represented world. For example, when using a graphic medium for representation, an author can use location, color, size, shape, connections, texture, and motion to portray relationships. Not all of these are necessarily used in a single representation. When using color, for instance, we often need a key or a legend that maps the significance of each color used in the representation to its counterpart in the represented world.[1]

Models of human memory indicate that a writer starts with loosely-connected sets of concepts and molds them into a final-form document, a sequence of text. Thus, the world represented here has a number of forms. Human memory is commonly modeled as a multi-dimensional semantic network; for a graphic representation, closeness must be reduced to a two-dimensional distance. At the start of the organizational phase of writing, we want to represent the similarity, disparity, and distinctions among concepts, which can be done readily with closeness and explicit connections between concepts. The result of the writing organization phase should be a set of sequences and hierarchies of concepts that can be reduced to the sequence of text expected and well-comprehended by a reader.

The writing system designed for this project aims to meet writers' needs in the three areas just discussed. It presents a place to work with ideas, instead of in their heads somewhere between Working Memory and Long-Term Memory. It allows concepts of the three types discussed in the literature to be captured: propositions, algorithms, and images. And, it permits representation of relationships of ideas—such as linkage, hierarchy, and closeness—in an easily-changeable medium.

[1] As a counter-example, the meaning of "high-intensity" fields and items colored "red" are often readily apparent without a legend.

This discussion of writing so far has surveyed the objects a writer deals with—concepts—and their external organization and representation. The following discussion examines the process by which writing is accomplished. It looks at the steps and thought processes a writer goes through when bringing a piece of technical prose from inception to completion.

2.1.5 The Writing Process

Writing is a complex process that draws on many different cognitive skills. Among these skills are:

- Retrieving information from the writer's memory or from external sources.
- Identifying associative relations among ideas.
- Drawing inferences and making deductions.
- Building large hierarchical structures.
- Translating ideas into words.
- Reading, analyzing, and rewording during the editing process.

Hayes and Flower, in their work reported in 1980 and since, have observed a shift of attention from product-oriented research in writing to process-oriented research. Using think-aloud protocols, they looked at the steps that writers employ in producing expository prose. Their results indicate that Rohman's three-stage description of the writing process (1965)—consisting of pre-writing, writing, and rewriting—was oversimplified. The process that writers use is both iterative and recursive and, hence, more complex.

Hayes and Flower found writing to be goal directed, with the goals often organized hierarchically. Three types of activity are used by writers to accomplish their goals: planning, sentence generation, and revision.

Planning: Writers use pointers, word images, and goals in planning their writing. Pointers can consist of references to complete text or just jotted notes to stir a writer's memory. Word images are fully-written sentence fragments, sentences, paragraphs, or more. Goals are of the form "Add an introduction." Adult planning consists of constructing a complex goal structure:

- Goals form a hierarchy or network.
- Expert writers generate more elaborate networks of goals than novices.
- Priorities are dynamic, frequently changing and evolving over the course of writing.

Sentence generation: Hayes and Flower found this step to consist of explaining briefly-sketched ideas produced during planning, interpreting non-verbal material, and carrying out instructions. They noted that:

- Essays are typically eight times the size of the writer's outline.
- Sentences are generally composed in parts: these parts are 7-12 words in length, separated by pauses.
- Experts write longer essays, with longer sentence parts.

Revision: The more expert a writer, the greater the time spent in revision.

- Experts attend more to global problems than novices.
- Experts detect and diagnose problems better than novices.
- Writers have difficulty detecting faults in their own text.
- Revision can be applied to writing plans as well as written text.

Rather than progressing through distinct stages, writers accomplished their goals in arbitrary sequences of activities; sometimes recursive and sometimes iterative. This complexity of goals was recognized and was a key factor in the construction of the analysis tools used to examine writers' protocols in this project.

2.1.6 Writing Modes

The Textlab group views the processes used by writers as constituents of a set of cognitive modes (Smith, Weiss, and Ferguson, 1987; Smith and Lansman, 1988). This concept of mode includes many of the elements of the writing process described by Flower and Hayes, but in a more structured form. They describe four components that comprise modes:

1. One or more cognitive processes.
2. A product produced and/or operated on by those processes.
3. Goals, representing a writer's intentions when undertaking the associated processes.
4. A set of rules or constraints that govern the kinds of products that can be produced within the mode and the relations that can exist among the parts of the product(s).

Writers use different cognitive modes to produce different forms of information or to transform one intermediate product into another. For an intuitive sense of modes, consider the following examples. During early work on a document, many writers adopt a mode of thinking in which the primary purpose is to identify ideas and data that may be included in the document and to consider various relations among them. The writer retrieves potential concepts from Long-Term Memory or from external sources, considers possible relations among ideas, and, perhaps, groups related ideas and constructs small hierarchical structures. In that mode, the underlying rules are those associated with a network: any idea can be related to any other idea through simple association. Thus, the intermediate product is a network or directed graph of ideas. This exploratory thinking is often creative and unfiltered as the writer generates and considers alternative possibilities for the document.

The mode of thinking used for organizing the content of the document is different. The writer shifts from exploring to building a single integrated structure for the document. Organization is the task of constructing an integrated structure for the document. For many documents, particularly those written by professionals, that structure is hierarchical. Thus, the product is a hierarchical structure and the rules are those that govern hierarchies. That is, each concept in the hierarchy can be subordinate to at most one other concept, but it may be superordinate to many concepts. Building such a structure requires a different set of cognitive processes from those used during exploration. The critical one is the process of abstract construction that includes perceiving subordinate/superordinate relations, comparative levels of abstraction, sequencing, proportion, and balance.

Writing, per se, involves still a different set of cognitive processes. Here, the primary task is encoding the abstractions of content and the relations of the hierarchical structure into a sequence of words, drawings, or other explicit forms. The structure of the encoded text is linear and represents a path through the hierarchy. Consequently, it is even more constrained than organization mode.

Thus, the Textlab group has adopted a multimodal view of writing. Inferences regarding the mental processes of writing are based on the changes writers make to the document they are developing. The modal view has been embodied by their *Writing Environment (WE)* system, which provides a set of windows, each of which encourages a particular modal activity. The complete description of their model of cognitive modes for writing is shown in following table.

Table 1. Cognitive modes for writing. This table is taken from figure 5 of Smith and Lansman (1988).

Modes	*Processes*	*Products*	*Goals*	*Constraints*
Exploration	• Recalling • Representing • Clustering • Associating • Noting subordinate and superordinate relations	• Individual concepts • Clusters of concepts • Networks of clustered concepts	• To externalize ideas • To cluster related ideas • To gain a general sense of available concepts • To consider various relations	• Flexible • Informal • Free expression
Situational Analysis	• Analyzing objectives • Selecting • Prioritizing • Analyzing audiences	• High-level summary statement • Prioritized list of readers (types) • List of (major) actions desired	• To clarify rhetorical intentions • To identify and rank potential readers • To identify major actions • Consolidate realization • To set high-level strategy for document	• Flexible • Extrinsic perspective
Organization	• Analyzing • Synthesizing • Building abstract structure • Refining structure	• Hierarchy of concepts • Crafted labels	• To transform a network of concepts into a coherent hierarchy	• Rigorous • Consistent • Hierarchical • Not sustained prose
Writing	• Linguistic encoding	• Coherent prose	• To transform an abstract representation of concepts and relations into prose	• Sustained expression • Not (necessarily) refined
Editing: Global Organization	• Noting large scale relations • Noting and correcting inconsistencies • Manipulating large-scale structural components	• Refined text structure • Consistent structural cues	• To verify and revise large-scale organizational components	• Focus on large-scale features and components
Editing: Coherence Relations	• Noting coherence relations between sentences and paragraphs • Restructuring to make relations coherent	• Refined paragraphs and sentences • Coherent logical relations between sentences and paragraphs	• To verify and revise coherence relations within intermediate sized components	• Focus on structural relations among sentences and paragraphs • Rigorous logical and structural thinking
Editing: Expression	• Reading • Linguistic analysis • Linguistic transformation • Linguistic encoding	• Refined prose	• To verify and revise the text of a document	• Focus on expression • Close attention to linguistic detail

While I commend the rigor of definition in the Textlab view of cognitive modes, I believe there is a great deal of shifting between the different modes during a writing session. Consequently, the writing system I developed for this study includes only one planning mode and one editing mode. The focus of my analysis of writing strategies is on the sequences of actions and the pauses between them, rather than the shifts between a number of different modes. Inferences regarding the mental processes involved are based on identifiable series of important actions. This is different from the Textlab work, which does its delimiting based on the modes being used and the products being produced. The design and implementation of this new writing system is the topic of Chapter 3.

As we examined the aggregate of reading and writing research—and how key principles might be incorporated in computer writing systems—the members of the Textlab group saw many similarities with existing computer systems that come under the title of structure editors and hypertext systems. Before building new writing systems from scratch, we surveyed existing structure editors and hypertext systems and the techniques used by their builders.

2.2 Tools to Help Writers Organize

The key task for developers of computer writing systems is to help writers with the individual processes, or cognitive modes, that constitute the overall process of writing in order to produce documents that can be read, understood, and remembered. Thus, these systems should be viewed against the background of theory and studies for both reading and writing, as discussed above.

This section reviews a family of systems—structure editors and hypertext—that attempt to help writers, particularly with organizational tasks. It concludes by briefly discussing the rationale for developing a new system as a testbed for this study, and its relation to earlier systems.

2.2.1 A Survey of Structure Editors and Hypertext Systems

The idea of a structure editor predates the computer and can be traced to Vannevar Bush's seminal paper, "As We May Think" (Bush, 1945). Bush proposed a new form of library, called the memex, based on microfilm technology. By including manual cross-references, the memex would help the knowledge worker create a vast network of associations among texts in a private library and to integrate new texts written by the user. While a complete memex has never been built, Bush's vision has guided many significant research efforts and commercial products. Based on the underlying model of text structure, these projects fall into two distinct groups: directed-graph systems and hierarchical systems.

2.2.1.1 Directed-Graph Structure Editors

The first serious attempts to realize Bush's memex in computer technology occurred in the 1960's. Ted Nelson viewed a text as a number of independent logical, rather than physical, segments that could be linked into a network in a variety of ways (Nelson, 1967). His system, Hypertext, was developed on Brown University's S/360 mainframe computer. It was used in an experimental class in literary analysis (van Dam, 1976), but has not been used extensively beyond that. However, van Dam and others (Feiner, Nagy, and van Dam, 1982) at Brown have included many of the features of Hypertext in a dynamic system that combines pictures and text. Implemented on stand-alone hardware with high-resolution graphics, this experimental system was impressive in it flexibility and its effective use of color graphics, but has not been made public for reasons of cost and response time. Another important network-based system is *ZOG*, developed at Carnegie-Mellon University (Robertson, McCracken, and Newell, 1981; Newell, McCracken, and Robertson, 1981; Akscyn and McCracken, 1984). *ZOG* is a high-performance system designed for accessing the massive documentation aboard a nuclear-powered aircraft carrier. While *ZOG* can be used as a writing tool, its primary function is fast, interactive traversal of a network of text components.

The *NoteCards* hypermedia[2] system allows its users to build general semantic networks using representations of 3x5 notecards and typed links (Halasz, Moran, and Trigg, 1987). "Its intended users are authors, designers, and other intellectual laborers engaged in analyzing information, designing artifacts, and generally processing ideas" (Halasz, 1987). Its designers reviewed its design by interviewing twenty of its users. Halasz describes seven key issues that he believes need to be addressed to better match this tool to the needs and preferences of its users. He describes these issues as "an agenda for the next generation of hypermedia systems."

In 1987, Apple Computer began shipping the *Hypercard* system with its Macintosh family of personal computers. Bill Atkinson extended the ideas of *NoteCards* somewhat with his *Hypercard* design; he represented groups of similar concepts as a stack of notecards, where the appearance of entire stacks and individual cards can be graphically tailored. Any card may be made to point to any other card in a stack. Cards may contain any combination of text, graphics, and audio. An interactive scripting language allows users to direct the flow among cards, in addition to automating the contents of a card.

The power and ease-of-use of Apple's *Hypercard* energized both academic and commercial researchers; a wider public became aware of the possibilities of structure editors and hypertext systems. Survey articles by Conklin (1987) and Smith and Weiss (1988) appeared in prominent computer science journals. The *Hypertext '87* conference at UNC brought together many of the key researchers in hypertext systems; the conference proceedings (1987) provide a rich cross-section of the state-of-the-art and the important issues involved in hypertext systems in 1987.

<u>2.2.1.2 Hierarchical Structure Editors</u>

Hierarchical structure editors are, in a sense, a subset of the directed-graph structure editors and hypertext systems discussed above. Systems seeking to support hierarchical text structures follow a similar chronology. Douglas Englebart and his colleagues at the Stanford Research Institute developed an experimental structure editor that also drew ideas from Bush (Englebart and English, 1968). At a landmark demonstration of the system at the 1968 Fall Joint Computer Conference, text, graphics, and live video of Englebart in San Francisco and his colleagues 20 miles away in Menlo Park were superimposed on multiple viewports on a screen, as they were working together and explaining what they were doing. "Chalk-passing" protocols were demonstrated for synchronizing multiple users. This demonstration was a forerunner of graphics- and sound-based teleconferencing.

Seeking to augment the mental capabilities of the emerging "Knowledge Worker," Englebart's group built an extremely powerful and innovative system, variously call *Augmented Knowledge Workshop*, *NLS*, and *Augment*. The structure editor portion created a hierarchy expressed in outline notation (1.1.2, 1.1.3, etc.); by manipulating the outline values associated with blocks of text, the writer could modify the structure of a document. *NLS* introduced the notion of conceptual models for the editing and authoring processes, tree-structured editing, and other features in the field of text editing and office automation. *NLS* and other related systems, such as *HES*, *FRESS*, and *Xanadu*, are important because they view the editor as an author's tool, an interactive means for organizing and browsing through information.

A more general group of experimental hierarchical systems was the *XS* series developed by Burkhart and Nievergelt in Zurich (Burkhart and Nievergelt, 1980; Stelovsky, 1984). These editors have as their core a flexible tree editor that allows the user to manipulate the elements at

[2] The term <u>hypermedia</u> implies that the nodes in the hypertext system may contain not only written text, but graphics, motion video, and/or audio.

the node level. The target-independent tree editor can be combined with target-dependent back-ends to create multiple editors, such as a document editing and formatting system. Intended for a broader range of applications, including software development, *XS-1* and *XS-2* provided flexible, powerful tree manipulation functions on a microcomputer. Innovations included verifying the integrity of the tree after each editorial change to a document's structure and representing the hierarchical structure with alternative forms.

Fraser's *s* is an attempt to provide standard editing primitives that can be used to build a variety of editors; Fraser argues that a generalized structure and a generalized text editor nucleus can be used for editing all applications. Janet Walker's *Document Editor* operates on a document as a collection of files in Scribe manuscript form; it infers the structure of a document from the tags in the file being edited. The specialized functions for technical writing provided by the Document Editor are extensions to the EMACS editor in the form of a user library. Hansen's *EMILY* extended the concept of the structure editor and developed the syntax-directed editor, in which the structure imposed on a program being edited was the structure of the programming language itself.

More recently, commercial "outline processors" such as *ThinkTank*, *Ready!*, *MaxThink*, and *More* have brought some of the features of these more ambitious but private systems to the personal computer. These systems help the writer create an outline, write sections of text within the outline, and manipulate text structure by manipulating the outline.

2.2.1.3 Textlab's Writing Environment (WE)

WE employs both a network (directed graph) and a tree (hierarchical) structure editor—but for different stages, or modes, of the writing process. Writers may work in either editor, developing graphs and hierarchies separately, and they may also explicitly transfer these conceptual structures from one mode to the other.

When designing the *WE* system, the Textlab group reasoned that writing could be viewed as a complex process involving different cognitive processes. A key question in their system design was how best to support these different cognitive modes and the flow of intermediate products among them. They saw two alternatives: 1) a single mode system where all system functions would always be available, and 2) a multimodal approach where the environment was divided into separate system modes, each including only the function appropriate for its corresponding cognitive mode. *WE* follows the multimodal approach.

Consequently, *WE* makes four system modes available at all times, each contained in a separate screen window. These windows are labeled network mode, tree mode, editor mode, and text mode, corresponding to the exploratory, organizational, writing, and editing modes of writing, respectively. Of the modes shown in Textlab table of modes, they did not include a mode for situational analysis, and they included only one mode for editing.

2.2.1.4 Why Build a New Structure Editor?

In the context of writing, a structure editor can cleanly separate the process of generating and organizing concepts from the process of writing and revising an extended text. I chose to design a new structure editor for this study for the following reasons:

- It provided me with the opportunity to design a system based on the ideas I felt were important, rather than on other people's emphases. I chose the alternative of implementing a system where all the organizational functions were always available, not explicitly separated into separate modal groups.
- It provided me with the data I needed to carry out my research objectives. More specifically, it let me "compile in" the "hooks" necessary for an automated tracker, and I could optimize the tracker to be fast and unobtrusive.

- This structure editor could be distributed worldwide without concern for the actual CPU, screen size, or national language support being used. It is implemented as a fast, powerful graphics-based structure editor for an IBM or compatible personal computer. The structure editor runs with Microsoft *Windows*, which provides independence from the hardware being used.
- It could be modified to incorporate the lessons learned in the study.
- It provided a personal testbed for exploring issues of representation and graphics layout algorithms.

The next chapter of this paper describes the structure editor I built in accord with these reasons.

2.3 Protocol Collection and Analysis

In this section I review methodological issues involved in detailed studies of human-computer interaction, including user's strategies. The discussion begins by considering different forms of data, called protocols, and the various concerns and issues raised in collecting them. Then, I shall discuss our particular approach to analysis—based on the concept of a grammar—by reviewing several projects that have used grammars to describe user behavior.

2.3.1 Protocol Collection

To study human behavior, psychologists gather data in the form of protocols. For writing, protocols have historically been verbal reports by the subjects themselves, with additional annotation consisting of observances by the tester(s). The verbal reports were either recorded at the time of the test as the subjects spoke aloud their thoughts and intentions, or recorded retrospectively—that is, at some time after the test—based on prompting by the tester. Further, audio and video recordings capture many aspects of an interaction with a great deal of fidelity. Finally, the interaction between a human and computer can be recorded and stored by the computer. In this subsection, I describe the strengths and limitations of these different means of collecting protocols.

2.3.1.1 Think-aloud Protocols

Think-aloud protocols, where subjects narrate their thought process while performing some task, have been used by researchers in examining the cognitive processes that take place in many complex mental tasks. However, there has been considerable debate about these data, which has been summarized by Smith, *et al.* (1985).

To summarize the issues briefly, Nisbett and Wilson (1977) raised three objections to think-aloud protocols: their validity or accuracy, their completeness, and their possible interference with the task being performed. Ericsson and Simon (1984) have responded to that criticism by identifying three levels of verbalization:

Level 1: Verbalization of concepts already stored in human Working Memory in verbal form,

Level 2: Verbalization of data that would not be heeded as part of the cognitive process, and

Level 3: Verbalization of data that is not part of the cognitive process and must be generated.

They argue that concurrent think-aloud protocols constitute valid data for Level 1 verbalization. They found no evidence that concurrent think-aloud protocols affect this type of cognitive processing or that such data are incomplete or distorted. However, for Level 2 and Level 3 conditions, think-aloud protocols did significantly change the cognitive process, especially recognizing complex patterns and relationships presented visually (Ericsson and Simon, 1984; Henry, 1934).

This is precisely the situation that highly-interactive software presents–the manipulation of complex structural relations and patterns of associations represented visually. Phrased another way, think-aloud protocols are well-suited (or indispensable) for reflective tasks such as reading, but may interfere with or distort generative tasks such as writing–particularly during planning.

Think-aloud protocols are also expensive to use because their coding is labor-intensive. Personnel must be trained to administer the tests and record the protocols. Test subjects must be found who are appropriate users for the target system and who are trained at verbalizing their thoughts while using it. Since such persons are sometimes hard to find, new subjects must generally be trained at the verbalization techniques. Some subjects may need quite a bit of practice. Consistency of verbalization for a given subject and among subjects is difficult to guarantee. An hour of think-aloud protocol typically requires about 15 pages of transcription (Hayes and Flower, 1980).

Test personnel or researchers must usually code the verbalization into some regular format in order for it to be analyzed across sessions and subjects. This coding is often subjective, and can vary with the experience of the coders or even with their mood at the time.

Despite these reservations, think-aloud protocols are used frequently to "get at" users' thoughts and observations while interacting with a software system. Atkinson (1985), for example, used think-aloud protocols to guide refinements in the development of the Macintosh MacPaint program. Tools, such as the *Mini-Protocol Analysis System (MPAS)* of Ericsson and Simon (1984), have been developed to aid those collecting these protocols, by prompting for the next utterance and timestamping the entry. The Playback methodology (Neal and Simons, 1984) also prompts for observer input, and also carries out logging of keyboard activity for later statistical analysis and measurement on a mainframe computer.

2.3.1.2 Video and Audio Protocols

Video equipment has become a powerful tool for recording user interaction with computers. It captures users' reactions (including subtle elements such as eye movement and body language) along with the computer display and users' input actions.

Card, Moran, and Newell (1983) used videotape equipment and tracking programs together to record users' keystrokes and the contents of the screen. These were later combined to obtain the full record of the session, which could be replayed readily by the researcher. However, this method requires special equipment (the video-recorder), possibly an equipment operator, and possibly a special room for testing. This makes the protocol collection process particularly obtrusive, and limits the number of subjects and length of tests that can be administered within a given time period. It also results in a large volume of fine-grained data.

Tater (Mackay, 1988) states that the analysis of videotaped records "is a time-consuming, painstaking affair, requiring perhaps an hour for every minute of tape." Hoping to overcome some of this tedious post-analysis, researchers at MIT (Mackay, 1988) are developing a visual workstation that digitizes moving video and presents it in an X-Window on a high-resolution display. A researcher can create software "buttons" to tag particular events for later analysis. During the session, the output from the video camera is channeled through the X-Window; the researcher can take online notes and tag events as they occur by pressing the appropriate button. After the session, the researcher can watch the complete session at any speed or view just the tagged events. As with the coding of think-aloud protocols, this process requires consistency and coordination among the researchers; for example, identical events must be identically tagged among a group of sessions, if they are to be compared.

Thus, while theoretical issues persist for think-aloud protocols, think-aloud, audio, and video protocols still cause practical problems with respect to effort and consistency of interpretation. An alternative is machine-recorded protocols.

2.3.1.3 Computer Protocols

Software developers have devised a variety of hardware and software facilities to encode and record the input to a computer. However, these have not been designed for protocol collection, but to aid in debugging or to allow a user to review and re-execute previous actions. Among the most familiar is the UNIX history mechanism (Greenberg and Witten, 1988), which records each command line entered by a user. Macro facilities (such as those in *WordPerfect* and Microsoft *Excel*) simplify the re-use of frequently-executed input sequences specific to these products. Standalone keystroke-capturing programs (*e.g.*, *Cocoon*) allow replay of keyboard input independent of the software being run.

The *ICARUS* VLSI circuit-layout system, studied by Card, Moran, and Newell (1983), collected the key name, clock time, and mouse coordinates as part of the protocol. Note that keystroke recording facilities are not usually sufficient to determine the actual command performed in an interactive system. For example, when analyzing session protocols from the *ICARUS* system, Card, Moran, and Newell reconstructed the actual commands that were executed from the keystroke recording using "heuristic programs and hand editing."

Such ambiguities were avoided in this project by recording actions at the command level. Each time an action is selected (say, moving nodes), the start of the action is recorded. When the action is completed (the nodes are released), the end of the action is recorded, along with its parameters (where the nodes were moved to).

2.3.1.4 Comparing Protocol Collection Techniques

Each of the three means of protocol collection discussed above is best suited to different research objectives. Concurrent verbal reports are good for capturing users' short-term view of their strategy and ongoing instances of frustration and delight. Audio and video recordings capture minutiae of the interaction that must still be painstakingly categorized and interpreted. Both of these means have the disadvantage of requiring special equipment and/or personnel at a user's setting.

Verbal reports and audio or video recordings also introduce problems of reliability. For any study, they must be transformed from their raw form into some set of categories in which the model being examined is defined (Swarts, Flower, and Hayes, 1984). Training and practice can increase the reliability and consistency of the transformations, but the encoding is still subjective. Hayes and Flower (1980) report encoding errors on the order of 25%.

To investigate software interactions in volume, what is needed is a means of collection that is:

- adequate–it captures the essential aspects of the interaction,
- consistent–it captures similar interactions similarly,
- unobtrusive–it does not influence the interaction, and
- cost-effective–it reduces the human processing required to use the protocols.

Although automated trackers miss most of the extraneous non-verbal (and verbal) reactions of their users, they appear to meet these requirements reasonably. An automated tracker that meets these requirements was implemented as the primary means of collecting protocols of writers' software usage in this project.

2.3.2 Protocol Analysis

Collecting the protocols and preparing them in a consistent format is the prelude to the more extensive task of understanding the behavior of test subjects. For many years, efforts by psychologists to analyze protocols have focused on specific techniques for deriving condensed, but valid, data from verbal reports. Frequently the skill of the investigator determined the accuracy, consistency, and completeness of their personal interpretations of the verbal reports.

> "Reliability is a problem with any method of research, and particularly so with protocol analysis. People who look at protocols have the same problem as people looking at clouds: everyone might see something different." (Swarts, Flower, and Hayes, 1984)

Ericsson and Simon summarized the assumptions, techniques, and limitations of verbal reports as data in their 1984 book, *Protocol Analysis.* They noted that a few programs have appeared that systematize protocol analysis on a rudimentary basis. PAS-I and PAS-II (Waterman and Newell, 1971 and 1973) produce a problem behavior graph. SAPA (Bhaskar and Simon, 1977) predicts the information addressed and decisions reached in solving thermodynamics problems. The context is greatly constrained because the solutions are all built around the use of the equation for the conservation of energy.

A more specific problem is the analysis of protocols collected in the study of human-computer interactions. The following discussion of one analysis by Card, Moran, and Newell of a human-computer interaction serves to provide specific details about their method of protocol collection and analysis. Their analysis techniques provided the basis for the automated analysis techniques applied in this project. Other models for analysis follow this detailed discussion.

2.3.2.1 The ICARUS Study by Card, Moran, and Newell

In Chapter 10 of their 1983 book, Card, Moran, and Newell discussed their hand-analysis of the protocol of one subject doing a VLSI circuit layout. Most of their book focuses on carefully-controlled text-editing tasks, which Card, Moran, and Newell analyzed at the keystroke level. They presented this single chapter as an extension of their study to "a more creative task domain" where the subject was not given specific instructions to follow, but used the system to solve a problem.

The subject for their experiment was an experienced user. He talked aloud during the session; his hands and the screen were recorded with two videotape machines. His keystrokes was also captured in a file with timestamps. The session lasted about 40 minutes.

When beginning their protocol analysis, Card, Moran, and Newell noted difficulty in coordinating the keystroke recording and the two videotapes. The keystrokes were assembled into commands using hand editing and some heuristic programs. Because of difficulty in aligning the coordinate systems across the three recordings, they were unable to tell exactly what circuit element was being worked on.

Their protocol analysis noted that the user frequently paused during the experimental session to check the work he had done and to think about what to do next. A threshold of 5 seconds was chosen to identify the pauses from the inter-event intervals (which varied from 0.3 seconds to 80 seconds). This divided the session into about 100 episodes, each with an average time of 25 seconds. Each episode consisted of a pause of at least 5 seconds followed by a varying number of *ICARUS* commands. The episodes were manually identified by the researchers as one of four different types of tasks: Draw, Alter, Dimension, and Check.

They observed that the experimental session could be partitioned into phases lasting several minutes each. In each phase, episodes are organized around one of the major subproblems of the VLSI circuit-layout problem. They identified three phases, each of which occurred only once. The phases lasted 14 minutes, 7 minutes, and 15 minutes, respectively.

Their analysis of this session fits within a hierarchical model: sequences of individual commands form episodes, and groups of episodes partition a session into phases. Similar models have been used in other research efforts that examined human-computer interaction. These formal models were used to guide the protocol analysis efforts of these researchers.

2.3.2.2 Formal Models for Analysis

Three notable research efforts have employed hierarchical models in understanding the semantics of sessions between humans and computers.

Reisner (1981, 1982) used a formal grammar specification to describe the user commands of a graphics drawing system, as a means of comparing alternative interface designs. Her grammar describes tasks such as "how to draw a green line" in terms of its constituent user commands. The grammar is context-free (Type 2 power in the Chomsky hierarchy) and is described in Backus-Naur form (BNF).

Reisner's formal grammar gives a precise definition of the syntax of the action language she describes. She showed that using a BNF specification can highlight the irregularities in the syntax of the language; similar actions should have similar syntax. She used the grammar to describe the system after it was implemented. She did not discuss broadening the grammar to encompass higher-level strategies.

Card, Moran, and Newell (1983) studied performance of structured tasks at the keystroke level to infer time predictions for the subtasks. An example subtask would be to edit a word in a manuscript; its operations include finding the marked-up word on the page, scrolling through the manuscript to locate the word, moving the cursor to the word on the screen, and making the necessary changes. They introduce the GOMS method for formally representing their language; it consists of goals, operators, methods, and selection rules. GOMS expresses structured tasks as a hierarchy of goals. At the lowest level in the analysis (the leaf level), a user selects a method of fulfilling a goal by executing specific operations. The grammar represented by GOMS is context-free.

The GOMS model is well-suited to the study of time predictions and operator sequence predictions for small subtasks. However, its fine grain of analysis seems too low a starting point for examining overall strategies. Its syntax resembles that of a recursive programming language; in their examples, it frequently becomes verbose and awkward to read for any but the simplest decisions. In their own *ICARUS* study, Card, Moran, and Newell invented new terminology when working with the strategies of an entire session.

Kieras and Polson (1985) presented two formal descriptions: one for the user's knowledge of how to use a device, and a second for representing interactive devices. Their goal was to quantify the complexity of a computer's user interface. By representing the device behavior formally as well as the user's knowledge of it, they could simulate the user-device interaction to obtain rigorous measures of user complexity.

The focus of Kieras and Polson was the separation of device-independent knowledge from device-dependent knowledge. For their job-task representation (the device-independent knowledge), they used a production system, which they described as equivalent in power to GOMS. For their device representation (the device-dependent knowledge), they used a generalized form of augmented transition network (ATN), presented in a graphical form in their paper. An ATN generally has Type 0 power (Woods, 1970; Bates, 1978).

Some common threads recur in these research approaches. First, all three groups were looking at low-level interactions between the human and computer, essentially at the keystroke level. Because of the fine grain of analysis and the work involved in analyzing the data by hand, each of these efforts reported on the human-computer interaction of only one subject and one task.

Secondly, these researchers used grammars of Type 2 power to represent the interaction. Fountain and Norman (1985) observe the following in their review of Reisner's grammar and GOMS; it also applies to the job-task knowledge of Kieras and Polson.

> "All grammars above Type 0 impose restrictions upon the languages which are capable of being specified. This has important bearing upon the field of human-computer interaction, because the manner in which the language is specified is saying something about human cognition. Type 2 grammars allow only one non-terminal symbol to the left of a rule so that 1) we are implicitly assuming that the human cognitive process is strictly serial and monotonic: only one goal or task at a time may be attempted 2) the ability of a human to solve goals is context-free and history insensitive: the current task is not affected in any way by either previously completed tasks or future unstarted tasks. Only the current task environment is to have any bearing on the goal at hand. It is not clear that either of these conditions are necessarily true of the human cognitive process." (Fountain and Norman, 1985)

Recognizing these deficiencies, the Textlab group at UNC has taken a broader and more powerful approach to automating the analysis of their users' protocols. Smith, Rooks, and Ferguson (1989) describe a cognitive grammar for modeling users of their hypertext *Writing Environment (WE)* system. They describe their cognitive grammar as "a computer program that interprets the actions of a user working with an interactive application system in order to infer the cognitive activities taking place in the mind of that user." This grammar, implemented as an OPS-83 program, can be considered as five separate grammars that work in conjunction with one another. These five levels incorporate their theory of cognitive modes (Smith and Lansman, 1988); the principal nonterminal symbols represent the cognitive products developed by writers, the cognitive processes used in their creation or transformation, and the cognitive modes engaged by writers during a session.

I believe that these more powerful—and more complex—approaches are a promising direction for research into protocol analysis and the underlying cognitive processes. The approach taken in this project (described in Chapter 4) was built upon the same fundamentals as the grammar-based approach of the Textlab group.

2.4 Summary

Thus, three themes guide the research described in this paper. An understanding of human memory and how ideas are manipulated was motivated by the central tasks of reading and writing. Writing research, in particular, was reviewed in two areas: the structure and representation of ideas, and the processes and stages a writer goes through in order to produce prose. Writing systems for computers, specifically some hypertext systems, attempt to aid writers by making concrete the structure of the prose as it evolves. Finally, techniques for collecting and analyzing writers' sessions assist researchers trying to understand writers mental processes—in order to build better writing systems.

Chapter 3 describes the design and implementation of the specific writing system developed for this project. Its description is thorough, since it introduces the operation of each command available to a user. These commands form the terminals for the grammar-based parser that is described in Chapter 4 of this paper.

CHAPTER 3. SYSTEM DESIGN AND IMPLEMENTATION

In addition to reviewing existing structure editors and hypertext systems, the preceding chapter reviewed many cognitive factors to be considered in the design of a computer writing system. Members of the Textlab group have implemented several computer writing systems to explore how these factors might drive design decisions: *PROSE* (a UNC class project from 1984), *Storyspace* (Bolter and Joyce, 1987), *Writing Environment* (Smith, Weiss, and Ferguson, 1987), and *Prose II* (the testbed for this project). These systems share some common goals, addressing the three interacting human subsystems proposed by Card, Moran, and Newell:

- To lessen the load on a writer's cognitive system,
- To lessen unnecessary motor activity, and
- To effectively use of the human perceptual system and human spatial abilities.

These researchers applied a common set of techniques to the design of the four systems. An important technique was to assist writers in the external representation of their evolving cognitive structure. Single conceptual items are explicitly represented. These items can reference an unlimited amount of detail and associations. Items can be bound together to form new items, can be associated with one another, and can be hierarchically organized; the hierarchies can be moved as a single item.

Further, the power of spatial perception and imagery was utilized in these systems. Conceptual items can be graphically represented as visual entities. Links between items can be explicitly drawn. Hierarchies can be represented by trees and other two-dimensional visual forms. The conceptual nearness of items can be represented by arranging them so that there are short physical distances between them.

These systems include the ability to zoom in, allowing a user to focus on a small set of individual items—much like the human mind focuses attention on particular items by moving them from Long-Term into Working Memory. By zooming out, large chunks and overall relationships can be seen in the total spatial layout of the set of items being considered. The ability to visually roam within the layout allows yet a different view of active items under consideration.

Direct manipulation of the visual concepts (Schneiderman, 1983) further reduces the cognitive load of the writer. Using a mouse, a writer can grab the edge of a region to be zoomed and accurately predict the results of the action. Thus, visually-represented items can be directly acted upon.

Prose II, the testbed system for this project, was designed to make these techniques accessible on personal computers. Section 2.2.1.4 on page 25 gives a detailed list of the motivations for the creation of *Prose II*. A thorough discussion of its design and operation follow.

3.1 Prose II Design and Operation

Prose II was designed from the outset as a Microsoft *Windows* application. *Windows* is a graphics environment for IBM or compatible personal computers running the PC-DOS® or MS-DOS® operating system. It uses a 2- or 3-button mouse for input, in addition to a keyboard. The look and feel of the *Prose II* user interface is consistent with the graphics techniques described in the Microsoft *Windows* style guidelines (*Microsoft Windows User's Guide*, 1987).

Windows provides a system designer with independence from the hardware and software used in a personal computer. A variety of CPUs, screen sizes, printers, international character sets, and memory configurations are supported by *Windows*. This allowed me to distribute *Prose II* worldwide with few prerequisites.

Particular attention was paid to reducing the amount of cognitive overhead required to operate the system. A common characteristic of many text editing systems is their use of multiple editing modes,[3] where the operation of the system differs according to context. *Prose II* has just three editing modes: a normal editing mode, and two exceptional modes—Delete Mode and Tidy Mode—where the system operates differently. The active editing mode is identified in the action bar of the main window. In Delete Mode (see 3.1.6 on page 38), the cursor image also changes, and deleting nodes is the only valid operation. In Tidy Mode (see Section 3.1.8 on page 40), operations work as in normal mode, except that every tree operation causes all the trees in the workspace to be neatly re-drawn. I devised no reasonable design alternatives to having these two additional editing modes, so made their operation unmistakable. They must be explicitly activated and deactivated by a user with a mouse selection. They are clearly described with separate help panels.

This section has two goals: to describe how *Prose II* looks and operates, and to describe the format and operation of each of its 39 commands. A hypothesis of this project is that commands and sequences of commands build up into larger, hierarchically-organized pieces. This review of the commands and formats is thus detailed, because these commands and file formats are discussed frequently throughout the remainder of this paper. The commands form the terminals for the grammar used for the protocol analysis; this chapter is the place where their operation is described.

3.1.1 An Introduction to Prose II

With *Prose II*, a user can create and label nodes, where each node represents a single idea that will later be expanded into text—such as a paragraph in a technical paper. Nodes, and thus ideas, can be easily moved, copied, and rearranged, for example, to cluster related ideas. Nodes can be linked with one another in a hierarchical manner, and the resulting tree structures can, in turn, be moved and joined with other trees.

Nodes and the links among them were designed to visually approximate the concepts and interconnections in human Long-Term Memory. Trees are used to facilitate chunking; progressively lower levels of detail can be seen or hidden using the appropriate tree manipulation commands.

[3] The term editing mode used in this context differs from the term cognitive mode introduced by Smith, Weiss, and Ferguson (1987) and Smith and Lansman (1988). See Chapter 2 on page 13 for more details on cognitive modes. The names of editing modes will be capitalized in the remainder of this paper, *e.g.*, Delete Mode.

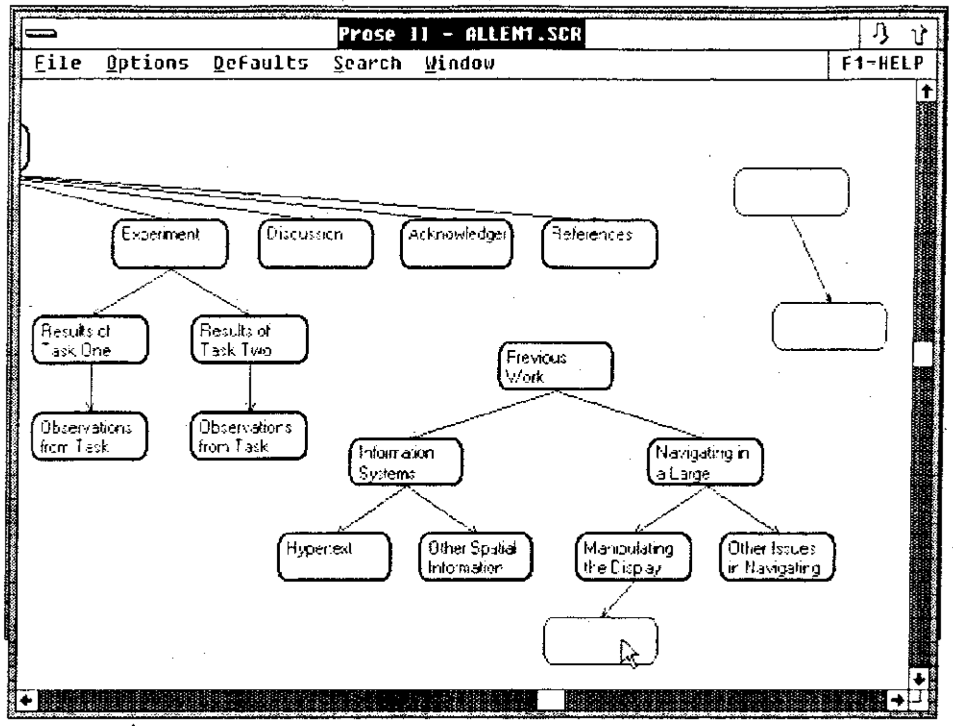


Figure 3. An example of portion of a workspace in *Prose II*

Figure 3 on page 34 shows an example *Prose II* screen, illustrating many of these features. Nodes are shown as rounded rectangles with identifying text labels. Some nodes are linked to other nodes with directed arrows. Parts of several trees are visible in this screen.

Additionally, a file can be associated with each node, and any program in the computer can be run against such a file. Usually a text file is associated with a node, and the program run against that file is a text editor—but users are not restricted to this combination.

When starting a new session, *Prose II* shows an empty workspace. The empty workspace represents a clean place to represent ideas, which are represented by labeled nodes.

3.1.2 Creating and Labeling a Node

Users can create a node in the main *Prose II* window by moving the mouse cursor to where they want the node located, and clicking the left mouse button. *Prose II* centers the newly-created node at the tip of the mouse cursor. The node can then be labeled.

A node label can consist of any text string; it is usually used to hold brief ideas (up to 250 characters) or to capture section headings in a document or article. Operationally, double-clicking inside a node with the left mouse button brings up a dialog box window for editing a label: the Edit dialog box. A user can then type the characters or words for the label. If a label already exists for a node, it may be modified, appended to, or deleted. Choosing the Save button ends the Edit dialog box and causes the label to be displayed inside the selected node. Alternatively, choosing the Cancel button ends the Edit dialog box without replacing the node's current label.

To reduce cognitive and motor effort, a new node may be created and labeled in a single step by moving the mouse cursor to the desired location and double-clicking the left mouse button.

3.1.3 Linking Nodes

Almost any two nodes in the main *Prose II* window may be linked.[4] All links in *Prose II* are hierarchical links. To link two nodes together in a parent-child relationship, a user moves the mouse cursor inside the node to be the "parent," holds down the left mouse button, and drags the mouse cursor to within the node to be the "child." The link follows the mouse cursor, and when the mouse button is released, a link is drawn between the pair of nodes.

Figure 4 on page 36 shows a link while it is being drawn. One end of the link is drawn in the parent node; the other end of the link follows the tip of the mouse cursor.

[4] No node may have more than one parent; nodes may not be linked in a circular relationship.

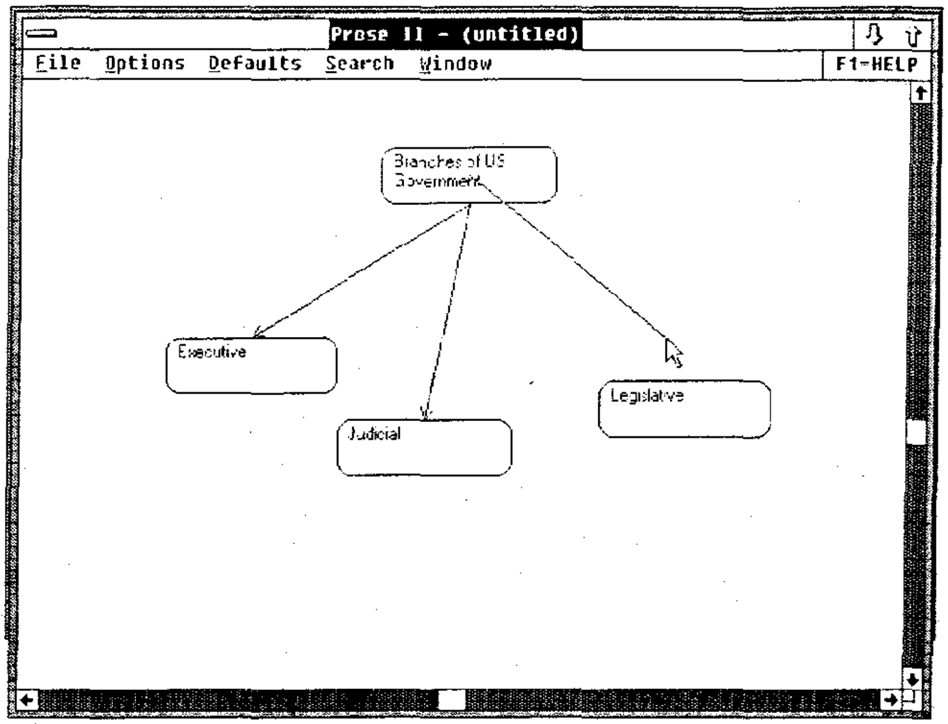

Prose II - (untitled)
File Options Defaults Search Window
F1=HELP
Branches of US Government
Executive
Judicial
Legislative

Figure 4. **An example of two nodes being linked.** Links already exist between the root of the tree, the node labeled "Branches of US Government," and two of its children: "Executive" and "Judicial."

A link between a pair of nodes can be broken by repeating the above procedure. Alternatively, giving a child node a new parent automatically breaks the link to its previous parent.

All the links in a workspace may be broken at once by choosing the "Break All Links" command. A user can also break all the links in a workspace and randomly reposition all the nodes by choosing the "Scramble" command. This affords an alternate method of brainstorming, by seeing nodes in different spatial relationships with one another.

3.1.4 Editing the File Associated with a Node

Aside from each node's label, a user can associate an arbitrary file with each node. Any program may be run against such a file. This gives the ability both to sketch out the structure of a paper (using nodes and labels), for example, and to write out detailed portions as they are thought of (by running a text editor against a node's file). *Prose II* thus supports the three types of concepts described by White (1985): propositions, algorithms, and images.

Prose II shows nodes that have files associated with them by drawing thicker frames than nodes without underlying files. A program can be run against the node's file any time the Edit dialog box is displayed. First, a user gets the Edit dialog box as before, by double-clicking the left mouse button in the node whose contents are to be edited. Next, choosing the Editor button invokes the program whose file extension is highlighted.[5]

Figure 5 on page 38 shows an example of the Edit dialog box, which is used to both write a label and to tie a node and a file together.

The Edit dialog box contains a list of file extensions. One of these file extensions is always highlighted in the list box–the last one used. Double-clicking with the left mouse button on a file extension in the list box saves the node's label and filename, then starts the corresponding program, using the current node's name as the first command-line parameter passed to the program.[6]

As a third alternative, pressing Enter when the Edit dialog box is showing either saves the label (if any is present) or starts the selected program (which is the first one in the list box, if one has not been selected yet). Users can thus operate with a single click if they are just entering labels or just editing files. When doing both, a double-click on a file extension or click on the Editor button is required.

The new program (such as a text editor) starts its own window or takes over the screen, depending upon its *Windows* setup. The mouse cursor is located in the new window so that *Prose II* no longer has control (this is known as the focus in *Windows* terminology).

3.1.5 Moving and Copying Nodes

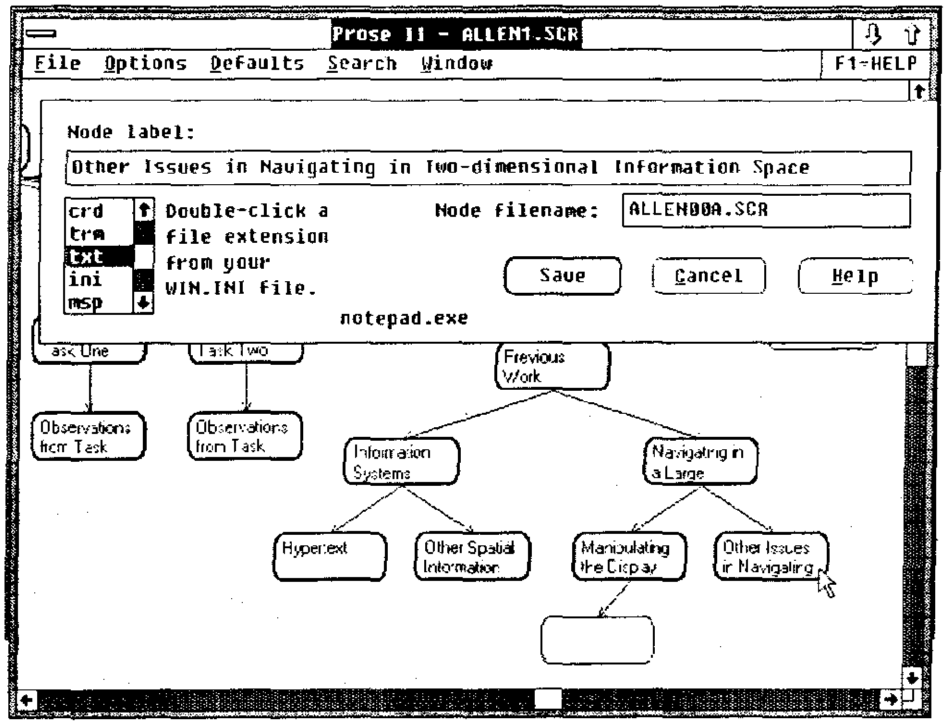
Moving and copying nodes requires the same type of mouse movements as linking nodes, but these three operations–linking, moving, and copying–look and feel different from one another. All three require starting with the cursor inside a node. Linking uses the left mouse button, moving: the right mouse button, and copying uses the Ctrl key plus the right mouse button.

Moving or copying nodes is done by positioning the mouse cursor anywhere inside the node that is to be moved, holding down the right button, moving the mouse, and releasing the button. While a node is being moved, links to the node stretch with a "rubber-band" effect; all its offspring nodes "snap" back into place when the move is complete. A node's position in a tree relative to its siblings is determined by its horizontal placement. Hence, moving a node to a different position among its siblings changes its place in the underlying tree structure. By additionally holding down the Ctrl key, these same steps are used to copy a node. There is no rubber-banding effect when copying.

Changeable defaults determine whether single nodes or whole trees are moved and copied. The initial default is to move and copy whole trees. Choosing the "Move node only" option causes subsequent moves to change only a single node (breaking any links it had to children nodes). Similarly, choosing "Copy node only" causes subsequent copy commands to copy only one node, rather than copying that node and all of its descendants. Additionally, the Defaults menu allows the choice for moving a node while maintaining its link with its parent, or not. The initial default is to maintain the link ("Keep Parent Link").

[5] In DOS, a file extension is a 1- to 3-character string that follows the filename and a period. When choosing to have a file associated with a node, *Prose II* forms a unique, default filename by appending a 3-digit node ID to the first five characters of the workspace filename. A user can choose to override this name and extension.

[6] In *Windows*, the association between a file extension and the related program to start with such a file is made by users in their internal run-time file, `WIN.INI`.


Prose II - ALLEN1.SCR

Node label: Other Issues in Navigating in Two-dimensional Information Space

Node filename: ALLEN00A.SCR

Double-click a file extension from your WIN.INI file.

notepad.exe

Figure 5. An example of the Edit Dialog Box in *Prose II*. The entire label is shown for the node at the tip of the mouse cursor.

3.1.6 Deleting Nodes

Three types of node deletion are available in *Prose II*: deleting a selected group of nodes, deleting the last node created, or deleting all the nodes in the workspace.

Deleting selected nodes is designed so that it is not done accidentally. To delete selected nodes, the Delete Mode must be explicitly activated. When activated, a special cursor is shown. The node is deleted by moving the mouse cursor inside a node that is to be deleted and clicking with the left mouse button. Delete Mode must then be turned off to return to normal operation.

Deleting the last node created is done by choosing the menu option "Delete the Last Node." As a shortcut, simultaneously pressing the Ctrl key and D deletes the last node created. This deletion shortcut is included because we observed that users occasionally created a node accidentally, then wanted to quickly delete it.

By choosing "New" from the File menu, a user can delete all the nodes in a workspace and reset the filename to (untitled) (as shown in the top of Figure 4 on page 36). Users can alternatively keep the current filename, but delete all the nodes, by choosing "Clear Drawing."

3.1.7 Moving through a Prose II Workspace

An important design tenet is to allow users a large workspace within which to lay out nodes, clusters of nodes, trees, and forest. As the workspace grows large, navigational techniques are needed to allow easy access to all regions. Three different navigational methods facilitated movement within the workspace: zooming in the current viewport, working with a map of the current workspace, and working with a text outline of the nodes' labels.

3.1.7.1 Zooming in the Main Window

Users can zoom in the main *Prose II* window to enlarge or deform anything in the viewport. Clicking the right mouse button in a blank region causes a dotted rectangle to be drawn, indicating the zoomed region. This rectangle follows the mouse cursor. When the button is released, the window is redrawn, with items in the rectangle now filling the main window. This option can also be used to change the shape and size of the nodes. For example, long labels can sometimes be viewed in their entirety by making nodes appear to be long and flat.

3.1.7.2 Working with the Map Window

A map of the entire workspace can be brought to the forefront when desired. This map shows all the nodes and links in the workspace, as well as their spatial relationship. It allows a user to move to any other area of the workspace and also to zoom in or out. Commands executed in the main *Prose II* window also cause the Map Window to be updated. Showing and hiding the Map Window is done by selecting a menu item. An example of the Map Window is shown in Figure 6 on page 40.

A wire-frame rectangle in the Map Window indicates what portion of the current workspace is presently visible in the main *Prose II* window. The shape of this wire-frame rectangle may not be proportional to the main window; by changing the shape of this rectangle, the nodes in the main *Prose II* window change their proportion accordingly. Their original shape and position can be reset by choosing the "Reset-Drawing" menu item (or by pressing the Home key).

Two different actions can be used to update the main *Prose II* window. To pan, a user makes a single click of the left mouse button anywhere inside the wire-frame rectangle and drags the wire-frame rectangle. To zoom, a user makes a single click of the right mouse button anywhere inside the Map Window and drags the new wire-frame rectangle. A blank screen beyond the extent of the furthest node can always be reached in any of the four directions.

Finally, the size, shape, and position of the Map Window itself can be changed using standard *Windows* techniques.

3.1.7.3 Working with the Outline Window

An Outline Window shows an outline view of the node hierarchy. Each level of tree depth is shown by indenting the labels three spaces. Unlabeled nodes are shown as `--unlabeled--`. Individual trees are distinguished by a line of dashes. As with the Map Window, commands in the main *Prose II* window cause the Outline Window to be updated. Also, showing and hiding the Outline Window is done by selecting a menu item, or by using the middle mouse button. An example of the Outline Window is shown in Figure 7 on page 41.

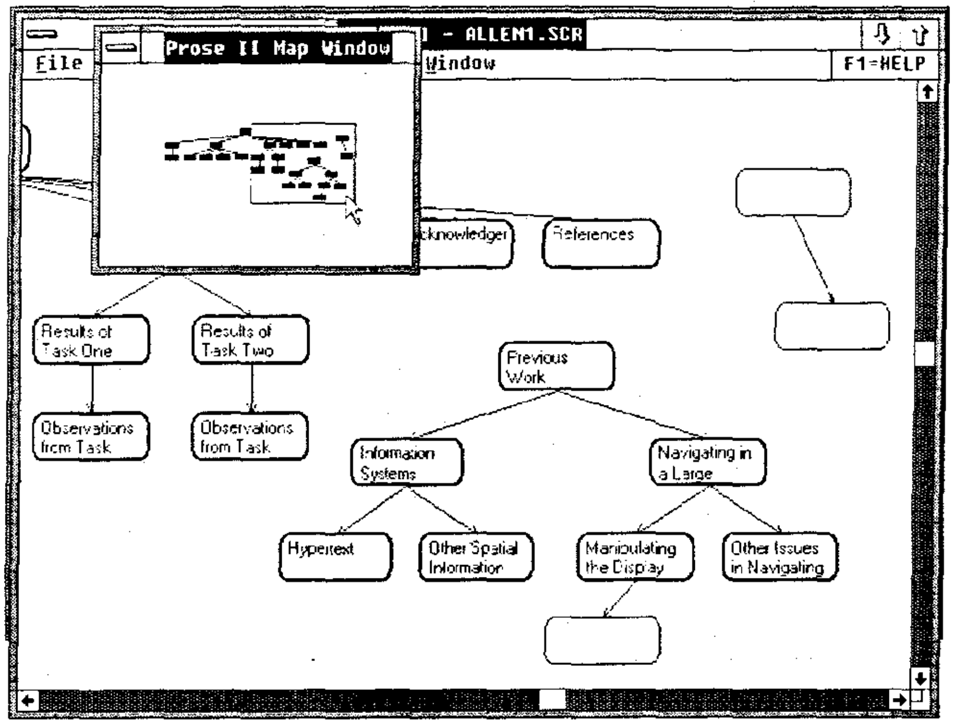


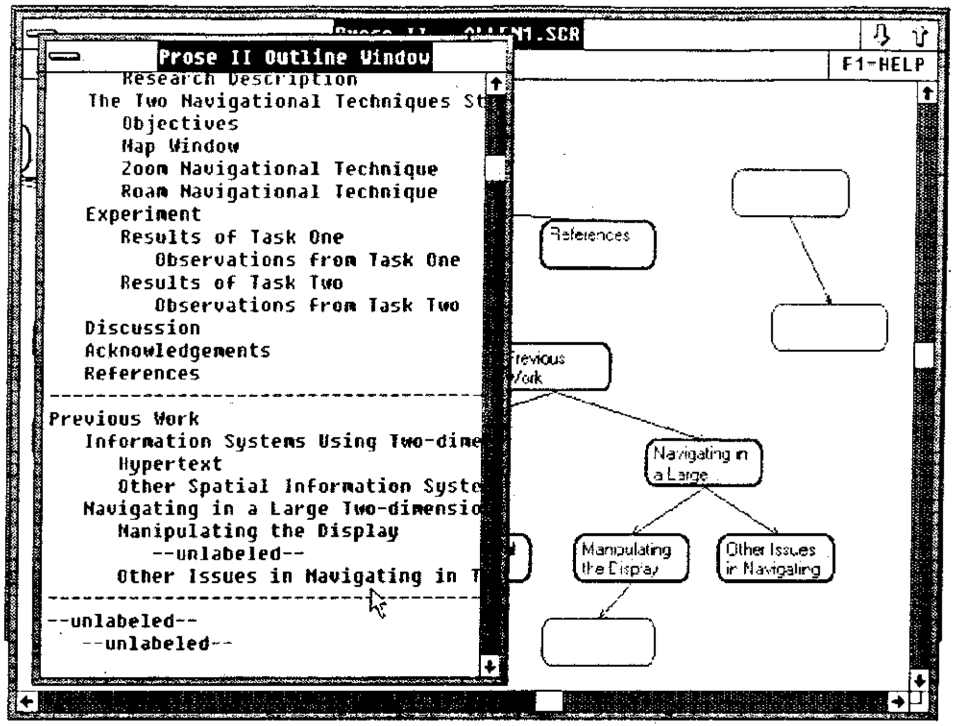
Figure 6. An example of the Map Window in *Prose II*

3.1.8 Tidying Trees

By default, *Prose II* shows a node in the position where it was originally placed. A tidy-tree operation allows a user to create nodes anywhere and later clean up the node arrangement. *Prose II* employs a new tree-tidying algorithm to reposition the nodes of a tree (Walker, 1990, 1991).

Tidy Mode must be explicitly activated as a separate editing state, which causes *Prose II* to tidy the entire workspace. In Tidy Mode, the workspace is then re-tidied after each command that affects the tree structure. Choosing "Turn Tidy Trees Off" from the Options menu returns a user to normal operation.

While in Tidy Mode, the gray minus ("−") and plus ("+") keys can be used to shrink and grow the tree drawings. Pressing "−" hides the lowest level of the trees; repeatedly pressing this key can hide all levels up to the root of the trees. Pressing "+" shows the last level that was hidden. Alternatively, by turning Tidy Mode off again, the full trees are restored. When doing a Save of the workspace file while the tree is shrunk, the nodes in the lower levels of the tree that have been hidden by the shrinking process are not saved. This allows displaying cross-sections of a document—just looking at everything in the document down to the second level, for example, without the detail.



```
Research Description
  The Two Navigational Techniques St
     Objectives
     Map Window
     Zoom Navigational Technique
     Roam Navigational Technique
  Experiment
     Results of Task One
        Observations from Task One
     Results of Task Two
        Observations from Task Two
  Discussion
  Acknowledgements
  References
-----------------------------------------
Previous Work
  Information Systems Using Two-dime
     Hypertext
     Other Spatial Information Syste
  Navigating in a Large Two-dimensio
     Manipulating the Display
        --unlabeled--
     Other Issues in Navigating in T
-----------------------------------------
--unlabeled--
  --unlabeled--
```

Figure 7. An example of the Outline Window in *Prose II*

A user can tidy up the workspace with one quick command by choosing the "Tidy the Drawing" command from the Options menu; this does not turn on Tidy Mode. As a shortcut, simultaneously pressing the Ctrl key and T tidies the drawing.

As an arbitrary design restriction, *Prose II* does not handle trees with more than 99 levels. It handles nodes until it runs out of memory (within the DOS 640 Kbyte memory boundary). I have had more than 1000 nodes concurrently.

3.1.9 Changing the Root Orientation

The default positioning for trees in *Prose II* is to draw the root of each tree at the top of the drawing, with its offspring below it. Dialog box options allow other orientations, such as placing the root on the left and the siblings to its right. Four such orientations of the root are available:

North	Root is at the top, its siblings are shown below it.
South	Root is at the bottom, its siblings are above it.
East	Root is at the right, its siblings are to its left.
West	Root is at the left, its siblings are to its right. An example of a tree with the West orientation is shown in Figure 8 on page 42.

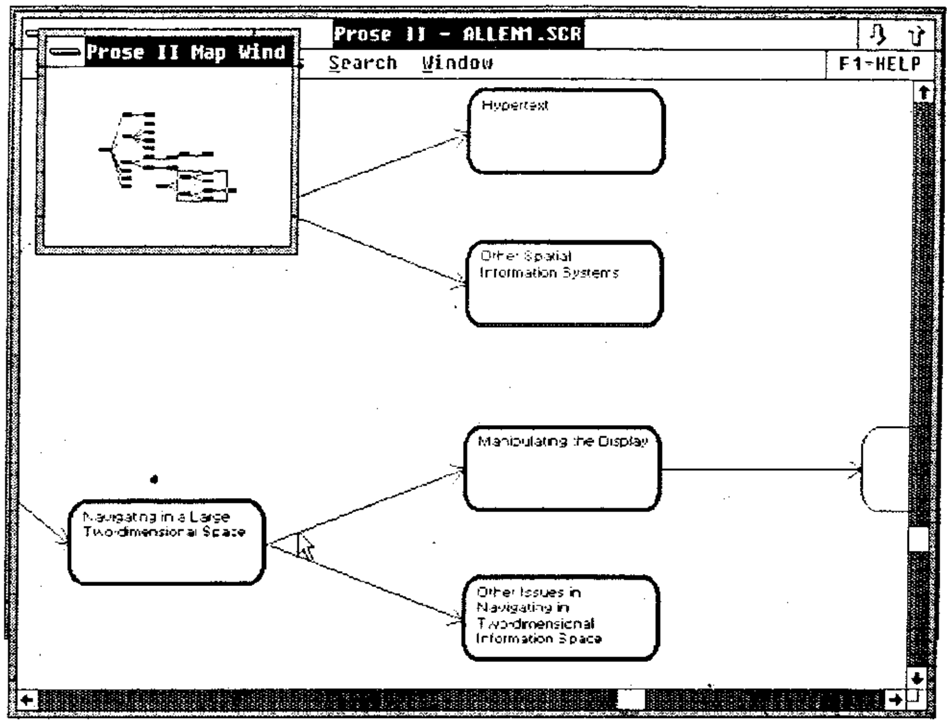

Prose II Map Wind
Prose II - ALLEN1.SCR
Search
Window
F1=HELP
Hypertext
Other Spatial Information Systems
Manipulating the Display
Navigating in a Large Two-dimensional Space
Other Issues in Navigating in Two-dimensional Information Space

Figure 8. An example of a tree with a root orientation to the west

3.1.10 Working with Workspace Files

Prose II workspace files can be created, saved, cleared, and listed using commands in the File Menu of the *Prose II* window. Commands from the File Menu in Window's Executive window are used to delete *Prose II* files.

3.1.10.1 Creating a New Workspace File

To create a new, untitled workspace file from the *Prose II* window, users select the File menu and choose the "New" command. To create a new, named file, they use the Open dialog box by selecting the File menu and choosing the "Open" command. The name of a new file is typed into the text box at the top of the Open dialog box. If the workspace file is not found, *Windows* displays a message inquiring whether a new file should be created or whether the file can be found on a diskette.

3.1.10.2 Opening an Existing Workspace File

To open an existing workspace file from the *Prose II* window, a user selects the Open dialog box. In the displayed list box, they can select the name of the file to open (by double-clicking on it with the left mouse button), or typing a pathname and filename in the text box at the top of the dialog box.

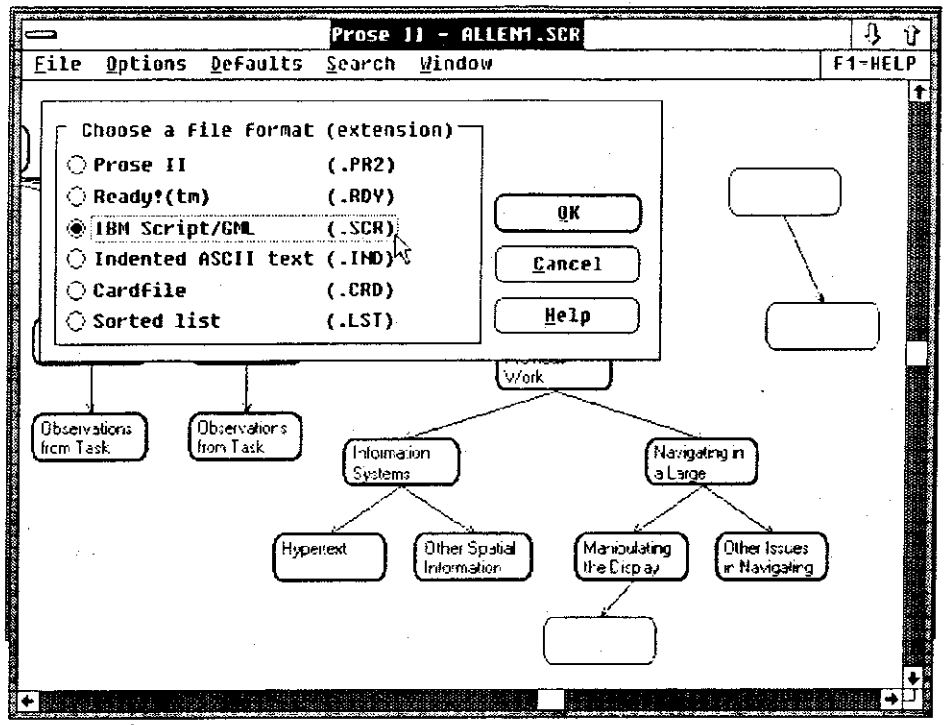
3.1.10.3 Workspace File Formats

The only workspace files *Prose II* can open are those with file formats it understands. Opening or saving a workspace in *Prose II* commits it to a file format. The six supported file formats are indicated by their DOS file extensions. The dialog box used in *Prose II* to select file formats is shown in Figure 9 on page 44.

Extension	**Description of this file format**
`.PR2`	This is the default file format for *Prose II*. It allows saving and opening files containing any mix of isolated nodes and multiple trees. This format also saves the x and y coordinates of each node and the date and time of their creation and last modification. This format also allows text, graphics, or any file to be associated with each node.
`.RDY`	This file format is compatible with the *Ready!*™ outline processor. Any file created by *Ready!* can be read by *Prose II*. There can be only one tree and no isolated nodes.
`.SCR`	This file format is compatible with the IBM Script, GML, and BookMaster document processors. Reading a `.SCR` file as input causes each appropriately-placed heading tag (such as, ":h2.") to become a node label in a tree. Files written by *Prose II* in this format contain a heading tag of the correct level of each node in the tree(s).
`.IND`	This file format is for reading and writing indented ASCII text. On input, each line becomes a node label (up to 250 bytes). *Prose II* correctly handles the tree construction for any consistent indenting method. On output, the labels are indented 3 blanks for each tree level.
`.CRD`	This is compatible with the Microsoft *Windows* Cardfile format. `CARDFILE.EXE` is one of the set of desktop applications that is shipped with Microsoft *Windows*. On output, the maximum Cardfile index line is 40 characters, so labels can get truncated. The maximum card text is 450 characters.
`.LST`	This file format is for input of a sorted list of words or phrases. The structure is automatically formatted into a single, left-to-right balanced, tidy tree by *Prose II*. This format served as my testbed for a new fanout algorithm to position the nodes in a left-to-right balanced tree.

A user can change the workspace file format at any time, using a dialog box. *Prose II* internally saves the current file format each time a workspace file is saved. This format is used as the default file format the next time a workspace file is opened or saved.[7]

[7] Preview of results: among the 112 sessions studied here, none of the subjects opened or saved saved any workspace using the .RDY or .LST formats.



Choose a file format (extension)

○ Prose II (.PR2)
○ Ready!(tm) (.RDY)
[✓ ● IBM Script/GML (.SCR)]
○ Indented ASCII text (.IND)
○ Cardfile (.CRD)
○ Sorted list (.LST)

Figure 9. An example of file format selection box in ***Prose II.*** The .SCR file format is being selected at the mouse cursor.

3.1.10.4 Saving and Deleting Files

The "Save As" command names and saves a new file, or saves the current file under a new filename or file format. Selecting the File menu and choosing the "Save" command causes *Prose II* to replace the file on disk with the current workspace file.

3.1.11 Copying to the Clipboard

The *Windows* Clipboard application holds information cut or copied from other *Windows* applications. After sending an image from *Prose II* to the Clipboard, a user can paste the contents of the Clipboard onto another *Windows* application. Images copied from *Prose II* to the Clipboard are sent as a bitmap. *Prose II* does not have a facility for pasting images from the Clipboard.[8]

[8] Preview of results: among the 112 sessions studied here, information was copied to the clipboard only once.

3.1.12 Searching for a Node

Users can search through the labels of all their nodes for a given search string. When a match is found, that node is centered in the main window. A user enters the search string in a popup dialog box. By repeating the search, other nodes with the same search string in their labels can be found.

3.1.13 Requesting Help

Prose II only operates as a *Windows* application; that is, it performs its graphics functions when started from within Microsoft *Windows*. However, if a user starts it at a DOS command prompt, *Prose II* displays 125 lines of information describing its purpose and usage.

Ten different help panels are available within *Prose II*. The menu of help panels can be raised by pressing the "F1" key at any time, as well as by selecting the proper help button with a mouse. Many of the helps are also available in appropriate context-sensitive situations. For example, the dialog box for editing a node's label contains a help button that immediately displays the help for editing nodes.

As a way to summarize the operation of *Prose II*, the text for all of the help panels is shown in Table 2 on page 45.

Table 2 (Page 1 of 3). *Prose II* Help Panels

Introduction	`Prose II Structure Editor`
	`Version 2.09`
	
	`Copyright 1986, 1989 John Q. Walker II`
	`All rights reserved.`
	
	`IBM VNET: JOHNQ at RALVM6`
	`IBM Internal Use Only`

The First Time?	`Prose II is a structure editor you use to explore and organize ideas. You can use Prose II to create and label nodes—each node representing a single idea to be included in a larger structure, such as a paragraph in a technical report. Nodes can be visually linked into trees, and the trees can be linked to other trees or simply kept as a forest or as clusters of nodes.`
	
	`Prose II gives you the functions of an outline processor in a large, 2-dimensional windowed workspace. With an outline processor, you are forced to commit your ideas to a hierarchical position. Now you can position or cluster them anywhere and organize whenever you wish.`

Table 2 (Page 1 of 3). *Prose II* Help Panels

The Mouse Buttons	Use the LEFT mouse button to Create, Edit, or Link nodes CREATE: single click outside of any node. EDIT Label and Contents: double click inside a node. LINK: single click and drag starting inside one node and releasing the button in the target node. BREAK LINK: redraw an existing link between nodes. Use the RIGHT mouse button to Move and Copy nodes and to Zoom In for a closer view: MOVE: single click and drag inside a node. COPY: same as MOVE, except also hold down Ctrl key ZOOM IN: single click outside a node and drag. Items in the rectangle will fill the window.
The Map Window	Show the Map Window to easily Move to other areas of the workspace and to Zoom In and Zoom Out. - The rectangle shown in the Map Window indicates what is presently visible in the Main Window. A single click and drag of the LEFT mouse button will move this rectangle, updating both the Main Window and the scroll bars. - A single click and drag of the RIGHT mouse button draws a new rectangle indicating what will show in the Main Window. This allows easy zooming. - The Map Window can be moved, sized, and hidden using the standard Windows techniques.
Delete and Tidy Modes	Select "Turn Delete On" from the Options menu to delete undesired nodes. - A special cursor will be shown. With DELETE ON, use a single click with the left mouse button to delete the selected node. - Shortcut: Ctrl+D deletes the last node you created. Select "Turn Tidy Trees On" to tidy up the entire workspace. If you leave TIDY ON, the workspace will be re-tidied after each tree operation. Use '-' and '+' to shrink and grow the tree drawings. - Shortcut: Ctrl+T tidies the drawing quickly.
File Formats	Choose a format for storing the file for future use. .PR2 the default format. Allows any mix of isolated nodes and multiple trees. Saves x,y coordinates and file references. The best for general use. .RDY compatible with the Ready!(tm) outline processor. Ready! allows only one tree and no isolated nodes. No coordinates or file references are saved. .SCR compatible with IBM Script/GML, except for titles. Reads and writes ':hx.' tags for each node label. .IND indented ASCII text. No coordinates are saved. .CRD Windows Cardfile, but reads the index line only. On output, labels are truncated to 40 characters. .LST input only; reads a sorted ASCII list and creates a complete left-to-right filled tree.

Table 2 (Page 2 of 3). *Prose II* Help Panels

Editing Nodes	Double-click in a node to enter a label and double-click on a file extension to edit a file tied to this node. - Use the label for brief ideas (up to 250 characters) or section headings in a document or article. - Type as normal the first time you enter a label. After the first time, the label will be in reverse-image. Just type to replace the old label. Cursor keys append to or position within it; End moves to the end; Backspace erases single characters. - Then: Save the label, click an extension, or Cancel. - Double-clicking an extension saves the node's label and filename, then starts the program named in the [extensions] part of your WIN.INI file.
Changing The Trees	By choosing the Change Nodes or Change Links items from the Options menu, you can change the way the tree looks. Among the interesting choices in the Change Links box is the ability to change the way the root of the trees are pointed. - The default is North, where the root of each tree is at the top of the tree. - Another common choice is West, where the root of each tree is at the left side of the tree. You may need to Tidy the Drawing after changing one of these options, depending on your file format.

Your WIN.INI File	When tying a file to a node, Prose II forms its filename by appending a 3-digit ID to the first 5 characters of your workspace filename. You choose from the list for a file extension, based on your WIN.INI file.
	
	When Prose II automatically starts this program, it must tell Windows if this is a native Windows application or not. In the [extensions] part of WIN.INI, make the program's file extension lowercase for native Windows applications and uppercase for standard applications.
	
	Example entries, under [extensions]:
	wri=write.exe ^.wri a native Windows application
	pr2=prose2.exe ^.pr2 a native Windows application
	scr=pe2.PIF ^.scr a standard application

Help Me	Please help me improve PROSE II. - Bugs: Let me know what bugs you stumble across. - Improvements: What features would you like me to add? Since I don't like manuals, what else should I add to the Helps? Where should I work on performance? What additional I/O formats should I support? - Recordings: Please e-mail me the automatic session recordings, which I will analyze and parse as part of my study of user interfaces. Press F2 to leave your comments; press F3 when you take a lengthy break.

3.2 Prose II Implementation

Before beginning the design of *Prose II*, I began by understanding the design of an existing structure editor. *PROSE*, A Tree-Structured Writing Support System, was developed as a class project for the Software Engineering course at UNC-Chapel Hill in Spring 1984. It was implemented in the *C* programming language and ran on the UNIX operating system for the departmental VAX/11-780. *PROSE* was character-based; that is, it used characters on the terminal for drawing lines. The displayed information was limited to 80x24 characters.

In 1985, I ported *PROSE* to the IBM PC, including writing my own Curses package for the PC to simplify the port. Given this background understanding, I built a new structure editor, *Prose II*, in 1986 and 1987. It was designed and implemented from scratch to run in the Microsoft® *Windows* graphical environment. *Prose II* is based on many of the ideas embodied in *PROSE*, although it shares no source code with it. I felt the bond was strong enough that it shares its name with its progenitor.

Another ancestor of *Prose II* is *tde*, a transition diagram editor available on the UNIX operating system running on the Sun Workstation™ (Mills, 1984). While primarily designed to illustrate finite state machines and state diagrams, *tde* can also be used as a general-purpose graph-drawing editor. The primary objects it manipulates are nodes and the arcs among them, which are also the principal components manipulated by *Prose II*. I spent the time to understand the visual operation of *tde* before I began implementing *Prose II*.

The version of *Prose II* used in this project consists of a single executable file, PROSE2.EXE. This file's size is 120,448 bytes when the tracker was compiled in, and 106,192 bytes without the tracker. Thus, the tracker increases the total size by about 13%.

The implementation of *Prose II* with the tracker is broken into 41 files of source code written in the *C* language and 2 files written in assembler. In these files are a total of 17,825 lines, which includes comments and blank lines.

***Prose II* source code file sizes, 43 files**

```
  range:  39 lines to 910 lines
 median: 414 lines
   mean: 415 lines
std dev: 231
```

CHAPTER 4. PROTOCOL COLLECTION AND ANALYSIS

The *Prose II* structure editor introduced in the previous chapter contains an additional capability not highlighted there: it can record the commands executed by its users. This recorder thus generates one complete protocol record for each session. For this study, I built a set of software tools to analyze these protocol records. This chapter, describing these tools, consists of two large sections: the first section discusses the protocol record files and how they were collected; the second section discusses the grammar and parser used to analyze the protocol records.

4.1 Automating Protocol Collection with a Tracker

As described in the first chapter, many of the methods and tools for collecting data about software usage have been expensive to use and obtrusive to the humans being observed. An automatic tracker was incorporated into the testbed software system to alleviate these problems. The design of this tracker is derived from the research of Card, Moran, and Newell (1983). During a session, all commands performed by a user of *Prose II* are automatically written to a new file in the user's file system. This section describes these data, along with the design and operation of the *Prose II* tracker. Because of the exploratory nature of this study, the rationale for the design decisions is discussed, as well as alternative approaches.

4.1.1 Content and Format of the Protocol Record Files

When designing this tracker, I attempted to strike a balance between too fine a granularity (and, thus, excessive data in each protocol record file) and too general a record. As the tracker captures each *Prose II* command (*e.g.*, "Create a node"), it associates it with the corresponding parameters and a timestamp. The tracker filters out a moderate amount of low-level information; for example, if a user doodles with the mouse (*i.e.*, moves the cursor around the window without making any explicit selection), the tracker records this simply as a pause.

In contrast, the tracker for the *WE* system (Smith, Rooks, and Ferguson, 1989) records information at a finer level than the *Prose II* tracker. The *WE* tracker records each separate user action; for example, the "create node" command entails the separate actions of opening a menu, highlighting the appropriate menu item, and giving the node a new name. The *WE* analysis tools then combine these actions in the first level of protocol analysis; thus, their "Operation Level" corresponds to the commands recorded directly by the tracker in *Prose II*.

Prose II is designed for worldwide use, on personal computers running Microsoft *Windows*. Other details of the actual hardware and software being used are unknown. As a result, the protocol record files do not contain sufficient detail for a high-fidelity playback of the session on an arbitrary computer. The reason for insufficient detail is that among sessions and subjects, any of the following system variables can differ:

hardware environment

Examples are the CPU speed, the screen size and density, the amount of memory, and the type of disk access (RAM disk, hard disk, or floppy disk).

operating system characteristics

Examples are the number of buffers and file handles set aside, and other programs that may be active.

Windows **parameters**

Examples are the size and placement of the windows, the international character set being used, and the frequency of internal messages.

Prose II **defaults**

Examples are the current file format being used, whether a node's children are deleted if it is deleted, and the size and placement of the Map and Outline windows.

In contrast, high-fidelity playback is provided in the *WE* system, since the range of computers it runs on is small. Researchers analyzing *WE* recordings have built playback software that reproduces the details of a user's interaction directly on the display screen. A researcher can step through a session recording one action at a time, or can play back a session at a variety of speeds without intervention.

Figure 10 on page 50 shows an example of an entire protocol record file generated by the *Prose II* tracker during a session. The session was short (less than a minute), and the user's actions can be considered trivial, since no text was written in the nodes and the nodes were not saved.

```
+-----------------------------------------------------------------------+
| Prose II Session Recording (v2.09)          Wed Feb 08 20:31:23 1989 |
| File B:~RCD1300.TMP                                                   |
+-----------------------------------------------------------------------+
| Start     Stop      Time  Operator              Parameters            |
|min:sec   min:sec     sec                                              |
+-----------------------------------------------------------------------+
31:23.52  31:29.61    6.09  PAUSE
31:29.61  31:29.78    0.17  CreateNode            ID(1) StartPt(14, -18)
31:29.78  31:34.23    4.45  PAUSE
31:34.23  31:34.45    0.22  CreateNode            ID(2) StartPt(115, -41)
31:34.45  31:35.44    0.99  PAUSE
31:35.44  31:35.66    0.22  CreateNode            ID(3) StartPt(273, -109)
31:35.66  31:44.66    9.00  PAUSE
31:44.66  31:47.96    3.30  NewWorkspace
                            + DeleteNode          ID(1)
                            + DeleteNode          ID(2)
                            + DeleteNode          ID(3)
31:47.96  31:59.49   11.53  PAUSE
```

Figure 10. The first, complete protocol record file returned by subject number 28. In this trivial session, the subject created three nodes, then deleted them by starting a new *Prose II* workspace.

The format of a protocol record file, such as this one, closely follows the format designed by Card, Moran, and Newell (1983). This format includes the start time, the stop time, and the elapsed time for each command, followed by the command name and its parameters. The timestamp is accurate to within one one-hundredth of second. For brevity, the timestamps contain only minutes, seconds, and hundredths of seconds. In retrospect, a minor weakness of using Card, Moran, and Newell's exact format is that it does not handle indefinitely long periods of time well. Because there is nothing longer than minutes recorded (that is, hours, days, or months are not recorded), the tracker (and, therefore, the parser) assumed that no more than one hour would elapse between consecutive entries. This could be a limitation in systems intended for actual-use studies; work was often continued across multiple sessions and *Prose II* windows could have easily been left running overnight.[9]

When a *Prose II* session is started, the tracker devises a unique file name for the protocol record, opens that new file, and writes a header in the file describing its contents. This seven-line header contains a date and timestamp for the beginning of the session. In retrospect, some of the environment and state variables that may have permitted replay might have been captured in the header of each protocol record file. For example, by using internal *Windows'* calls, the tracker could have recorded the screen size in pixels, and written that information in the header.

Many of the individual commands have side-effects; for example, opening a new workspace file will cause any existing nodes to be deleted, as shown in Figure 10. Similarly, opening an existing workspace file can cause many nodes to be created and linked together. The tracker records these side-effects, called secondary commands in this study—in contrast to the primary commands executed by the user. Most commands can be either primary or secondary, depending on the context in which they are used. The tracker places secondary commands in a protocol record file below their primary counterpart, indented with a plus sign but with no timestamp.

Another example of a protocol record file, with primary and secondary commands, is shown in Figure 11 on page 51.

```
+-----------------------------------------------------------------------+
| Start     Stop      Time  Operator            Parameters              |
|min:sec   min:sec     sec                                              |
+-----------------------------------------------------------------------+
23:45.29  23:48.75    3.46  LinkNodes           ParentID(7) ChildID(19)
23:48.75  23:50.18    1.43  PAUSE
23:50.18  23:50.62    0.44  LinkNodes           ParentID(19) ChildID(13)
                            + BreakLink         ParentID(7) ChildID(13)
```

Figure 11. A portion of a protocol record file showing primary and secondary commands. This example shows two LinkNodes commands. Linking node 7 to node 19, and then linking node 19 to node 13 caused *Prose II* to break the existing link between nodes 7 and 13. This secondary command, BreakLink, is shown beneath its primary LinkNodes command.

One result of tracking secondary commands is that the overall size of a protocol record may be disproportionate to the number of primary commands executed by a subject. A simple operation, such as moving the root of a tree, may cause many secondary commands to be tracked. In this example, when the root of a tree is moved, there is one secondary command recorded for each of its descendants, tracking the change in position of each descendant.

The tracker can also show that a command is invalid or canceled. For example, *Prose II* does not allow a linked cycle of nodes to be drawn; when attempting to link the last nodes, the LinkNodes command fails since it is invalid with respect to the *Prose II* data model of a forest of

[9] The collecting of elapsed hours would be a simple modification. Without changing Card, Moran, and Newell's compact format, a tracker alarm could cause a line to be written to mark the passing of each elapsed hour.

trees. This would be shown in the protocol record as the parameter "`-- invalid --`." Similarly, commands corresponding to dialog boxes can be canceled.

An example of a protocol record containing a canceled command is shown in Figure 12 on page 52.

```
+------------------------------------------------------------------------+
| Start      Stop       Time  Operator             Parameters            |
|min:sec   min:sec      sec                                              |
+------------------------------------------------------------------------+
40:01.67  40:10.84    9.17  EditLabel            ID(3) -- canceled --
```

Figure 12. A portion of a protocol record showing a canceled command. The subject brought up the Edit dialog box for editing node 3's label, but canceled the command after 9.17 seconds.

To increase runtime speed and decrease storage space, templates for all the command and parameter title strings written by the *Prose II* tracker were compiled internally, as opposed to being dynamically allocated. Care was taken to reduce their overhead in the *Prose II* executable module and in each of the lines written to a protocol record. For example, whenever a string of eight blanks is to be written, a one-byte tab character is written instead. Careful attention to these details kept the protocol records small, which improved performance for the user.

I designed the tracker to produce protocol records that are human-readable—in plain English. The tracker could have produced its output in a more compact, computer-oriented format—possibly in a straight binary format—but the plain English form accomplishes three things. First, the protocol records are portable; they can be printed or hand-analyzed without additional processing. For example, this let me analyze printed protocol records by hand until I knew enough about users' behavior to construct a parser to produce similar results automatically.

Second, the human-readable format gave subjects the assurance that nothing "underhanded" was going on in the protocol records. Everything in a protocol record is immediately readable and obvious to any subject; they could see that the tracker was not capturing secret data. Secret data might conceivably consist of any information a user considered private to their own computer, such as the names and contents of other files on their computer. Further, the tracker x'd out all the letters in the labels and search strings in the protocol records. This preserved the anonymity of the subjects and their topics, and allowed them to use the system for business purposes without compromising IBM confidentiality. Since *Prose II* and its protocol records were distributed via a global computer network in a time of heightened concern for computer viruses. For this type of study to be successful, researchers and subjects must be assured that free software and free recordings were adequately controlled against security problems.

Finally, many commercial products are capable of writing trace records or recording sequences of user commands. The protocol format used in this project is "application-aware," but is straightforward with little additional context information. I designed the techniques described here so that they could be adopted by future researchers to construct parsers for commercial products that produce traces or macros.

As seen in the examples, the tracker generates simple characters in the header (*e.g.*, plus signs, dashes, and bars), rather than "prettier" graphics symbols. I had four reasons for using simple characters:

- to assure that a user could view the file with even the simplest of text editors,
- to assure that it could be uploaded to a host system and sent across a network without character translation anomalies,

- to reduce the overall size of the protocol record files. Subjects were more likely to upload and return their protocol recordings if their size was relatively small. For example, tab characters were used wherever possible to reduce 8 blanks to a single character.
- to simplify its later handling as input to the parser. For example, the parsing software generated by the UNIX tools LEX and YACC recognize only 7-bit ASCII characters.

Table 3 on page 53 is an alphabetical list of the 39 *Prose II* commands and their possible parameters. It also shows precisely when the time value associated with each action is recorded. These commands are the terminal symbols for the grammar used to analyze *Prose II* protocol records.

Table 3 (Page 1 of 2). Commands recorded by the tracker in *Prose II*, listed alphabetically

Command Name	*When was the start time recorded for this command?*	*When was the stop time recorded for this command?*	*Parameters: "–" indicates no parameters*
BreakLink	mouse down in a linked node	mouse up in a linked node	ParentID(), ChildID() CANCELED
BreakAllLinks	menu selection	instantaneous	– CANCELED
ChangeDefault	menu selection	instantaneous	ID()
ClearDrawing	menu selection	instantaneous	– CANCELED
ClipboardCopy	menu selection	instantaneous	–
Comment	press F2 function key	select save or cancel	Text() CANCELED
CopyNode	mouse down	mouse up	StartPt(,) INVALID
CreateNode	mouse up	instantaneous	NodeID() StartPt(,) CANCELED INVALID
DeleteNode	mouse up in node to be deleted	instantaneous	ID()
EditLabel	mouse up	select any dialog button	ID() NewText() ID() CANCELED
EditNode	menu selection	return from file	ID() Editor() File() CANCELED
GoTo	menu selection	select OK or cancel	Text() CANCELED
HelpRequest	help requested	exit	ID() CANCELED
LeaveProseII	lose focus from a *Prose II* window	regain focus	ID(0|1) 0: went to a different window 1: pressed the F2 key
LinkNodes	mouse down in a node	mouse up	ParentID() ChildID() CANCELED
MainWindowReset	menu selection	instantaneous	–

MainWindowZoom	mouse down	mouse up	StartRect(,,,)
			EndRect(,,,)
			INVALID

MapMove	mouse down	mouse up	StartRect(,,,)
			EndRect(,,,)

Table 3 (Page 1 of 2). Commands recorded by the tracker in *Prose II*, listed alphabetically

Command Name	*When was the start time recorded for this command?*	*When was the stop time recorded for this command?*	*Parameters: "–" indicates no parameters*
MapSize	mouse down	mouse up	StartRect(,,,) EndRect(,,,)
MapWindow	mouse down	mouse up	OPEN CLOSE
MapWindowRoam	mouse down	mouse up	StartRect(,,,) EndRect(,,,)
MapWindowZoom	mouse down	mouse up	StartRect(,,,) EndRect(,,,)
MoveNode	mouse down in a node	mouse up	ID() StartPt(,) EndPt(,) INVALID
NewWorkspace	menu selection	instantaneous	– CANCELED
OpenWorkspace	menu selection	select any dialog button	File() Format() CANCELED
OutlineWindowMove	mouse down	mouse up	StartRect(,,,) EndRect(,,,)
OutlineWindowSize	mouse down	mouse up	StartRect(,,,) EndRect(,,,)
OutlineWindow	mouse down	mouse up	OPEN CLOSE
Pause	end of previous command	start of next command	–
SaveWorkspace	menu selection	select any dialog button	File() Format() CANCELED
Scramble	menu selection	instantaneous	– CANCELED
SetDeleteMode	menu selection	instantaneous	ON OFF
SetTidyMode	menu selection	instantaneous	ON OFF
SystemIcon	mouse down	mouse up	–
SystemMove	mouse down	mouse up	StartRect(,,,) EndRect(,,,)
SystemSize	mouse down	mouse up	StartRect(,,,) EndRect(,,,)
SystemZoom	menu selection	instantaneous	–
TidyWorkspace	menu selection	instantaneous	–
TreeGrow	press the gray "+" key while in Tidy mode	instantaneous	–
TreeShrink	press the gray "–" key while in Tidy mode	instantaneous	–

4.1.2 Implementation and Operation of the Tracker

The tracker was implemented by adding a one-line *C* macro at appropriate places in the *Prose II* source code. This line of code was placed at places where it would capture the appropriate elapsed time for each command.

```
#define P2mInformTracker(cmd,ps,lp)    if(fRecordPlayStatus==IDDRECORD)\
        (void)SendMessage(hWndTracker,(unsigned)(cmd),(WORD)(ps),(LONG)(lp))
```

Figure 13. The *C* macro definition used in *Prose II* to contact the tracker

Figure 13 on page 55 shows the source code for this *C* macro, P2mInformTracker. A sample invocation is shown in Figure 14 on page 55. This macro was inserted at the appropriate places in the *Prose II* source code for each of the 39 commands. This macro takes three parameters: a constant containing a command identifier and whether this is its start or finish, a Boolean value indicating whether this is a primary or secondary command, and a pointer to any parameter information needed for the protocol record.

```
BOOL P2PostorderMove(LPNODE lpThisNode,
       BOOL bFirstTime,    /* Marks the first call of this recursive proc */
       POINT ptDistanceMoved)
{

    if (lpThisNode) {
        /* update the time that this node was last touched */
        P2mGetTime(lpThisNode->lLastUpdateTime);

#ifdef BUILDTRACKER
        /* only mark the start if this is a secondary command */
        if (!(bPrimaryCmd && bFirstTime))
            P2mInformTracker(P2_MOVE_NODE_START,
                             FALSE, /* this is a secondary command */
                             (LPNODE)P2mPoint2Long(lpThisNode->pt));
#endif

        /* update the new location of this node */
        lpThisNode->pt.x += ptDistanceMoved.x;
        lpThisNode->pt.y += ptDistanceMoved.y;

#ifdef BUILDTRACKER
        P2mInformTracker(P2_MOVE_NODE_END,(bPrimaryCmd & bFirstTime),lpThisNode);
#endif

        /* this implements the post-order walk */
        if (P2HasChild(lpThisNode))
            P2PostorderMove(P2FirstChild(lpThisNode), FALSE, ptDistanceMoved);
        if ((!bFirstTime) && (P2HasRightSibling(lpThisNode)))
            P2PostorderMove(P2RightSibling(lpThisNode), FALSE, ptDistanceMoved);

        return(TRUE); /* this node has been moved */
    }
    else
        return(FALSE);  /* no further to go; this was not a node */
}
```

Figure 14. Example *C* code, inserted in *Prose II* to implement the tracker

Figure 14 on page 55 shows the source code, in *C*, inserted in the procedure used to implement the MoveNode command. The tracker is contacted by the statements surrounded by the pairs of *C* preprocessor statements

```
#ifdef BUILDTRACKER
:
#endif
```

This allowed the tracker to be compiled into *Prose II* or omitted by setting compiler flags. Thus, the decision of whether to include the tracker is made at compile time, not at runtime.

In my first design for the tracker, the code that writes the commands and parameters to the recording file was to be an asynchronous *Windows* process. *Prose II* would send a message to that tracking process each time a command was started or completed. The tracking process could have a low priority so that the file I/O for the tracking would not interrupt a user. Since *Windows* uses a message-passing process model, such a design would be clean and straightforward to implement.

I rejected this asynchronous design after much work because of two reasons:

1. Timing problems were difficult to overcome.

   The design required a great deal of state information to be saved in some messages because of possible race conditions. A simple example of this problem occurs when a node is deleted before its creation is recorded by the tracker process. More complex problems involved primary commands that had many potential secondary commands; for example, a single delete command may delete a large tree.

   Note that this could be eliminated if the tracker has the highest, preemptive priority in the system, but this defeats the intention of operating the tracker at the lowest priority.

2. A synchronous tracker did not appear to affect users' perceived performance of the system.

   To solve the timing problem, I implemented the tracker as a synchronous interface, using the same message passing interface.[10] No perceptible slowdown induced by the tracker was seen, nor did any subject complain or comment on the performance (except that they thought the overall system was fast).

Thus, the tracker, as finally implemented, uses a synchronous interface. This reaffirmed a favorite software engineering principle: if you can afford the luxury, try alternate designs and see what happens.

There was a problem with the minimal elapsed time between consecutive commands being less than the operating system's clock resolution, which is 0.06 seconds in DOS. Whenever this occurred, the tracker originally showed the elapsed time as 0.00 seconds, which was difficult to handle in the parser. Therefore, I revised the tracker to force a minimal elapsed time of 0.01 seconds for each primary command. This could have caused a problem if more than six primary commands were accomplished in less than 0.06 seconds, but this never occurred.

I did not provide a means of saving or checkpointing a protocol record file with the *Prose II* tracker. In DOS, a file is not committed to disk until it is explicitly closed. So, while a user could save a <u>workspace</u> file at any time (forcing it to be written to disk and closed), there was no mechanism for writing the protocol record file to disk until the session was ended. If a user's computer crashes or is re-booted for any reason during a session, the protocol record is lost. Some good times to do checkpointing would have been: 1) whenever the user saves the workspace file, or 2) during long pauses (a suitable length for a long pause is discussed in later sections). I did not consider automatic checkpointing of protocol files until after the study was completed; no subjects reported lost sessions.

Among the drawbacks of distributing this tracker without other forms of concurrent protocol collection (*e.g.*, think-aloud or videotape) is that there is no knowledge of what users are doing or thinking when they are not using the system. Also, users cannot readily contact those doing the

[10] In the top of Figure 13, I replaced the *Windows* function `PostMessage` (which is asynchronous) with `SendMessage` (which is synchronous), as shown.

study to comment or complain. I built a couple of extra "hooks" into the tracker to give users a way to step outside the system and leave additional information in the protocol record.

Commentary

By pressing the "F2" function key while using *Prose II*, subjects could leave comments in the protocol record at any time. This allows subjects to record their reactions to using this software when they occur. Something an automatic recorder cannot easily capture is a user's "attitude" and "intentions"; this function key made such a capability available, although its use was entirely voluntary.

Section 5.3.3.7 on page 118 lists all the comments left by the subjects during the study. It is interesting to see that this command was used for many different purposes by the subjects.

Focus of Attention

By pressing the "F3" function key while using *Prose II*, subjects could indicate that they were taking a lengthy mental or physical break during a session. Examples of this are answering the telephone or going to lunch. When this key is pressed, a message box appears in the main *Prose II* window indicating a lengthy pause; selecting OK in the message box returns the subject to normal operation.

With these function keys, *Prose II* provides subjects with commands for interacting with the automated tracker, and thus with the eventual analyst (and presumably the software developer).

This section has described the operation of the tracker, the handling of each user action, and the generation of formatted recordings. These protocol records capture user actions at the command level in a consistent format, easily readable by both humans and computer programs. The next section describes a grammar, and subsequent computer program, for automatically analyzing and summarizing a protocol record.

4.2 Automating Protocol Analysis with a Parser

Having the protocol records captured in a regular form—readable by humans and computers—made their analysis much more tractable than the handcoded methods discussed in the second chapter. This section describes the grammar developed to describe users' interaction with the system. It also describes a parser based on that grammar; the parser was the principal tool for automating the analysis of session protocols.

The grammar and parser described here are designed to categorize portions of an input string. The input string consists of symbols that correspond to each line of a protocol record file. These categorizations are represented by the names associated with the intermediate nodes in the resulting parse trees.

The model of user behavior defined by the grammar is based on the work of Card, Moran, and Newell (1983), primarily their *ICARUS* study (which was described in Section 2.3.2.1 on page 29). The remainder of this chapter describes how the grammar and parser works. A summary of the key decisions made in producing these rules and symbols can be found in Section 6.2.2 on page 133.

The grammar is presented below in stages; the description follows a breadth-first strategy of first providing an overview and then providing successively finer layers of detail. An important concept for understanding the grammar is that it is written in terms of levels. Separate sets of rules are responsible for identifying sequences of symbols in one level and outputting more general symbols representing that sequence in another higher level. Thus, the output of one level is the input for another level.

First, the grammar symbols, both terminals and non-terminals, are introduced. A general overview describes the relationships among the levels and the symbols seen at each level. Each of the symbols is listed in table form. Next, the overall structure of the grammar is introduced, with an emphasis on how the various levels relate. Each level is generated with a separate state machine (and accompanying logic); these machines are tied together by an Augmented Transition Network (ATN) structure. Third, a detailed specification of each state machine is presented, along with the tests and internal structures that further specify the ATN. Finally, a parser is described that implements this grammar. In this study, the parser serves as the final authoritative definition of the grammar.

4.2.1 Grammar Symbols

The grammar is described in terms of four principal components: sessions, phases, episodes, and commands.

- A protocol record for one session is partitioned into phases.
  - Each phase lasts several minutes.
  - A single type of activity predominates in each phase.
- A phase consists of a sequence of cognitive tasks, called episodes.
  - There is an underlying pulse in the sequence of episodes in a phase, in which episodes of tree construction and writing alternate with periods of inaction and housekeeping.
  - These periods of housekeeping and inaction (here called housekeeping episodes) delimit the periods of tree construction, writing, and editing (called constructive episodes).
- The constructive episodes and housekeeping episodes are composed of sequences of individual user commands and intervening pauses.
  - Each command lasts a few seconds or less.
  - Commands are the atomic elements of the protocol record.

Figure 15 on page 59 illustrates a generic parse tree. It shows a session composed of two phases, three episodes, and eight total commands. The symbols for the pauses between each pair of commands are not shown in this figure. An actual parse tree, showing actual commands, pauses, episodes, phases, and a session is shown in Figure 2 on page 6.

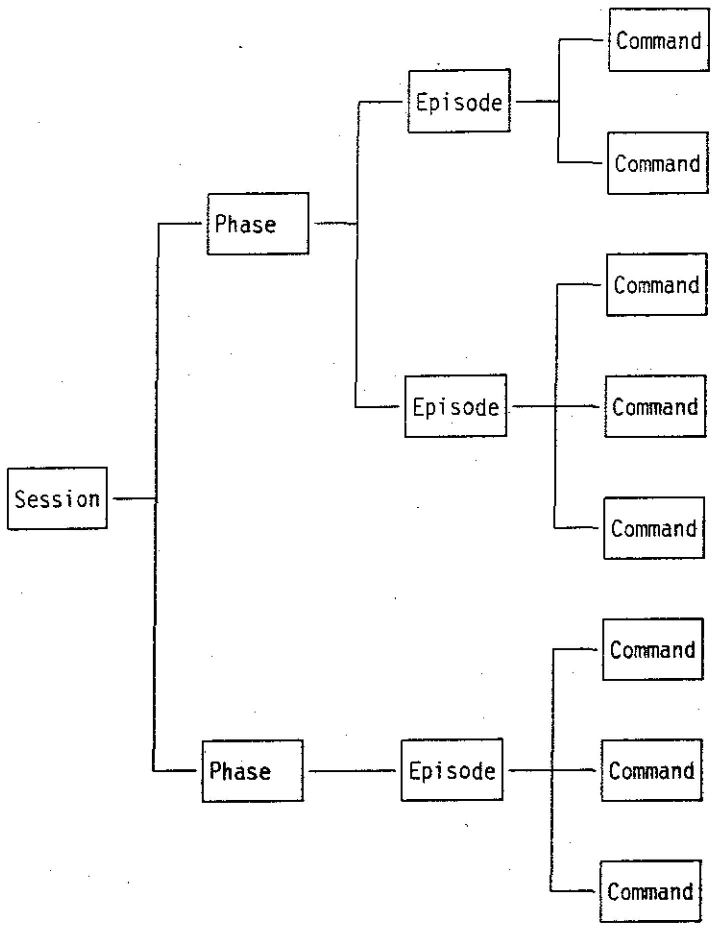


Figure 15. A simple, generic parse tree

The terminal symbols in this grammar correspond to individual user commands. A sequence of commands (and their parameters) is classified as an episode, the first level of non-terminal symbols. Table 4 on page 60 classifies the 39 different *Prose II* commands, indicating whether a particular command is parsed as part of a housekeeping episode or a constructive episode. These 39 commands, plus Pause, comprise the terminal symbols for the *Prose II* grammar.

Table 4. Classification of the *Prose II* commands, grouped by type of episode. Each of these commands was introduced inTable 3 on page 53 .

Commands in Housekeeping Episodes	*Commands in Constructive Episodes*
Pause	BreakLink
ChangeDefault	BreakAllLinks
ClipboardCopy	CopyNode
GoTo	CreateNode
MainWindowReset	DeleteNode
MainWindowZoom	EditLabel
MapMove	EditNode
MapSize	LinkNodes
MapWindow	MoveNode
MapWindowRoam	Scramble
MapWindowZoom	
OutlineWindowMove	
OutlineWindowSize	
OutlineWindow	
SaveWorkspace	
SetDeleteMode	
SetTidyMode	
SystemMove	
SystemSize	
SystemWindow	
SystemZoom	
TidyWorkspace	
TreeGrow	
TreeShrink	
LongPause	
ClearDrawing	
Comment	
HelpRequest	
LeaveProseII	
NewWorkspace	
OpenWorkspace	
SystemIcon	

These two classes of episodes are divided into individual types of episodes that further classify the sequence of commands. For example, some housekeeping episodes consist solely of session maintenance, such as saving the workspace and closing unneeded windows, while others involve a change in the focus of operations, such as moving to other regions of the workspace and zooming in on a specific region.

Table 5 on page 61 classifies the episodes identified by the grammar from sequences of *Prose II* commands. These 20 different episodes comprise the first level of non-terminal symbols for the grammar. The names of the episodes describe the types of activity being done in that episode. We devised the names by successive cycles of listing the representative kinds of activity we expected to see during sessions, and then observing sequences of commands that indicated each particular kind of episode.

Table 5. Taxonomy of *Prose II* episodes

Housekeeping Episodes	*Constructive Episodes*
Cleanup	Assembled trees
Cleanup and Take Stock	Broke existing links
Help Request	Created new trees
Long Pause	Created solo nodes
Medium Pause	Deleted nodes
Refocus	Edited existing nodes
Take Stock	Grew existing trees
Tracker Comment	Hooked existing nodes to trees
	Moved existing nodes
	Start over
	Unproductive work

Phases of work span one or more constructive episodes, including any intervening housekeeping episodes. They represent the broad categories of work executed by users of *Prose II*. For example, when creating a document from scratch, users often begin with several episodes of exploration, where they are laying out their initial ideas, labeling them, and rearranging them before connecting them into larger structures.

The idea of phases of work was introduced by Card, Moran, and Newell (1983). Table 6 on page 61 classifies the phases identified by the grammar from sequences of episodes. These 7 different phases comprise the third level of non-terminal symbols for this grammar. Their names and descriptions were based on writing research and on the specific operations and capabilities of *Prose II*.

Table 6. Taxonomy of *Prose II* phases

Phases identified in Prose II sessions
Bottom Up Construction
Define Hierarchies
Document Revision
Exploration
New Workspace
Top Down Construction
Tree Structure Revision

Finally, the grammar describes a session as consisting of one or more phases. A single session is the highest non-terminal symbol in the grammar. A session could be viewed as the start symbol for a top-down parse.

4.2.2 Design and Operation of the Grammar

This subsection contains the general description of the rules that map sequences in one level of the grammar to individual symbols in the next higher level. At this degree of detail, the architecture of the grammar is influenced by the architecture of the parsing program that implements it. Thus, the grammar rules are described for a particular level in terms of a pass for the parsing program that carries out that part of the overall analysis.

The grammar rules are described in their relationship to five passes. Passes 0 and 1 are actually preprocessing steps, and thus their operation and effect are not discussed in this overview. Passes 2 and 3 map commands to episodes. Pass 4 maps episodes to phases which, in turn, are gathered to form the session.

The same generic parse tree shown in Figure 15 on page 59 is shown below, but associating each level of the parse tree with the grammar passes that handle its categorization.

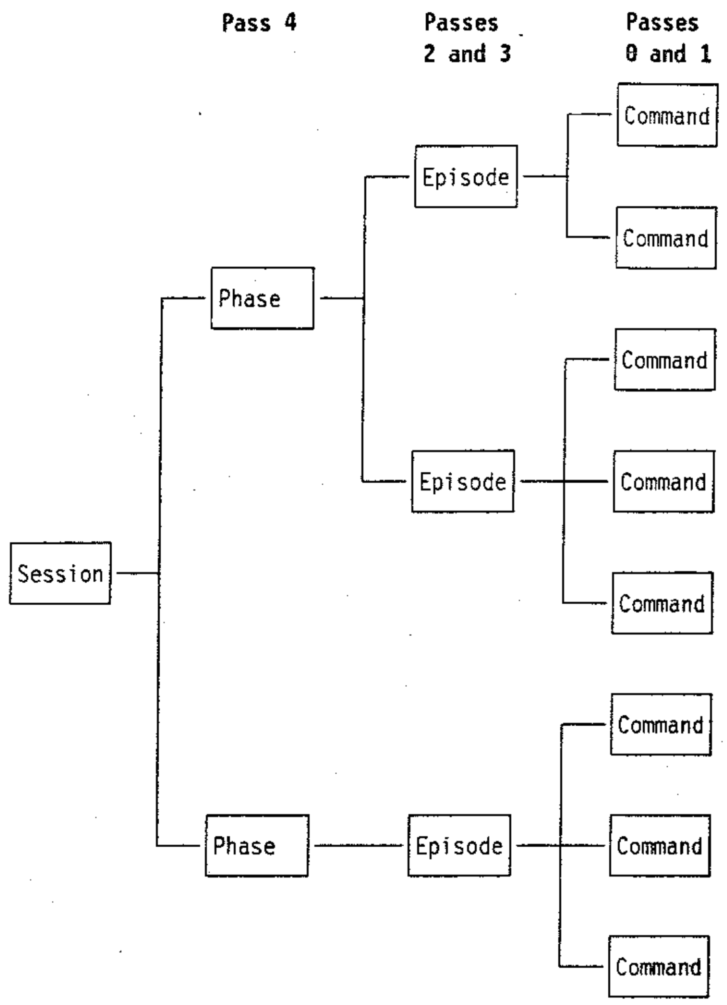


Figure 16. The same parse tree, showing corresponding grammar passes for each level. Implementation details of the parse have been omitted. For example, commands are shown as single symbols; they actually begin as text strings in a file that are reduced to single symbols by Pass 0. Also, the pauses between commands are not shown.

No portion of the overall grammar needs power greater than that of a context-sensitive grammar to describe its operation. Each pass, with its corresponding grammar, is introduced below.

Pass 0 Translates a human-readable protocol record into 16-tuples, one for each primary and secondary command. (n-tuples are explained, with an example, in Figure 21 on page 69.)

Type 2 power: a context-free grammar can describe this pass. The strings of English that it reads have a context-free syntax: for example, parentheses must be properly nested for label parameters.

Pass 1 Gathers selected subsequences into a single 16-tuple.

Type 1 power: a context-sensitive grammar can describe this pass. Pass 1 reads in an arbitrarily long sequence of symbols and decides whether to output that string or a single replacement symbol. If the original string is to be output, it must be in FIFO order, which precludes using a push-down automaton.

Pass 2 Segments the constructive episodes by determining and characterizing the housekeeping episodes.

Type 3 power: a finite-state machine can describe this pass. Pass 2 reads in a string of input symbols, and generates a single output symbol corresponding to its current state at the end of a housekeeping episode.

Pass 3 Characterizes the constructive episodes.

Type 1 power: a context-sensitive grammar can describe this pass. Pass 3 builds a great deal of context information (for example, the current tree hierarchy) as it reads each new symbol. It generates one or more output symbols at the start of the next housekeeping episode or at the end of the input string. The amount of context information it keeps is proportional to the length of the entire input string.

Pass 4 Distinguishes phases in the sequence of constructive episodes.

Type 3 power: a finite-state machine can describe this pass. Pass 4 reads in a string of input symbols, and outputs a single symbol whenever the state machine indicates a reset.

The overall relation of these passes can be seen in Figure 17 on page 63.

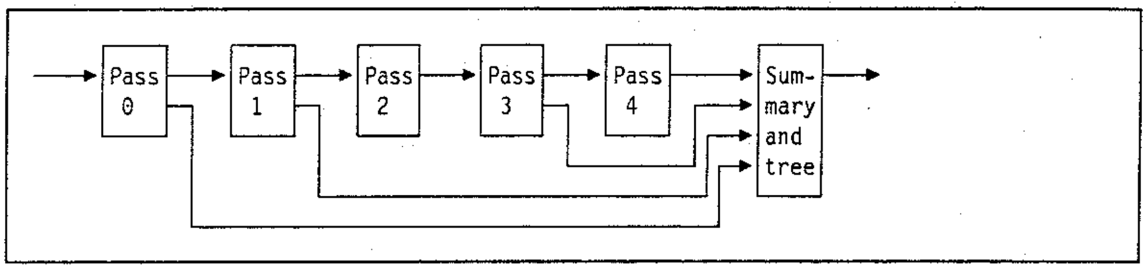


Figure 17. Data flow diagram for the parser

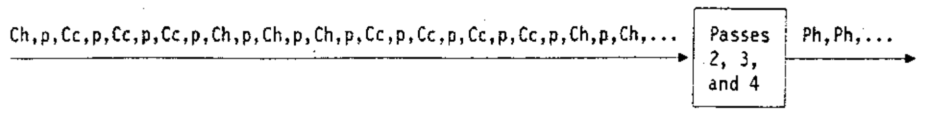
The input to Pass 0 is an actual protocol record file for one user's session. The summary and parse tree are constructed using the intermediate results of passes 0, 1, 3, and 4. Otherwise, pass 1 uses, as its input, the output of Pass 0, and so on through each of the other passes.

A lot of context information is carried along in these passes, but at any point in the input sequence, the current state of each pass is always valid and can be determined. Hence, every state of every pass can be considered one step away from a stop state, because a user could have been interrupted or the session could have been ended at any time.

Three passes do most of the important grammatical analyses: passes 2, 3, and 4. To describe the details of these passes, additional terminology will be introduced here to make the diagrams that follow more succinct.

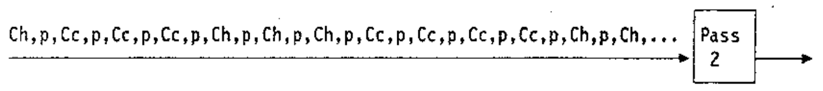
p	designates a pause
Ch	designates a housekeeping command
Cc	designates a constructive command
Eh	designates a housekeeping episode
Ec	designates a constructive episode
Ph	designates a phase symbol

For these three important passes, the parser can be illustrated by the following example diagram. As a black box, these three passes reduce an arbitrary sequence of command symbols to a sequence of phase symbols.

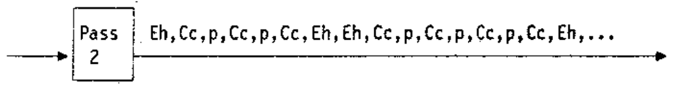

Ch,p,Cc,p,Cc,p,Cc,p,Ch,p,Ch,p,Ch,p,Cc,p,Cc,p,Cc,p,Cc,p,Ch,p,Ch,...
Passes 2, 3, and 4
Ph,Ph,...

In more detail, these three passes perform discrete transformations on their respective input streams.

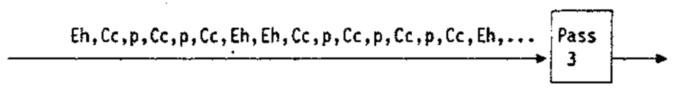
Pass 2: compresses sequences of housekeeping commands into one or more housekeeping episodes. For example, an input sequence like the following:


Ch,p,Cc,p,Cc,p,Cc,p,Ch,p,Ch,p,Ch,p,Cc,p,Cc,p,Cc,p,Cc,p,Ch,p,Ch,...
Pass 2

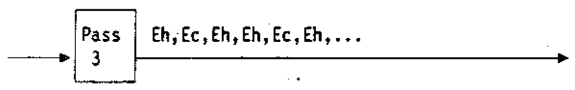
would produce an output sequence where strings of housekeeping commands (Ch) are reduced to housekeeping episodes (Eh).



Pass 3: compresses sequences of constructive commands into constructive episodes. For example, an input sequence like the following:

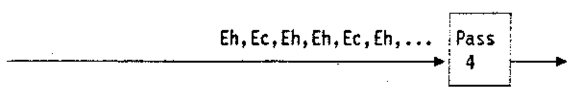


would produce an output sequence where strings of constructive commands (Cc) are reduced to constructive episodes (Ec). Since Eh symbols are passed through unchanged, the output of pass 3 is a stream of interspersed Eh and Ec symbols.



Because they operate on different input symbols, passes 2 and 3 could have been combined into a single pass. Separating them made their description and implementation more modular. It also clearly separated the Type 3 grammar used for Pass 2 from the Type 1 grammar used for Pass 3.

Pass 4: compresses sequences of related constructive and housekeeping episodes into phases. For example, an input sequence like the following:



would produce an output sequence where strings of episodes (Eh and Ec) are reduced to phases (Ph).

Each of these three passes are previewed in more detail below. Rudimentary state diagrams are used to show the overall handling of classes of symbols; the actual details of the handling of specific symbols and their parameters are described later in this chapter, as are the implementation details.

4.2.2.1 Overview of Pass 2

Pass 2 categorizes sequences of housekeeping commands as housekeeping episodes. Sequences of housekeeping commands (Ch) are delimited by sequences of constructive commands (Cc). After a sequence of housekeeping commands, any constructive command causes a symbol corresponding to a housekeeping episode (Eh) to be output. Constructive commands pass unchanged through Pass 2. Thus, the effect of Pass 2 is to consolidate Ch's into Eh's, leaving the Cc symbols alone.

An overview of Pass 2 is shown in the following state diagram.

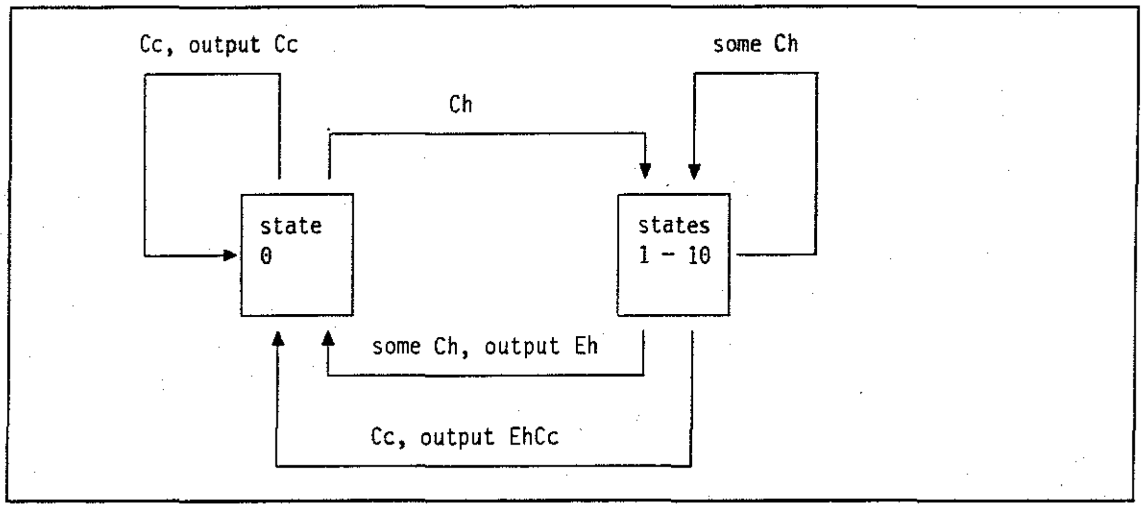


Figure 18. State machine overview for Pass 2

States 1 through 10 of this state machine operate similarly, and are thus represented as a single node in Figure 18 on page 65. For each of these states, an input symbol causes the machine to either stay in the same state, make a transition to state 0, or make a transition to some other state. Depending on their value and the current state, some members of Ch cause a transition to state 0 and generate an output symbol. The remaining members of Ch may or may not cause a state transition, but they generate no output symbol. Only when a transition is made to state 0 is an output symbol generated.

Constructive commands (Cc) that follow housekeeping commands (Ch) always cause a transition to state 0 and cause an output symbol to be generated. Constructive commands in state 0 cause no state transition, although they are directly output as output symbols.

The rules that describe the different actions are described in Section 4.2.3.3 on page 73.

4.2.2.2 Overview of Pass 3

Pass 3 categorizes sequences of constructive commands as constructive episodes. Sequences of constructive commands (Cc) are delimited by the housekeeping episodes (Eh) from pass 2. After a sequence of constructive commands, any housekeeping episode causes one or more constructive episodes (Ec) to be output. Although they cause Ec output symbols to be generated, Eh symbols themselves are passed through unchanged.

An overview of this pass is shown in the following state diagram.

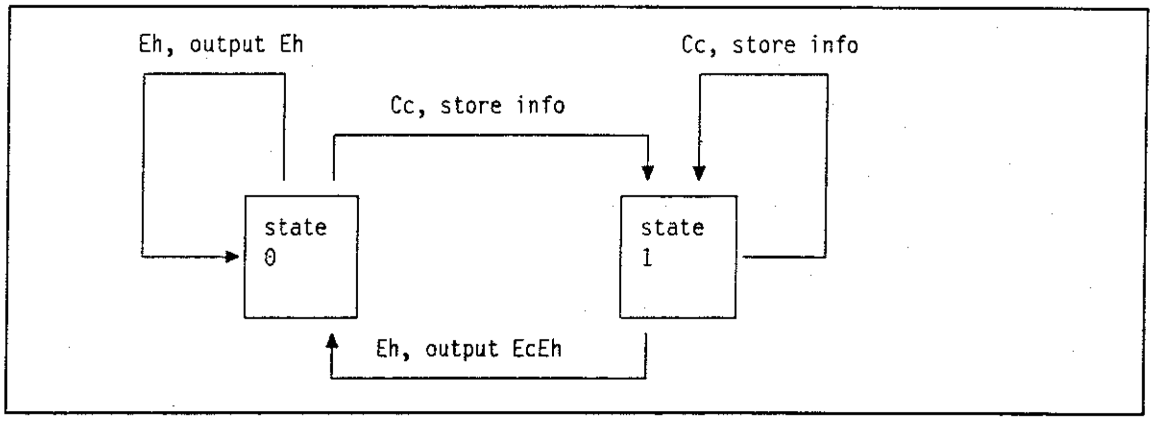


Figure 19. State machine overview for Pass 3

Pass 3 is strongly context dependent. Its operation is unconventional, as are many Type 1 parsers. Its overall state operation appears simple; Cc symbols are accumulated by Pass 3 until an Eh symbol is encountered. Each Eh symbol causes at least one output symbol to be generated. Determining which output symbol(s) to generate, characterizing the constructive episode, is the complex part of Pass 3's operation.

store info and **output Ec** are extensive operations that involve building and resetting contextual information about the order of operations and the ongoing structures being built in this pass.

store info consists of the following operations:

1. Look at the exact contents of Cc, including what command this is, its timestamp, and the value of its parameters. In some cases, increment counters, counting such things as the number of new nodes created in this constructive episode.
2. Add the command to a history list. This is a list of the Cc symbols seen since the last Eh.
3. Update the global data objects. Since the user was using the system to build nodes and trees, the global data objects here consist of an internal representation of all the nodes built so far and their position in trees.

output Ec consists of the following operations:

1. Decide the primary (and any secondary) Ec symbols to be output, based on
   - counter values
   - the history list for the episode
   - the state of the global data objects
   - a set of production system rules, which are conditional statements evaluated in a pre-defined order.
2. Update the global data objects, if necessary.
3. Reset the history list and all the counters.

Each time a constructive episode is identified, at least one output symbol is generated. However, because of the complexity of the sequence of operations in an episode, secondary descriptions of the episode may also be generated. For example, in a single episode a user might both construct a tree and delete a set of standalone nodes. A primary Ec symbol is output, along with a secondary symbol. Pass 4 uses these primary and secondary constructive episode symbols (which carry the same timestamp) to further classify the phase. These two different types of symbols are described as Ecp and Ecs in the discussion of Pass 4 that follows.

The detailed operation of Pass 3 is described in Section 4.2.3.4 on page 78.

4.2.2.3 Overview of Pass 4

Pass 4 categorizes sequences of constructive episodes as phases. Groups of similar input symbols map onto a single symbol. Of the input symbols, Pass 4 looks only at the primary constructive episodes. Housekeeping episodes, which were used to delimit the constructive episodes in Pass 3, do not affect the phase symbol output by Pass 4, since they do not describe constructive work on a document.

An overview of this pass is shown in the following state diagram.

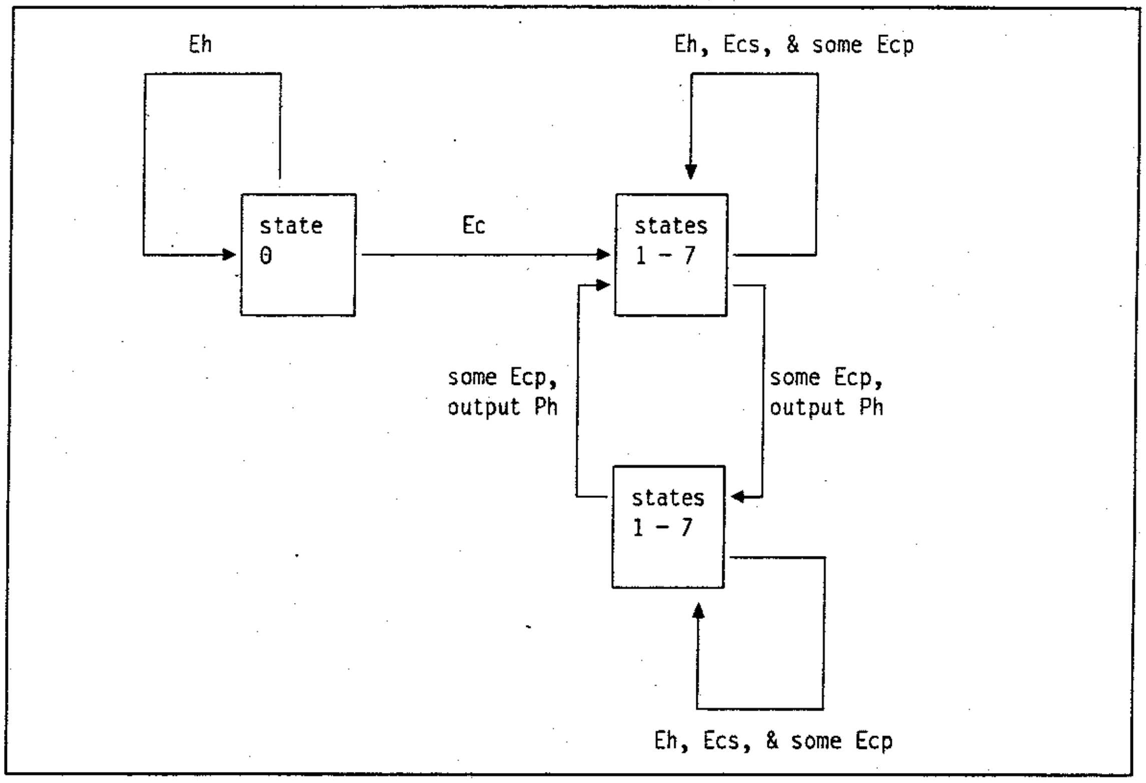


Figure 20. State machine overview for Pass 4

Pass 4 consists of 8 distinct states, here reduced to 3. A full state diagram for pass 4 is a mesh, with its 29 inputs causing different transitions in the 8 states. After seeing a constructive episode, Pass 4 never returns to state 0. Housekeeping episodes (Eh) never cause a state transition.

There are two types of constructive episodes, Ecp (primary constructive episodes) and Ecs (secondary constructive episodes). Depending on their value and the current state, some members of Ecp cause a state transition; the remaining members of Ecp cause no state transition. If a transition to a new state is made because of an Ecp symbol, a phase symbol (Ph) is output; the phase describes the state that was just left.

Depending on their value and the current state, members of Ecs may or may not cause a state transition. As a secondary constructive episode, these symbols never directly cause an output symbol to be generated, but may change the current state, and thus influence what the next output symbol will be.

The detailed operation of Pass 4 is described in Section 4.2.3.5 on page 83.

4.2.2.4 Overview Summary

Passes 2, 3, and 4 form the core of the grammar. They specify the transformation from individual command symbols into a sequence of phase symbols that define a session. The subsection that follows contains more detail on the construction of these passes, as well as the preliminary passes that read and transform the text in actual protocol records. Pass 0 does the translation from text into grammar symbols. Pass 1 clarifies the association between certain command and their parameters, as well as reorganizing a few specific sequences of commands. Pass 1 is necessary because of sequencing anomalies in the tracker. For these passes, as well as passes 2 through 4, the meaning and usage of each input and output symbol is described. The extended logic for processing these symbols is then presented, showing the transitions for each symbol in each state.

4.2.3 Details of the Grammar Operation

This subsection provides the detailed state machines and grammar elements introduced in the preceding section. Appendix B on page 170 contains a full example of a parse, showing the original input file and each of the intermediate files for one of the protocol records in the study.

In passes 2, 3, and 4 of the grammar, a sequence of commands is collapsed into an episode, and these episodes are similarly collapsed into a phase. To preserve sufficient status of what was seen, the grammar carries some contextual information forward from one pass to another. For example, a user might have saved a workspace during an episode where the predominant activity was creating and editing nodes. The fact that a save occurred must be carried forward.

To carry this information between passes, one of the fields carried between the passes is a collection of Boolean flags. For example, a Boolean flag indicates whether a workspace was saved during that episode. As another example, Pass 0 sets a Boolean flag whenever the current command is invalid or canceled. Similarly, Pass 3 of the grammar looks at how many of the commands in an episode were invalid. When it sees commands with the "invalid command" flag set, it increments its counter of invalid commands. These thirteen flags are listed below.

1. Is the Main Window showing?
2. Is the Map Window showing?
3. Is the Outline Window showing?
4. Is Delete Mode on?
5. Is Tidy Mode on?
6. Was the workspace tidied?
7. Is this the first editing of this label?
8. Is this the first editing of this node's contents?
9. Was this command canceled?
10. Was this command invalid?
11. Was this constructive episode a complex one?
12. Was a file saved in this episode?
13. Did the subject start over in the workspace?

The flags are first assembled in Pass 0, which translates the protocol record text into grammar symbols. One example of this translation is the "`-- invalid --`," discussed in Section 4.1.1 on page 49. A grammar symbol for an invalid command would consist of the command and its timestamp, yet that flag indicating it is invalid would be set by Pass 0.

4.2.3.1 Pass 0: Translating the Tracker File

This pass of the grammar carries out the first step in analyzing a user's session: it translates a text file, *i.e.*, a protocol record, into distinct grammar symbols used in the later passes. Each line of the input file is translated into a single output symbol, consisting of up to 16 numbers; this is known as a *16-tuple*. This group of numbers captures all the relevant information about one single command. Thus, each line in a protocol record is translated into a single 16-tuple, whether it is a primary command, a secondary command, or a pause. An example of one of these 16-tuples is shown in Figure 21 on page 69.

An example line from a protocol record, indicating a pause:

```
39:52.00  39:59.91    7.91  PAUSE
```

The corresponding 16-tuple output generated by Pass 0:

```
0257, 0239200, 0239991, 0000791, 0001, 0000, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

Figure 21. An example protocol line and the corresponding 16-tuple generated by Pass 0

Reading across the 16-tuple line in Figure 21 on page 69: "0257" is the internal identifier used by the parser for Pause; "0239200" and "0239991" are the start and stop times of this pause in hundredths of seconds; "0000791" is the elapsed time of this pause in hundredths of seconds; "0001" indicates that this 16-tuple represents one command (this field is not important until later passes, where it counts the number of commands in an episode); "0000" indicates the thirteen bit-significant Boolean flags carried between passes. The remaining ten fields are specific to the parameters used in each command (they would contain such fields as node ID and coordinates).

One concern was that users might alter a protocol record file in such a way that it could not be parsed. If this occurred in a simple manner, I expected to be able to correct the problem on an individual basis. No protocol records were altered by any of the subjects, but, as mentioned in Section 4.2 on page 57, one subject did attach a paragraph of text to the bottom of a protocol record. The syntactic analyzer of Pass 0 terminated when it encountered the beginning of this paragraph. I moved the paragraph to a separate file with a text editor, which corrected the problem.

Figure 22 on page 70 shows an example of the input to Pass 0, and the corresponding output.

```
    +-------------------------------------------------------------------+
    | Prose II Session Recording                 Sat Nov 26 09:39:52 1988 |
    | File: C:~RCD3AIA.TMP                                               |
    +-------------------------------------------------------------------+
    | Start     Stop      Time  Operator              Parameters          |
    |min:sec   min:sec    sec                                            |
    +-------------------------------------------------------------------+
 1  39:52.00  39:59.91   7.91  PAUSE
 2  39:59.91  40:00.07   0.16  OpenWorkspace         File(`IBMSC.SCR`) Format(`.PR2`)
 3                             + CreateNode          ID(1)
 4                             + CreateNode          ID(2)
 5  40:00.07  40:02.87   2.80  LeaveProseII          ID(0)
 6  40:02.87  40:03.15   0.28  PAUSE
 7  40:03.15  40:16.83  13.68  PAUSE
 8  40:16.83  40:35.61  18.78  OpenWorkspace         File(`IBMSC.PR2`) Format(`.PR2`)
 9                             + CreateNode          ID(1)
10  40:35.61  40:58.84  23.23  PAUSE
11  40:58.84  41:15.60  16.76  PAUSE
12  41:15.60  41:18.62   3.02  PAUSE
13  41:18.62  41:19.50   0.88  EditNode              ID(1) Editor(`notepad.exe`) File(`IBMSC001.PR2`)
14  41:19.50  41:39.98  20.48  LeaveProseII          ID(0)
15  41:39.98  41:53.11  13.13  PAUSE
    --------------------------------------------------------------------------------------------
 1  0257,  0239200,  0239991,  0000791, 0001, 0000 0 0 0 0 0 0 0 0 0
 2  0286,  0239991,  0240007,  0000016, 0001, 0000 0 0 0 0 0 0 0 0 0
 3  0264, -0000001, -0000001, -0000001, 0001, 0000 1 0 0 0 0 0 0 0 0
 4  0264, -0000001, -0000001, -0000001, 0001, 0000 2 0 0 0 0 0 0 0 0
 5  0275,  0240007,  0240287,  0000280, 0001, 0000 0 0 0 0 0 0 0 0 0
 6  0257,  0240287,  0240315,  0000028, 0001, 0000 0 0 0 0 0 0 0 0 0
 7  0257,  0240315,  0241683,  0001368, 0001, 0000 0 0 0 0 0 0 0 0 0
 8  0286,  0241683,  0243561,  0001878, 0001, 0000 0 0 0 0 0 0 0 0 0
 9  0264, -0000001, -0000001, -0000001, 0001, 0000 1 0 0 0 0 0 0 0 0
10  0257,  0243561,  0245884,  0002323, 0001, 0000 0 0 0 0 0 0 0 0 0
11  0257,  0245884,  0247560,  0001676, 0001, 0000 0 0 0 0 0 0 0 0 0
12  0257,  0247560,  0247862,  0000302, 0001, 0000 0 0 0 0 0 0 0 0 0
13  0271,  0247862,  0247950,  0000088, 0001, 0000 1 0 0 0 0 0 0 0 0
14  0275,  0247950,  0249998,  0002048, 0001, 0000 0 0 0 0 0 0 0 0 0
15  0257,  0249998,  0251311,  0001313, 0001, 0000 0 0 0 0 0 0 0 0 0
```

Figure 22. An example of Pass 0 input and output. The bottom half of this figure illustrates the results of running Pass 0 against the protocol record show in the top half. The fifteen lines are numbered to show the one-to-one correspondence.

An extensive example of the output from Pass 0 is shown in appendix B.2 on page 178.

4.2.3.2 Pass 1: Combining Common Sequences

This pass of the grammar carries out some minor corrective surgery on the sequence of grammar symbols representing the commands. The grammar described here is dependent on the sequence of the input symbols it received. Pass 1 serves to correct some minor anomalies in the sequence of input symbols. It also widens the set of symbols considered by later passes; for example it changes the symbol SetTidyMode with parameter Off into a new symbol: SetTidyModeOff.

Pass 1 is the simplest of the grammar passes. It was added late in the process of developing the grammar—long after the protocol collection had been completed. In this pass, sequences of commands that could have been represented with a single symbol are combined into that single symbol. For example, setting TidyMode on and then off is equivalent to executing the TidyWorkspace command. This pass handles side-effects of the command structure of *Prose II* such as this example. The conversions done in Pass 1 cannot easily be done in later passes because the symbols involved deal with both housekeeping and constructive commands (these are discussed in Section 4.2.3.3 on page 73).

Pass 1 handles only three cases:

1. When the EditNode and LeaveProseII commands are adjacent, they signal that the subject has started an editing program to edit the contents of a node. Pass 1 maps either of the following sequences onto the single symbol EditNode.

   - EditNode
   - LeaveProseII

   or

   - LeaveProseII
   - EditNode

2. The sequence SetDeleteModeOn, DeleteNode, and SetDeleteModeOff signals the deletion of a single node. *Prose II* also has a single command for deleting one node, so this longer sequence is mapped onto the single symbol DeleteNode. ("*" represents a Kleene star, indicating this symbol could occur zero or more times.)

   - SetDeleteModeOn
   - pause*
   - DeleteNode
   - pause*
   - SetDeleteModeOff

3. The sequence of SetTidyModeOn followed by SetTidyModeOff serves to tidy the workspace. *Prose II* also has a single command for tidying the workspace, so this longer sequence is replaced by this symbol TidyWorkspace.

   - SetTidyModeOn
   - pause*
   - SetTidyModeOff

A custom lexical analyzer reads the output from Pass 0, combining the commands with their parameters to form unique symbols. It converts the 40 Pass 0 output symbols (39 commands plus Pause) into 50 different symbols; for example, the SetTidyMode command with the parameter "ON" is converted to the internal symbol SetTidyModeOn.

The syntactic analysis of Pass 1 is implemented with a finite-state machine and a FIFO stack. At any time in the parse, many commands can be on the stack. Pass 1 handles the generation of an output symbol in one of two ways: either a single symbol is output and the stack is flushed, or all of the symbols on the stack are output, in FIFO order. Thus, if the state machine so indicates, the symbol from the lexical analyzer is pushed on the stack and another symbol is read. When the state machine indicates the stack should be flushed, either the contents of the stacked are replaced by a single symbol, or each of the symbols on the stack is output.

No processing is done by Pass 1 if the input command is a secondary command, or if the command is canceled or invalid. The output symbol is identical to the input symbol, with no changes to the current state.

Pass 1 Finite-State Machine: The following table describes the states defined for the Pass 1 finite-state machine.

Pass 1 State	*Description of the state*
0	Start state.
1	Saw EditNode, waiting for LoseFocusShort or LoseFocusLong
2	Saw LoseFocusShort or LoseFocusLong
3	Saw SetDeleteModeOn, waiting for DeleteNode
4	Saw DeleteNode, waiting for SetDeleteModeOff
5	Saw SetDeleteModeOff
6	Saw SetTidyModeOn, waiting for SetTidyModeOff
7	Saw SetTidyModeOff
8	Saw LoseFocusShort or LoseFocusLong, waiting for EditNode
9	Saw EditNode

Table 7 (Page 1 of 2). Pass 1 finite-state machine

State Names—>	*Start State*	*Saw Edit-Node*	*Saw Lose-Focus-xxxx*	*Saw Set Delete Mode-On*	*Saw Delete Node*	*Saw Delete Mode-Off*	*Saw Set-Tidy-Mode-On*	*Saw Set-Tidy-Mode-Off*	*Saw Lose-Focus-xxxx*	*Saw Edit-Node*
Inputs *State Numbers—>*	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
BreakAllLinks	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
BreakLink	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
ChangeDefault	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
ClearDrawing	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
ClipboardCopy	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
CopyNode	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
CreateNode	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
DeleteNode	-(B)	0(B)	0(C)	4(A)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
EditLabel	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
EditNode	1(A)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	9(A)	0(C)
GoTo	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
HelpRequest	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
LinkNodes	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
LoseFocusLong	8(A)	2(A)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
LoseFocusShort	8(A)	2(A)	0(C)	-(A)	-(A)	0(D)	-(A)	0(E)	0(B)	0(C)
MainWindowReset	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
MainWindowZoom	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
MapMove	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
MapSize	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
MapWindowClose	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
MapWindowOpen	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
MapWindowRoam	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
MapWindowZoom	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
MoveNode	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
NewWorkspace	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
OpenWorkspace	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
OutlineWindowClose	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
OutlineWindowMove	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
OutlineWindowOpen	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
OutlineWindowSize	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
PauseType0	-(B)	-(A)	0(C)	-(A)	-(A)	0(D)	-(A)	0(E)	0(B)	0(C)
PauseType1	-(B)	-(A)	0(C)	-(A)	-(A)	0(D)	-(A)	0(E)	0(B)	0(C)
PauseType2	-(B)	-(A)	0(C)	-(A)	-(A)	0(D)	-(A)	0(E)	0(B)	0(C)
PauseType3	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
SaveWorkspace	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
Scramble	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
SetDeleteModeOff	-(B)	0(B)	0(C)	0(B)	5(A)	0(D)	0(B)	0(E)	0(B)	0(C)
SetDeleteModeOn	3(A)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
SetTidyModeOff	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	7(A)	0(E)	0(B)	0(C)
SetTidyModeOn	6(A)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
SystemIcon	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
SystemMove	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
SystemSize	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
SystemWindowClose	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
SystemWindowOpen	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
SystemZoom	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
TidyTree	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	-(A)	0(E)	0(B)	0(C)
TidyWorkspace	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	-(A)	0(E)	0(B)	0(C)
TrackerComment	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
TreeGrow	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)

Table 7 (Page 1 of 2). Pass 1 finite-state machine

State Names---->	*Start State*	*Saw Edit-Node*	*Saw Lose-Focus-xxxx*	*Saw Set Delete Mode-On*	*Saw Delete Node*	*Saw Delete Mode-Off*	*Saw Set-Tidy-Mode-On*	*Saw Set-Tidy-Mode-Off*	*Saw Lose-Focus-xxxx*	*Saw Edit-Node*
Inputs *State Numbers-->*	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
TreeShrink	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
secondary, canceled, or invalid command	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)
end of input	-(B)	0(B)	0(C)	0(B)	0(B)	0(D)	0(B)	0(E)	0(B)	0(C)

Pass 1 Output Code	*Function*
A	Add this input symbol to the FIFO stack.
B	Add this input symbol to the FIFO stack, generate one output symbol for each symbol currently on the stack, in FIFO order, then clear all symbols in the FIFO stack.
C	Generate the output symbol EditNode, and clear all symbols in the FIFO stack.
D	Generate the output symbol DeleteNode, and clear all symbols in the FIFO stack.
E	Generate the output symbol TidyWorkspace, and clear all symbols in the FIFO stack.

An extensive example of the output from Pass 1 is shown in appendix B.3 on page 185.

4.2.3.3 Pass 2: Segmenting and Characterizing Housekeeping Episodes

At this stage in the parsing process, the goal is to find the beginning and end of episodes, and to characterize each episode. Between consecutive constructive episodes are periods of housekeeping activities; intervening housekeeping episodes may be as simple as a six-second pause. Pass 2 determines each of the episodes of housekeeping activity; what is left are the constructive episodes, which are characterized by Pass 3.

Pass 2 reduces selected sequences of 16-tuples into a single 16-tuple. The output symbols represent delimiters between sequences of productive work. Pass 2 operates using a finite-state machine for the symbols in the housekeeping episodes. Upon reading each new housekeeping symbol, it looks up the next state, and goes to that state. Upon encountering a constructive-episode symbol, Pass 2 outputs a new symbol that characterizes the state it was in, but without characterizing where it has wandered through the state table.

Some special housekeeping symbols also cause a symbol to be output. These symbols are treated differently because they indicate a "meta-housekeeping" command by the user, not directly related to working with the housekeeping itself. The user has jumped out of the current working environment.

- time spent in external windows,
- a physical or mental break, signalled to the tracker,
- a long pause (more than 100 seconds),
- asking for help, or
- leaving a comment in the tracker.

Table 8 on page 74 classifies the 50 output symbols generated by Pass 1, indicating whether a particular command is parsed as part of a housekeeping episode, a meta-housekeeping episode, or a constructive episode. Pass 2 determines and characterizes these first two categories of episodes.

Table 8. Classification of Pass 1 output symbols, grouped by episodes

Housekeeping Commands (handled by Pass 2)	*Meta-housekeeping Commands (handled by Pass 2)*	*Constructive Commands (handled by Pass 3)*
Pause	LongPause	BreakLink
ChangeDefault	ClearDrawing	BreakAllLinks
ClipboardCopy	Comment	CopyNode
GoTo	HelpRequest	CreateNode
MainWindowReset	LeaveProseII	DeleteNode
MainWindowZoom	NewWorkspace	EditLabel
MapMove	OpenWorkspace	EditNode
MapSize	SystemIcon	LinkNodes
MapWindow		MoveNode
MapWindowRoam		Scramble
MapWindowZoom		
OutlineWindowMove		
OutlineWindowSize		
OutlineWindow		
SaveWorkspace		
SetDeleteMode		
SetTidyMode		
SystemMove		
SystemSize		
SystemWindow		
SystemZoom		
TidyWorkspace		
TreeGrow		
TreeShrink		

A grammar with type 3 power, a finite-state automaton, was used for Pass 2 to characterize the housekeeping episodes. I experimented with whether more power was needed for Pass 2. To correctly characterize a housekeeping episode, Pass 2 needs to say what was happening in the context of a string of symbols. Because it must make a distinct decision on every input symbol in every state, a finite-state machine generally does not handle singular events well. Perhaps the power of a context-free or context-sensitive grammar was required, I wondered. At one point, I generated all the symbol sequences that are reduced by Pass 2, and, surprisingly, the finite-state machine (FSM) turned out to be adequate. Each time I have re-examined this question, the Pass 2 FSM has turned out to be robust.

Pauses: Pauses are central to the operation of Pass 2. The fact that a writer pauses during writing does not tell us much about what mental processes were taking place during the pause. The writer may have been planning the next sentence, daydreaming, or answering the telephone. The tracker described here records pauses without knowledge of their nature, but uses them as separators between episodes of productive work.

One of the first research efforts I pursued was to categorize the different durations of pauses I saw in some of my test protocol records. Finding the right decision points in pause durations was a balancing act; the values were found through about 2 years of trial and error. A value too low resulted in many, short episodes, which gave insufficient data reduction. A value too high resulted in a few, composite episodes, which were too complex to provide much insight. I ultimately divided all possible pause durations into 4 groups for the sessions studied in this project: 0 to 6 seconds, 6 to 8.5 seconds, 8.5 to 100 seconds, and greater than 100 seconds. These were designated as "PauseType0" through "PauseType3," respectively.

PauseType0

A short pause. In this analysis, this type of pause is indicated by times less than 6.0 seconds. If a pause for 6.0 seconds or longer occurs between two consecutive constructive commands, they are considered in separate episodes.

Card, Moran, and Newell used 5.0 seconds in their *ICARUS* study of one session. In the early stages of the project, I observed a natural break in *Prose II* sessions at about

5.5 seconds. After about a year of parsing pilot protocol records, it appeared that the break was more naturally at 6.0 seconds.

PauseType1

A medium pause, frequently seen between constructive episodes. In this analysis, I implemented a medium pause as the narrow range from 6.0 seconds to 8.5 seconds. PauseType1 was used in this project to distinguish among the housekeeping episodes named "Cleanup," "Take Stock," and "Cleanup and Take Stock." It played no role in distinguishing constructive episodes. A pause of 6.0 seconds or greater always dictated a break between 2 constructive episodes, even if they are similar in class.

PauseType2

A long pause, indicating some deliberation or interruption between constructive episodes. In this analysis, I implemented this as the range from 8.5 to 100 seconds. PauseType2 was used in this grammar to determine participation in the housekeeping episodes that included "Take Stock." Pauses in this range were considered part of the process of taking stock by the human user; longer pauses were considered to be situations where a user had stepped out of the task in some way.

PauseType3

A very long pause, indicating the user has been interrupted, has pursued other tasks, or has stepped away from the system.. This was indicated by a pause of 100 seconds or greater. In their *ICARUS* study, Card, Moran, and Newell noted that the longest pause they observed (while the subject was still working on the assigned task) was 80 seconds. The threshold of 100 seconds was picked somewhat arbitrarily, but held up as a reasonable reference point throughout the project.

Pauses of this duration were characterized in their own episode, one whose time durations were so long that they were not included in many parts of the protocol analysis that focused on time durations.

The cap on the duration of PauseType0 was the value that most affected the output symbols generated by the parser. If this value is too short, the number of constructive episodes increases--with these episodes containing fewer commands, and occurring together in sequence. If this value is too long, the number of constructive episodes decreases, and they are characterized by more composite constructive episodes. The fixed values used in this parser are admittedly inflexible to individual differences, but proved robust enough to give useful results for the study.

In this project, the grammar handled the following situations as two different conditions:

- A long pause (*e.g.*, an hour)
- Exiting the software (closing the workspace) and entering it again in the same time period.

It could be argued that there are only a few keystrokes difference between these two kinds of situations. Perhaps a future preprocessor pass of the parser might review sets of protocol records from a subject and determine if they should be concatenated. Similarly, if a pause is significantly longer than 100 seconds (*e.g.*, a day), a single session might be divided into multiple sessions, because significantly different cognitive activities may have taken place between the times the subject used the system.

Newell's "Timescale of Human Actions" (Newell, 1988), reproduced in Table 9 on page 76, generally concurs with the different orders of magnitude for my framework of categorizing pause durations. The individual commands and pauses studied in this project fall within the "Cognitive Band"; sessions had durations within the "Rational Band." Episodes and phases were in the range of several seconds to several minutes, spanning these two bands. This concurrence is reasonable, since the grammar in this project was based upon earlier work of Card, Moran, and Newell, which segmented a task into similar phases and episodes.

Table 9. Timescale of Human Actions, from Newell (1988). This table is copied from figure 7-2, page 257.

Scale (in seconds)	*Time Units*	*System*	*World (theory)*
10^7	months		Social Band
10^6	weeks		
10^5	days		
10^4	hours	Task	Rational Band
10^3	10 minutes	Task	
10^2	minutes	Task	
10^1	10 seconds	Unit Task	Cognitive Band
10^0	1 second	Operations	
10^{-1}	100 ms	Deliberate Act	
10^{-2}	10 ms	Neural Net	Biological Band
10^{-3}	1 ms	Neuron	
10^{-4}	100 µs	Organelle	

Section 5.3.1.4 on page 98 discusses the actual results of the study, in terms of the distribution of the duration of the pauses in the 112 protocol records.

Pass 2 Output Symbols: Pass 2 generates eight different output symbols: five of these constitute housekeeping episodes; the other three constitute meta-housekeeping episodes. These symbols are shown in Table 10 on page 76. We chose the names for these output symbols to characterize, in a few words, the underlying sequence of activities. Card, Moran, and Newell did not name these types of sequences in their *ICARUS* study. For the set of input sequences we saw in our study, these eight symbols proved sufficient. However, we can conjecture odd sequences (not seen in this study) where additional symbols might be necessary.

Table 10. Pass 2 output symbols

Pass 2 output symbol name	*Description of the conditions that cause this output symbol to be generated*	*Internal Parser Symbol*
Housekeeping Episodes		
Cleanup	• Set Delete Mode on or off • Tidy trees • Close windows • Turn modes off	2002
Take Stock	• Look at a *Prose II* Outline Window • Save a workspace file • Make *Prose II* an Icon • Encounter a PauseType2 (8.5 to 100 seconds)	2004
Cleanup and Take Stock	• Cleanup plus Take Stock, in either order or interleaved	2003
Refocus	• Zoom or size using the map or main window • Search for a node (GoTo)	2005
Medium Pause	• Encounter PauseType1 (6 to 8.5 seconds) • Leave *Prose II* for another application in *Windows* for a brief period	2008
Meta-housekeeping Episodes		
Long Pause	• Encounter a PauseType3 (greater than 100 seconds) • Leave *Prose II* for another application in *Windows*	2008
Help Request	• View one or more help panels	2009
Tracker Comment	• Leave one or more comments in the protocol record	2010

Pass 2 Finite-State Machine: The following table describes the states defined for the Pass 2 finite-state machine.

Pass 2 State	*Description of the state*
0	Start state.
1	Long pause. A long pause was encountered, or the user left *Prose II* for another *Windows* application.
2	Cleanup. Set Delete Mode on or off, or tidy trees
3	Cleanup and take stock. Cleanup plus look at outline window, save file, icon
4	Short pause. This is a holding state, on the path to other states.
5	Refocus. Zoom or size using the map or main window, or search for a node.
6	Help request. Click on one of the help buttons.
7	Cleanup. We get here by setting Delete Mode or Tidy Mode.
8	Take stock. Look at outline window, save file, icon.
9	Start over. Open a workspace file, clear drawing, or start a new workspace.
10	Tracker comment. Leave a comment in the tracker.

Table 11. Pass 2 finite-state machine

State Names--->	*Start State*	*Long Pause*	*Clean-up*	*Clean-up & Take Stock*	*Short Pause*	*Re-focus*	*Help Re-quest*	*Clean-up*	*Take Stock*	*Start Over*	*Track-er Com-ment*
Inputs State Numbers-->	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
PauseType0	4	-	-	-	-	-	-	-	-	-	-
PauseType1	1	-	-	-	1	-	-	-	-	-	-
PauseType2	1	-	3	-	1	-	-	3	-	-	-
PauseType3	1	-	0(A)	0(A)	0(A)	0(A)	-	0(A)	0(A)	0(A)	-
ChangeDefault	5	5	-	-	5	-	0(A)	-	-	-	0(A)
ClipboardCopy	8	8	3	-	8	-	0(A)	3	-	-	0(A)
GoTo	5	5	5	0(A)	5	-	0(A)	5	5	0(A)	0(A)
HelpRequest	6	6	0(A)	0(A)	6	0(A)	-	0(A)	0(A)	0(A)	0(A)
LoseFocusLong	1	-	0(A)	0(A)	1	0(A)	0(A)	0(A)	0(A)	0(A)	0(A)
LoseFocusShort	4	-	-	-	-	-	-	-	-	-	-
MainWindowReset	5	5	-	-	5	-	-	-	-	-	0(A)
MainWindowZoom	5	5	5	5	5	-	0(A)	5	5	-	0(A)
MapMove	5	5	-	-	5	-	0(A)	-	-	-	0(A)
MapSize	5	5	-	-	5	-	0(A)	-	-	-	0(A)
MapWindowClose	2	2	-	-	2	-	0(A)	-	-	-	0(A)
MapWindowOpen	5	5	5	-	5	-	0(A)	5	-	-	0(A)
MapWindowRoam	5	5	5	5	5	-	0(A)	5	5	-	0(A)
MapWindowZoom	5	5	5	5	5	-	0(A)	5	5 -	-	0(A)
MentalPause	1	-	0(A)	0(A)	1	0(A)	0(A)	0(A)	0(A)	0(A)	-
OutlineWindowOpen	8	8	-	-	8	-	0(A)	3	-	-	0(A)
OutlineWindowClose	2	2	-	-	2	-	0(A)	-	-	-	0(A)
OutlineWindowMove	8	8	3	-	8	-	0(A)	-	-	-	0(A)
OutlineWindowSize	8	8	3	-	8	-	0(A)	-	-	-	0(A)
SaveWorkspace	8	8	3	-	8	-	0(A)	3	-	-	0(A)
SetDeleteModeOn	7	7	7	-	7	7	0(A)	-	3	0(A)	0(A)
SetDeleteModeOff	2	2	-	-	2	-	-	-	-	-	0(A)
SetTidyModeOn	7	7	7	-	7	-	0(A)	-	3	0(A)	0(A)
SetTidyModeOff	2	2	-	-	2	-	-	-	-	-	0(A)
SystemIcon	1	-	0(A)	0(A)	0(A)	0(A)	0(A)	0(A)	0(A)	0(A)	0(A)
SystemMove	5	5	5	-	5	-	-	3	-	-	0(A)
SystemSize	5	5	5	-	5	-	-	3	-	-	0(A)
SystemWindowOpen	9	0(A)	0(A)	0(A)	0(A)	0(A)	0(A)	0(A)	0(A)	-	0(A)
SystemWindowClose	9	0(A)	0(A)	0(A)	0(A)	0(A)	0(A)	0(A)	0(A)	-	0(A)
SystemZoom	5	5	-	-	5	-	0(A)	-	-	-	0(A)
TidyWorkspace	7	7	7	-	7	-	0(A)	-	3	-	0(A)
TrackerComment	10	0(A)	0(A)	0(A)	0(A)	0(A)	0(A)	0(A)	0(A)	0(A)	-
TreeGrow	8	8	3	-	8	-	0(A)	8	-	-	0(A)
TreeShrink	8	8	3	-	8	-	0(A)	8	-	-	0(A)
Unproductive	1	-	-	-	0(A)	-	-	-	-	-	-
end of input	-	0(A)	0(A)	0(A)	0(A)	0(A)	0(A)	0(A)	0(A)	0(A)	0(A)

Pass 2 Output Code	*Function*
A	Generate an output symbol corresponding to the current state, go to state 0, then handle the current input symbol.

An extensive example of the output from Pass 2 is shown in appendix B.4 on page 191.

4.2.3.4 Pass 3: Characterizing Constructive Episodes

The design assumption for Pass 3 is that a human is pursuing purposeful work during each of the periods labeled as a constructive episode. This work is carried out in a sequence of small actions with little elapsed time between them. Several goals may guide a user's actions during a single constructive episode: for example, a user might move some nodes, delete some nodes that are no longer needed, and then create, label and link a new set of nodes.

The Pass 3 output symbols characterize a constructive episode. The input to Pass 3 are the grammar symbols that were output by Pass 0 and Pass 1, separated in the input string by the new symbols written by Pass 2. Pass 3 reduces selected sequences of 16-tuples into one or more n-tuples. Passes 0, 1, and 2 each generated 16-tuple records, as illustrated in Section 4.2.3.1 on page 69. Pass 3, however, can generate tuples with unlimited size, since a single output symbol might need to contain a list of all the nodes in a newly-created tree. Output symbols generated by Pass 2 exit from Pass 3 unscathed; the resulting output string is thus an intermixing of Pass 2 and Pass 3 output symbols.

The lexical analysis of Pass 3 is simple; the syntactic analysis is complex. In its lexical analysis, Pass 3 reads input symbols, where each input symbol is a 16-tuple in the same format as the output symbols generated by Passes 0, 1, and 2. Thus, the lexical analysis should never fail unless an intermediate file, containing the output symbols, is tampered with.

The syntactic analysis for Pass 3 understands two characteristics of the node currently being operated upon: the node's structural place in the current workspace and whether it was created within the current constructive episode. The syntactic analysis is implemented using:

- A set of counters, reset whenever a new constructive episode starts. These counters record such information as the number of nodes created so far in this constructive episode, the number of existing nodes that had their labels changed, and the number of invalid commands attempted during this constructive episode. A complete description of these counters is shown in Table 13 on page 80.

- A set of lists of nodes, also reset whenever a new constructive episode starts. These lists record the information that is later written as elements of the output symbols. For example, one of the output symbols might say that 10 new nodes were created. The IDs of these 10 nodes are listed as parameters of the output symbol. Thus, a list of the newly-created nodes during the constructive episode is maintained, to be used when Pass 3 generates the output symbol. A complete description of these lists is shown in Table 14 on page 81.

- A history list of the commands done so far in this constructive episode, to determine whether later commands in this episode act upon new or old nodes, new or old trees, and so on.

- The complete underlying node/forest structure. For example, each time a pair of nodes are linked, they are linked in the internal data structure kept by Pass 3. This lets Pass 3 query the internal structure during later analysis—for example, "Is this node that is being deleted a part of a tree?"

  In this internal data structure, Pass 3 captures the information supplied by both primary and secondary commands.

For example, linking two nodes may cause an existing link to be broken. This appears as a primary command, LinkNodes, followed by a secondary command, BreakLink. Pass 3 executes both actions upon its internal data structure.

The decision of what output symbols for Pass 3 to generate raised a group of questions with the following flavor: "Did such-and-such occur in this constructive episode?" There is little concept of sequencing among the events in an episode—rather, it is simply, "did X and Y occur during this time period?" To correctly characterize each constructive episode, many elements had to be stored by the grammar during each episode.

This allows more than one goal to be occurring concurrently, but says little about them qualitatively—such as, how long they lasted, how much they overlapped, or the percentage of time spent on each goal. As with Pass 2, I generated all the symbol sequences that are reduced by Pass 3, and the approach I have taken turned out to be adequate. Each time I have re-examined this question, Pass 3 has turned out to be robust.

Pass 3 Output Symbols: Pass 3 generates twelve different output symbols. Most of these symbols carry an additional list of parameters; for example, the output symbol "Created n solo nodes" carries a list of the solo nodes created in this constructive episode, along with a count (n) of these nodes.

Table 12 on page 79 describes each of these output symbols.

Table 12. Pass 3 output symbols

Pass 3 output symbol	*Description of the conditions that cause this output symbol to be generated*	*Internal Parser Symbol*
Created n solo nodes	Created new nodes, but did not link them to other nodes. The node's labels and contents may have been edited.	3001
Created n new trees	Created a new tree, using new nodes, or copied a tree, creating a complete tree with new nodes.	3002
Grew n existing trees	Linked new nodes to one or more existing trees.	3003
Assembled n trees	Took existing nodes and linked them together into a tree. The root of the tree may be newly-created in this episode.	3004
Broke existing links	Broke the link between nodes that had been created and linked in previous episodes.	3005
Deleted n nodes	Deleted nodes that had been created in a previous episode.	3006
Edited existing nodes	Edited the label and/or file contents of nodes created in a previous episode. This is the first time these nodes had been edited.	3007
Revised existing nodes	Edited the label and/or file contents of nodes that had been previously edited.	3008
Moved existing nodes	Moved nodes created in a previous episode.	3009
Hooked existing nodes to n trees	Linked existing nodes to one or more trees.	3010
Unproductive work	Canceled one of more commands, or attempted an invalid command.	3011
Start over	Started over with a fresh workspace. The commands that correspond with this either open a workspace file, clear the drawing, or start a new workspace. Any nodes already in the workspace when the command was executed are deleted.	3012

The episodes titled "Edited existing nodes" and "Revised existing nodes" both comprise editing of a node's label or its associated file. While these might be considered different types of activities, *Prose II* encouraged an easy flow between editing of a node's label and its contents. The additional creation of 2 more constructive episodes to draw a distinction was tried, but did not show much value in the results of the study.

Pass 3 Counters and Lists: The counters and lists maintained by Pass 3 were dictated by the output symbols. For example, to see if any new trees have been created in a constructive episode, each time a new tree is created, a counter for "new trees created in this episode" is incremented. Each time the count is incremented, the identifier of the node at the root of the new tree is added to the list of new trees. When the output symbol "Created n new trees" is generated, the corresponding counter is used to fill in the value for n, and the list is used to generate the parameters for this symbol, which are an enumeration of the names of the roots of the new trees.

Table 13 on page 80 describes the counters accumulated by Pass 3 during each constructive episode. At the end of a constructive episode, Pass 3 uses the values in these counters to determine which output symbols to generate. The counters are all reset to zero at the end of each constructive episode; nothing is carried between episodes, except the ongoing structure of nodes and trees.

Table 13 (Page 1 of 2). Pass 3 counters, listed alphabetically

Pass 3 counter name	*Description of the Counter*
BreakNewLinksCounter	Counts how often a link was broken, where the link was made in this constructive episode.
BreakOldLinksCounter	Counts how often a link was broken, where the link was made in a previous constructive episode.
CancelledCommandsCounter	Counts how often a command was canceled in this constructive episode.
HookInNodesCounter	Counts how many pairs of nodes were linked together, where both nodes were created in a previous constructive episodes, and the parent node is already a member of a tree. Literally, the child node is hooked into an existing tree.
InvalidCommandsCounter	Counts how often an invalid command was attempted in this constructive episode.
NewLabelsEditedCounter	Counts how often a node, newly created in this constructive episode, was labeled for the first time.
NewLabelsRevisedCounter	Counts how often a node, newly created in this constructive episode, was labeled more than once.
NewNodesCopiedCounter	Counts how often a node, newly created in this constructive episode, was copied.
NewNodesDeletedCounter	Counts how often a node, newly created in this constructive episode, was deleted.
NewNodesEditedCounter	Counts how often a node, newly created in this constructive episode, was edited for the first time using an editing program.
NewNodesMovedCounter	Counts how often a node, newly created in this constructive episode, was moved.
NewNodesRevisedCounter	Counts how often a node, newly created in this constructive episode, was edited more than once using an editing program.
NewRootsCreatedCounter	Counts how often a pair of nodes are linked, where both nodes were solo nodes before this linking operation.
NewTreesAssembledCounter	Counts how often a new tree was assembled from parts. This is indicated by one of two situations: 1) the parent node is a solo node and the child is the member of a tree—and at least one of these was created in a previous episode, or 2) the parent tree is new and the child node was created in a previous episode—and at least one of these is a member of a tree.
NodesCreatedCounter	Counts how often a CreateNode command is successfully executed in this constructive episode.
OldLabelsEditedCounter	Counts how often a node, created in a previous constructive episode, was labeled for the first time.
OldLabelsRevisedCounter	Counts how often a node, created in a previous constructive episode, was labeled more than once.
OldNodesCopiedCounter	Counts how often a node, created in a previous constructive episode, was copied.
OldNodesDeletedCounter	Counts how often a node, created in a previous constructive episode, was deleted.

Table 13 (Page 1 of 2). Pass 3 counters, listed alphabetically

Pass 3 counter name	*Description of the Counter*
OldNodesEditedCounter	Counts how often a node, created in a previous constructive episode, was edited for the first time using an editing program.
OldNodesMovedCounter	Counts how often a node, created in a previous constructive episode, was moved.
OldNodesRevisedCounter	Counts how often a node, created in a previous constructive episode, was edited more than once using an editing program.
OldTreesGrownCounter	Counts how often a new node is linked as a child into an existing tree
SoloNodesCreatedCounter	Counts how often new solo nodes are created in this constructive episode. This can occur because of CreateNode command, or because an existing link between a pair of node is broken.
StartOverCounter	Counts how often the workspace was started afresh in this constructive episode.

Table 14 on page 81 describes the singly-linked lists that Pass 3 constructs as part of characterizing each constructive episode. After it sees the last symbol of a constructive episode, Pass 3 uses these lists of nodes and trees to determine the parameters for some of its output symbols. These five empty lists are created when Pass 3 is first called. The contents of a list have a lifetime of one constructive episode; the lists are all flushed at the end of each constructive episode.

Table 14. Pass 3 lists, listed alphabetically

Pass 3 list name	*Description of the list*
HookInNodesList	Lists the nodes that have been hooked into existing trees. If the newly-linked node is the root of a tree, it is added to this list as a new root. Otherwise, it is added to the list as the descendant of a node which is already on the list.
NewRootsCreatedList	Lists the nodes that are new roots of trees. These nodes became the new roots during this constructive episode.
NewTreesAssembledList	Lists the trees that have been assembled in this constructive episode. If the newly-linked node is the root of a tree, it is added to this list as a new root. Otherwise, it is added to this list as the descendant of a node which is already on the list.
OldNodesDeletedList	Lists the nodes, deleted in this constructive episode, that were created during previous constructive episodes.
OldTreesGrownList	Lists the existing trees that have been added to during the current constructive episode.

Pass 3 Logic: At the end of a constructive episode, Pass 3 generates one or more of the output symbols listed in Table 12 on page 79. It makes this decision based on the values of the counters. Table 16 on page 82 provides the logic used in generating these output symbols. This table is given as an alternative notation for what are essentially production rules. The evaluation order of the rules is from top to bottom as shown in the table. For each true condition on the righthand side, one of the symbols on the lefthand side is generated. The symbols in the righthand side of each table entry should be interpreted as being ANDed together.

The following is an example of an entry in Table 16 on page 82.

Table 15. Pass 3 production rule example

Pass 3 output symbol	*Logic*
Created n new trees	NewRootsCreatedCounter > 0 NodesCreatedCounter > 0 SoloNodesCreatedCounter = 0

In more traditional, production rule notation, this logic might alternatively be shown as:

```
If      the NewRootsCreatedCounter is greater than 0, and
        the NodesCreatedCounter is greater than 0, and
        the SoloNodesCreatedCounter is 0,
then    generate the output symbol "Created N new trees,"
        where N is the value of the NewRootsCreatedCounter.
```

No more than one instance of any output symbol is generated in a constructive episode. The first of the output symbols generated serves as the primary constructive episode symbols. Later symbols serve as secondary constructive episode symbols. This distinction between primary and secondary constructive episodes is used in Pass 4 when characterizing phases of constructive.

Table 16 (Page 1 of 2). Pass 3 output symbols and how they are generated

Pass 3 output symbol	***What combination of counter values causes this output symbol to be generated?***
Start over	StartOverCounter > 0
Created n new trees	NewRootsCreatedCounter > 0 NodesCreatedCounter > 0 SoloNodesCreatedCounter = 0
Created n new trees	NewRootsCreatedCounter > 0 NodesCreatedCounter = 0
Created n new trees	NewRootsCreatedCounter > 0 NodesCreatedCounter > 0 SoloNodesCreatedCounter > 0
Grew n existing trees	OldTreesGrownCounter > 0 NodesCreatedCounter > 0 SoloNodesCreatedCounter = 0

Grew n existing trees	OldTreesGrownCounter > 0
	NewRootsCreatedCounter = 0
	NewTreesAssembledCounter = 0

Assembled n trees	NewTreesAssembledCounter > 0 NodesCreatedCounter > 0 SoloNodesCreatedCounter = 0
Assembled n trees	NewTreesAssembledCounter > 0 NodesCreatedCounter = 0
Hooked existing nodes to n trees	HookInNodesCounter > 0

Created n solo nodes	SoloNodesCreatedCounter > 0
	SoloNodesCreatedCounter = NodesCreatedCounter
	OldTreesGrownCounter = 0
	NewTreesAssembledCounter = 0
	NewRootsCreatedCounter = 0

Created n solo nodes	NewRootsCreatedCounter > 0 NodesCreatedCounter > 0 SoloNodesCreatedCounter > 0
Broke existing links	BreakOldLinksCounter > 0 HookInNodesCounter = 0
Deleted n nodes	StartOverCounter = 0 OldNodesDeletedCounter > 0
Edited existing nodes	OldLabelsEditedCounter > 0
Edited existing nodes	OldNodesEditedCounter > 0
Revised existing nodes	OldLabelsRevisedCounter > 0
Revised existing nodes	OldNodesRevisedCounter > 0

Table 16 (Page 1 of 2). Pass 3 output symbols and how they are generated

Pass 3 output symbol	*What combination of counter values causes this output symbol to be generated?*

Moved existing nodes	OldNodesMovedCounter > 0
	OldTreesGrownCounter = 0
	NewRootsCreatedCounter = 0
	NewTreesAssembledCounter = 0
	HookInNodesCounter = 0
	OldLabelsEditedCounter = 0
	OldNodesEditedCounter = 0
	SoloNodesCreatedCounter = 0 or
	no solo nodes left remaining

Unproductive work	no solo nodes remaining (nodes created then deleted)
Unproductive work	no output symbol so far CancelledCommandsCounter > 0
Unproductive work	no output symbol so far InvalidCommandsCounter > 0

An extensive example of the output from Pass 3 is shown in appendix B.5 on page 195.

4.2.3.5 Pass 4: Phases of Activity

Pass 4 characterizes sequences of similar activity among the constructive episodes. Its input is the grammar symbols generated by Pass 2 (for the housekeeping and meta-housekeeping episodes) and Pass 3 (for the constructive episodes). Pass 2 and Pass 3 output symbols are interleaved in the output sequence of Pass 3; Pass 2 output symbols move unscathed through Pass 3. As with previous passes, Pass 4 reduces selected sequences of n-tuples into a single n-tuple. The size of the input n-tuples varies, depending upon the information generated by Pass 3; for example, a single n-tuple may contain the number of nodes created in an episode and the identifier numbers of each of the nodes.

The output symbols represent phases of work activity. Pass 4 operates using a finite-state machine for the input symbols. Upon reading the first input symbol in a phase, it moves to the indicated non-zero state. Whenever an input symbol causes the state to change, the current output symbol is generated and the state of Pass 4 is reset. Otherwise, no state change is made, and the next input symbol is read.

Since Pass 3 can generate multiple symbols for a single constructive episode, the Pass 4 finite-state machine is divided into two parts. The first part is used to handle the first of multiple symbols; the second part handles any subsequent symbols for the same constructive episode. The decision on what output symbol to generate is heavily biased by the first of the Pass 3 symbols, since Pass 3 can generate multiple symbols to characterize each constructive episode.

Pass 4 Output Symbols: Pass 4 generates seven different output symbols; these are the highest level symbols generated by the grammar described here. These symbols are shown in Table 17 on page 84.

Table 17. Pass 4 output symbols

Pass 4 output symbol	*Description of the conditions that cause this output symbol to be generated*	*Internal Parser Symbol*
Exploration	Created solo nodes. Edited new or existing nodes.	4001
Define Hierarchies	Created new trees.	4002
Top Down Construction	Grew old trees or hooked nodes to them. In top-down construction, the root of a tree and its children are created and labeled early, with the rest of the document construction consisting of creating new nodes and adding them to the leaves of the evolving tree.	4003
Bottom Up Construction	Assembled trees from existing nodes. The root of the tree may be a newly-created node. In bottom-up construction, nodes are created and labeled before they are linked; small trees are linked into larger trees.	4004
Tree Structure Revision	Broke old links, deleted old nodes, moved old nodes	4005
Document Revision	Revised the label and contents of nodes that had been created or edited in previous episodes.	4006
New Workspace	Started over: opened files or cleared the workspace.	4007

Pass 4 Finite-State Machine: The following table describes the states defined for the Pass 4 finite-state machine.

Pass 4 State	*Description of the state*
0	Start state
1	Exploration: create solo nodes
2	Define hierarchy
3	Top-down construction
4	Bottom-up construction
5	Revise structure: break old links, delete nodes
6	Revise document: edit and revise node labels and contents
7	New workspace

In the following state table, the housekeeping episodes are listed first, followed by the primary constructive episodes, and concluding with the secondary constructive episodes.

Table 18 (Page 1 of 2). Pass 4 finite-state machine

State Names---->	*Start State*	*Exploration*	*Define hierarchy*	*Top-down Cons.*	*Bottom-up Cons.*	*Revise Struc*	*Revise Doc*	*New workspace*
Inputs *State Numbers-->*	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
START_OVER	7	7(A)	7(A)	7(A)	7(A)	7(A)	7(A)	-
LONG_PAUSE	-	-	-	-	-	-	-	-
CLEANUP	-	-	-	-	-	-	-	-
CLEANUP_TAKE_STOCK	-	-	-	-	-	-	-	-
TAKE_STOCK	-	-	-	-	-	-	-	-
REFOCUS	-	-	-	-	-	-	-	-
HELP_REQUEST	-	-	-	-	-	-	-	-
CREATE_SOLO_NODES	1	-	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)
CREATE_NEW_TREES	2	2(A)	-	2(A)	2(A)	2(A)	2(A)	2(A)
GROW_OLD_TREES	3	3(A)	-	-	3(A)	3(A)	3(A)	3(A)
ASSEMBLE_TREES	4	4(A)	4(A)	4(A)	-	-	4(A)	4(A)
BREAK_OLD_LINKS	5	-	-	5(A)	5(A)	-	-	5(A)
DELETE_OLD_NODES	5	-	-	-	-	-	-	5(A)
EDIT_OLD_NODES	6	-	-	-	-	6(A)	-	6(A)
REVISE_OLD_NODES	6	-	-	6(A)	-	6(A)	-	6(A)
MOVE_OLD_NODES	5	-	-	-	-	-	-	5(A)

Table 18 (Page 1 of 2). Pass 4 finite-state machine

State Names----->	*Start State*	*Ex-ploration*	*De-fine hier-archy*	*Top-down Cons.*	*Bot-tom-up Cons.*	*Re-vise Struc*	*Re-vise Doc*	*New work-space*
Inputs *State Numbers-->*	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
HOOK_IN_NODES	3	3(A)	3(A)	-	3(A)	3(A)	3(A)	3(A)
UNPRODUCTIVE_WORK	-	-	-	-	-	-	-	-
CREATE_SOLO_NODES (secondary)	1	-	-	-	1	1	1	1
CREATE_NEW_TREES (secondary)	2	2	-	2	2	2	2	2
GROW_OLD_TREES (secondary)	3	3	-	-	3	3	3	3
ASSEMBLE_TREES (secondary)	4	4	-	4	-	-	4	4
BREAK_OLD_LINKS (secondary)	5	-	-	-	-	-	-	5
DELETE_OLD_NODES (secondary)	5	-	-	-	-	-	-	5
EDIT_OLD_NODES (secondary)	6	-	-	-	-	6	-	6
REVISE_OLD_NODES (secondary)	6	-	-	-	-	6	-	6
MOVE_OLD_NODES (secondary)	-	-	-	-	-	-	-	1
HOOK_IN_NODES (secondary)	3	3	-	-	-	3	3	3
UNPRODUCTIVE_WORK (secondary)	-	-	-	-	-	-	-	-

Pass 4 Output Code	*Function*
A	Generate an output symbol corresponding to the current state, then go to the new state.

An extensive example of the output from Pass 4 is shown in appendix B.6 on page 196.

4.2.3.6 Grammar Summary

The grammar has been described in progressive levels of detail. First, the levels in the abstract model of user interaction that correspond with levels in the parse tree were introduced: session, phases, episodes, and commands. Next, the grammar rules for each of these levels were introduced; key categorizations in the grammar occur where sequences of commands are identified as episodes, and where sequences of episodes are characterized as phases. Finally, the particular state machines that implement each level were shown. Further implementation details of the parser are described in the following subsection. These are included because of the methodological emphasis of this project: what difficulties were encountered in implementing such a parser?

4.2.4 Further Implementation Details of the Parser

Aho and Ullman (1977, p. 146) define a parser for a grammar *G* as "a program that takes as input a string *w* and produces as output either a parse tree for *w*, if *w* is a sentence of *G*, or an error message indicating that *w* is not a sentence of *G*."

In all cases except one, the input strings (that is, the protocol records) in this study were valid, that is, they were recognized by the parser without error.[11] So, more important for this study is a parser's ability to create parse trees.

The parser used six standalone programs to convert a protocol record into a parse tree and summary file. Operating system pipes and filters were used to connect the six passes of the parser. "The pipes and filters technology packages re-usable code into small, standalone pro-

[11] In the single exception, a subject used a text editor to append a paragraph to the bottom of a protocol record. The parser stopped and indicated an error when it tried to read the beginning of this block of text.

grams, in which each program is a tool (a filter) that does a single job well. The filters can be connected by pipes, a communication channel through which the output of one filter can be received as the input of the next." (Cox, 1986)

All but the last pass of the parser read their input from the byte-stream known as standard input (**stdin**) and write to standard output (**stdout**). **stdin** and **stdout** are generic files that are easily bound to a filename by the operating system. For the first pass, an actual protocol record is supplied as the standard input. By using the operating systems' redirection capability, the output of any parser pass can be sent to a single file or concatenated to a larger file. For example, by running the parser (with the necessary command line parameters) against all the protocol records, the list of all the commands issued in all the sessions can easily be written in a single file.

To construct a parse tree at the end of the parsing process, the final pass opens and reads from the intermediate files produced in the preceding passes. If these intermediate passes are not otherwise required, the passes can be piped together in such a way that a given pass is reading directly from the output of its preceding pass.

Reading from **stdin** and writing to **stdout** have another advantage. Although it was not done this way in this study, this parser could be used to parse protocol records in real time. The parser accommodates one of the facts of human behavior: at any point the telephone might ring and a session would be abruptly ended. Thus, at all times during the parsing, the state of each pass is known. A future enhancement to the testbed system can be conceived where a user could see the current parse tree for a session at all times during the session. The tracker itself could write its output to **stdout**, instead of to the unique file that it creates, and that output could then be piped into the parser. The memory constraints and operating system restrictions of DOS and *Windows* made such an arrangement impractical for the current system.

Using pipes and filters is efficient in programmer time, but because of their buffering characteristics, they are not always efficient in terms of computer performance. For this study, this was a reasonable trade-off. Performance of the parser was not a consideration (unless it had taken days or hours). Another disadvantage of using pipes and filters is that they pass bytes, not the highly-structured data that is frequently seen in complex programs. You cannot pass a pointer or a linked list across a pipe without great difficulty. Again, this did not pose a problem here.

The parser writes its intermediate output files in a generic spreadsheet format—the Lotus spreadsheet ASCII format—which has become a standard among personal computer programs. This makes it easy to run a spreadsheet, database, word processing, or statistical analysis program against such an output file at any intermediate pass of the parsing process. Further, the entire parsing process can be driven by a single *Make* input file, making it easy to run a consistent analysis against any selected group of sessions. This is discussed further in Section 4.2.4.4 on page 88.

The parser is designed to operate on a protocol record for a single session. One of my first lessons from this study is that a document is frequently not written in one sitting. Many subjects submitted a <u>group</u> of protocol records that defined their work on a single document. I considered modifying the parser to handle this entire group as a single piece, rather than as a group of individual files. The individual files define sessions; the conglomerate defines extended work on a document over several days. I decided this was a problem I did not understand well, and should be a topic for future study. Using the parser and its command line parameters as described here allowed looking at a number of the features of the sessions for a single document; in addition, sessions could be concatenated by hand—but, this solution would not be tenable for thousands of sessions.[12] The parser described here operates on one protocol record at a time.

[12] Some of the ambiguities that can occur when concatenating sessions, as well as ways to resolve them, are discussed as possible future enhancements in Section 6.5 on page 140.

The conversion of the grammar ATN into a sequence of connected parser passes made the implementation task manageable. Individual passes could be reworked without compromising other portions of the parser. Most of the passes were simple to implement, with the state machines as large compiled arrays. Some of the key implementation decisions made in Passes 0, 2, and the summary pass are highlighted below.

4.2.4.1 Pass 0 Details

For Pass 0, the UNIX tool LEX (Lesk, 1975) was used to generate the lexical analyzer, and YACC (Johnson, 1975) was used to generate the syntactic analyzer. The lexical analyzer generated for Pass 0 reads the stream of bytes which comprise a protocol record file. The lexical analyzer is designed to recognize the format of the header lines, the timestamps, the English keyword for each command, and the parameter sequences. The syntactic analyzer for Pass 0 recognizes the sequences of these on a line; for example, the YACC input is designed to recognize a well-formed CreateNode command as having start, stop, and elapsed times, the keyword CreateNode, and parameters consisting of a node ID and starting coordinates (or keywords like "`-- invalid --`"). For secondary commands, the syntactic analyzer additionally constructs distinctive values for the timestamps.

The individual records produced by the tracker were designed in conjunction with the Pass 0 lexical and syntactic analyzers. One concern was producing unambiguous information in the protocol record; for example, the parser should not be confused if a command keyword is included in the label for a node.

```
39:59.91  40:00.07    0.16  OpenWorkspace       File(`IBMSC.SCR`) Format(`.PR2`)
```

I solved this by delimiting strings in the protocol record by the symbols (` and `). To avoid any ambiguity, the *Prose II* tracker converts any grave quote (`) character in a user's string to an acute quote (´). As an additional piece of pre-processing, the tracker converts any alphanumeric characters in the node labels or search strings to the letter `x`, for confidentiality.

All subjects worldwide spoke English, at least as a second language, so none of them commented about the fact that the entire *Prose II* system uses English phrases and keywords. A problem with international character sets did occur, however. The version of LEX used to generate the scanner dealt only with the seven-bit ASCII characters represented numerically by 0 to 127. I did not anticipate that characters outside this range would be entered in node labels or search strings, since *Windows* filters these symbols as part of its operation. But, I was using a US English version of *Windows*! Subjects in Finland and Greece submitted protocol records that contained characters that were not `x`'d out by *Prose II* and that were in the ASCII numeric range of 128 to 255. The lexical analyzer of Pass 0 terminated when it tried to read these characters. I had to modify these characters by hand in order for them to be parsed; fortunately, this occurred in only 4 of the 112 protocol records.

4.2.4.2 Pass 2 Details

I first implemented the syntactic analysis portion of Pass 2 as a simple state look-up table, with a custom lexical analyzer (similar to the lexical analyzer portion of Pass 1). The table was large and not sparse, making it hard to maintain. After several months, it became difficult for me to comprehend the paths through the FSM. I later converted to a syntactic analyzer generated by YACC. The YACC description, with its BNF-like syntax, made the operation of this pass easier to comprehend. Working with the YACC source, I made several changes in the FSM for correctness and uniformity–these were paths through the FSM that had been difficult to see before.

Fortunately, no users demonstrated truly strange or bizarre behavior among the protocol records studied here. For example, there was no clear-cut playing with the system, such as going back and forth between commands repeatedly, or creating nodes to fill the entire screen.

4.2.4.3 Producing the Parse Tree and Summary Information

A final pass of the parser does not read from a standard input stream. Instead it uses the output from each of the previous passes to construct the parse tree and summary information about the parse.

Its operation consists of opening the original protocol record (from which it reads the header information), and the intermediate output files of Passes 0, 1, 3, and 4. (The output information from Pass 2 is entirely contained in the Pass 3 output.) Starting with the Pass 4 output, it reads the first line of information, which is the first n-tuple generated by Pass 4. This line of information includes a timestamp with its start and stop time. It then reads lines from the Pass 3 file until one of the starting times exceeds the stop time of the current Pass 4 line. Similarly, for each Pass 3 record, it recursively reads Pass 1 records, and for each Pass 1 record it recursively read Pass 0 records. An example parse tree diagram is shown in Figure 23 on page 89.

The parse tree is written to a file in text representation in one of the formats described in Section 3.1.10.3 on page 43. Since these are all tree formats, *Prose II* itself can be used to view and manipulate the resulting parse tree. The large size of some of the parse trees exceeded the original memory capabilities of *Prose II*, and led me to a re-write of the memory management routines in *Prose II* to allow thousands of nodes to be active in memory and viewed in a single window.

This summary pass maintains a set of counters during the process of summarizing the parse for one session. At the end of this pass, this counter information is written to **stdout**. This information provides a summary of what happened during the parse. An example of the summary output is shown in Figure 24 on page 90.

Using these counters, many variables can be examined and compared in aggregate across the entire sample of subjects--or by selected groups. For values that vary within a task or among subjects (such as the time spent on the task) standard measures such as range, mean, and variance are easily computed.

An extensive example of the output from this summary pass is shown in appendix B.7 on page 197.

4.2.4.4 Controlling the Parser Operation

I developed the parser a pass at a time. A "run" of the parser was controlled with a DOS batch file, which handled the input to each pass and the sequencing of passes. A DOS batch file was adequate for parsing a single input file, but it lacks flexibility. It had no knowledge of whether the parse was already complete and it was difficult to use it to parse larger groups of protocol records. I later switched to using the *Make* utility program to control the parsing.

Make is a program shipped with many current operating systems and compilers. *Make* operates by examining the temporal relationship among a set of files. A given file is said to have one or more files that it is dependent on; if the dependent files are younger than the given file, *Make* runs the prescribed programs, which presumably update the given file. For this parser, the given file is frequently the parse tree file and the dependent file is the original protocol record file. When *Make* is run, the protocol record is parsed if a parse tree file currently exists, or if the protocol record file is younger than the current parse tree file.

Make examines these dependencies for groups of files and runs the same programs against these files, using wildcard characters to substitute the changing file names. Further, *Make* stops its operation if one of the programs it is running ends with an unexpected ERRORLEVEL value. This allows me to run *Make* against a large group of files while the parser was still being developed; if the parser failed for any reason, *Make* also stopped its operation.

Figure 25 on page 91 shows the *Make* input file (called a "Makefile") used for running a complete parse, in this case, against the list of non-trivial sessions. An abbreviated Makefile, shown in

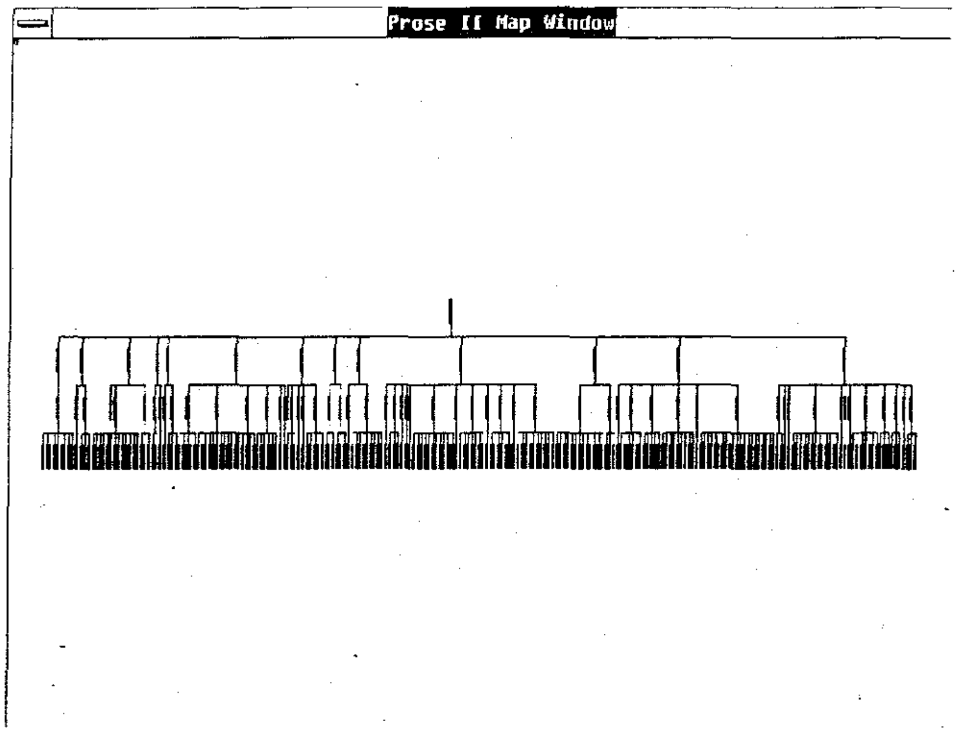


Figure 23. Example parse tree drawing, for session S16R0102. In the first row is the non-terminal symbol representing the session. The second row indicates the non-terminal symbols for the 13 phases of the session. The third row indicates the non-terminal symbols for 46 different episodes. On the bottom row are terminal symbols for the individual commands and pauses.

Figure 26 on page 91, illustrates how to generate a specific set of Pass 3 output, without the creation of any intermediate files.

```
"S16R0102"
    302, Number of nodes in the parse tree
    118, Number of commands

     10, Number of creates
      4, Number of deletes
      0, Number of copies
     12, Number of links
      0, Number of break links
      3, Number of moves
      7, Number of tidies
      8, Number of canceled operations

      0, Number of opens
      3, Number of saves
      6, Number of nodes in last saved tree
      0, Number of nodes with no offspring
      5, Maximum depth of saved tree
     -1, Cumulative X vector of creates
      0, Cumulative Y vector of creates
   0.00, Stage Index

      0, Number of edit node operations
      4, Number of help requests
      2, Number of long pauses
      0, Number of user comments
      0, Number of times subject left Prose II
     25, Number of constructive episodes
     21, Number of housekeeping episodes
     13, Number of phases

  208.8, Longest constructive episode, in seconds
  148.5, Longest meta-housekeeping episode, in seconds
  202.3, Longest housekeeping episode, in seconds

    0.0, Total seconds spent editing nodes
   85.4, Total seconds spent in help
  252.3, Total seconds spent in long pauses
 1480.8, Total seconds in all pauses
    0.0, Total seconds spent in comments
    0.0, Total seconds spent outside Prose II

  475.6, Total seconds in constructive episodes
  449.3, Total seconds in meta-housekeeping episodes
 1181.3, Total seconds in housekeeping episodes

 2106.2, Total seconds in this session
```

Figure 24. Example session summary information, for session S16R0102

```
.RCD.XXX:
    PASS0                           <$*.RCD >$*.P0
    PASS1                           <$*.P0  >$*.P1
    PASS2       <$*.P1 | PASS3              >$*.P3
    PASS4                           <$*.P3  >$*.P4
    PASS5       -I                  $*      >>NTRIV.SUM

S01R0101.XXX:   S01R0101.RCD

S01R0201.XXX:   S01R0201.RCD
  ⋮
```

Figure 25. An example Makefile for controlling the parsing process. The programs are explained below; the command line parameters are discussed in Appendix C on page 205.

In the example Makefile shown in Figure 25 on page 91, files with the extension .XXX must be younger than files with the extension .RCD to avoid having the programs run. So, starting with the first file, *Make* compares file S01R0101.XXX to file S01R0101.RCD. Since the first file does not exist (and hence is not younger), *Make* carries out the operations shown at the top. The symbol "$*" serves as a wildcard to replace the current filename.

Program PASS0 reads file S01R0101.RCD as its standard input, and writes its standard output to file S01R0101.P0. When this is complete, program PASS1 reads from S01R0101.P0 and writes to S01R0101.P1. Program PASS2 reads from S01R0101.P1 and writes its standard output directly to program PASS3, which is waiting on it for input. In this way, the parsing continues through pass 4 and the summary pass. Program PASS5 (the summary pass) appends its summary output to file NTRIV.SUM, which, in this case, is a summary file for all of the non-trivial sessions.

When the parsing of file S01R0101.RCD is complete, *Make* begins the parsing for file S01R0201.RCD.

```
.RCD.XXX:
    PASS0   <$*.RCD | PASS1 | PASS2 | PASS3    >>EPISODN.SUM

S01R0101.XXX:   S01R0101.RCD

S01R0201.XXX:   S01R0201.RCD
  ⋮
```

Figure 26. An example Makefile for examining Pass 3 output. A protocol record is read from stdin by Pass 0, then piped through Pass 1 and Pass 2 into Pass 3. Its output is appended to file EPISODN.SUM. The symbol "$*" serves as a wildcard to replace the current filename.

4.2.4.5 Parser Implementation Details

The parser consists of ten executable modules: `PASS0.EXE` through `PASS5.EXE` plus four supplemental modules. These were constructed from 26 source files, in addition to sharing about ten files with the *Prose II* source code. The parser source code files were coded with LEX, YACC, and *C* language source.

In these 26 files are a total of 7,845 lines, which includes comments and blank lines. In source lines of code, the parser is about 44% of the size of *Prose II* itself.

Parser source code file sizes, 26 files

```
  range:  18 lines to 1160 lines
 median: 214 lines
   mean: 302 lines
std dev: 281
```

The mean line length, including blank lines, in these files was 40.5 bytes. The 26 source files totaled 318,048 bytes in size.

Appendix C on page 205 list the command line parameter options available for each pass of the parser.

CHAPTER 5. OBSERVATIONS ON THE TASK AND USER STRATEGIES

I conducted a study of users in their field locations to test the ideas proposed in *Prose II* and its parser. Observations on the results of this study are described in this chapter. The chapter begins with an introduction to the subjects and the experimental setup. An overview of the sessions, protocols, and user documents follows. The bulk of the chapter is an examination of detailed questions about how users spent their time with the software. A portrait of a typical session, built from the analysis results, closes this chapter.

5.1 Experimental Setup and Subjects

As mentioned earlier in this paper, a protocol, in cognitive psychology, is a report of the steps performed by a subject in attempting some task. The *Prose II* software, used by subjects in this study to design and write structured documents, included a tracker that automatically made a protocol recording of each subject's sessions with the software. When we designed this study in June 1988, our goal was to collect about 100 protocol records from 20 subjects using *Prose II*.

I made *Prose II* freely available over the IBM corporate network worldwide. The potential subjects were adult professionals who do expository writing as part of their daily jobs. They already had the hardware and software required to use *Prose II*. Between January 1988 and January 1989, 210 different people requested copies of *Prose II*.

In the middle of January 1989, I sent a cover letter to each of these individuals describing the motivation for my planned study and the procedure for returning session recordings to me over the network (see Appendix A on page 163 for the text of the letter). I also sent the latest version of the *Prose II* software with the tracker activated, and its user's guide. At the end of the data collection period (the end of February 1989), I had received a total of 112 session protocol recordings from 29 of the 210 potential subjects. This is a response rate of about 14%.

The cover letter explained the exploratory nature of this study, which assumed few experimental controls. These are discussed further in Section 1.1.4 on page 7.

5.2 An Overview of the Protocol Data and Parser Results

In this section, the 112 protocol records are categorized and itemized. For example, was a given document constructed in one or multiple sessions? Did it contain extensive writing in the files associated with the nodes? Overall statistics describe how many files were returned by the subjects, how many documents were constructed, and how many sessions were used to complete a document. The 112 session protocols are exhaustively listed in Table 40 on page 164, grouped in chronological order by the person who submitted them.

5.2.1 Five Categories of Sessions

One of the uncontrolled aspects of the study was not anticipated and became an early problem. There was no restriction on the number of sessions per subject, on the number of documents per subject, on the number of documents per session, or on the number of sessions per document. Further, there was no requirement that any meaningful work be done in a session. The tracker and analysis tools were designed to automate an analysis like Card, Moran, and Newell described in their *ICARUS* study; I had assumed one standalone session per document, with the work on a single document done from beginning to end in a session. Of course, humans use tools as they wish, despite the presumed intentions of their designers!

We categorized the 112 sessions many ways, but eventually the key separation was the nature of the work done in a session and how pieces of work spanned sessions. Figure 32 on page 168 shows the categorization of the 112 sessions into five groups.

42 sessions are considered trivial

38% of the sessions contained little or no substantive work. A subject would start a session, create a node or two, try some of the features, then end the session abruptly with no save of the workspace. Such trivial sessions are usually readily apparent on visual examination; more rigorously, a trivial session consists of one of the following situations:

- The subject never saved the workspace.
- The subject opened a file, but never used it.
- The session was short (several minutes or less), with little work done and no follow-up session.

The parse trees for these sessions were not of primary interest for this study. An example of a protocol record for a trivial session is shown in Figure 10 on page 50.

27 sessions are standalone work on one document

These sessions showed a one-to-one correspondence between a session and a complete document. All subjects saved the workspace at least once. The bottom 5 of these 27 sessions in Figure 32 on page 168 contained substantial writing in most or all of the nodes in the document.

37 sessions constitute work on documents spanning multiple sessions

Many of these sessions have no saves of the workspace, but in all of those sessions, at least one node was edited. This implies the underlying node(s) file was changed and saved. In all but three of these 37 sessions, nodes were edited.

5 sessions are editing of existing documents

These five sessions were distinguished by one open of an existing document; most nodes were not created in sessions submitted by subjects. The document may or may not have been originally created using *Prose II*. Four of these five sessions were somewhat similar to one another; they included one or two creates and one link command. Session S14R0105 had 12 creates, 12 links, and 5 moves.

3 sessions constitute work on multiple documents in one session

Three sessions consisted of work on two documents during each of those sessions.

This adds up to 114 sessions. Two sessions (S08R0201 and S29R0103) are counted twice since they included follow-up editing of an existing document plus work on a new document with multiple opens and saves.

Figure 33 on page 169 shows yet a different subset of these 112 sessions. These 10 sessions were non-trivial sessions where the subjects used *Prose II* as more than just an outline processor; they wrote substantial text in the files behind the nodes.

5.2.2 Documents and Sessions per Subject

Given an uncontrolled nature study of this nature, how many documents might researchers expect subjects to work on? Although several subjects returned 10 or more recordings, no subject worked on more than 4 documents. Table 19 on page 95 shows the distribution of the 45 total documents among the 29 subjects. Note that for four subjects, all of the protocol records they submitted were trivial.

Table 19. Distribution of 45 documents among 29 subjects

Number of documents	*Number of subjects*
0	4
1	12
2	5
3	5
4	2

How were subjects' documents constructed: over one session or over many sessions? How many sessions might researchers expect a writer to use to construct a document with this system? Table 20 on page 95 shows how the 70 non-trivial sessions were distributed among the 45 documents. About three-quarters of the documents were operated upon with *Prose II* in only a single session. No more than six protocol files were returned for any document.

Table 20. Distribution of 70 non-trivial sessions among 45 documents

Number of sessions	*Number of documents*
1	34
2	3
3	4
4	2
5	1
6	1

5.3 Results and Discussion

This section is the core of the protocol analysis results. It consists of four large subsections:

1. Time Distribution: looks at the range of time durations for documents, sessions, phases, episodes, and pauses.
2. Frequency Distribution: looks at the range of frequency distributions for the types of phases, constructive episodes, housekeeping episodes, and commands. Also examines the relationships among these: commands per episode, episodes per phase, commands per session, and so on.
3. Command Usage: looks at the details of commands important in understanding user behavior. For example, how many nodes were created, how were nodes labeled and edited, how was help used, how was file I/O done, and so on.
4. Overall Patterns: looks at questions that span the sessions. For example, how big are trees, which node is the final root, how does planning precede writing, and so on.

5.3.1 How Long are the Time Periods?

Designers of computer software rarely have a quantitative measure of the time scale of users' sessions with their software. How long did sessions with *Prose II* last? How long did subjects work on a document, even if it spanned multiple sessions? How long were the phases that the parser identified, how long were the constructive episodes, and how long were the housekeeping episodes that separated them? How long did subjects pause between commands, and how much of the total session time did these pauses consume?

This kind of information was readily extracted from the protocol records. These questions are answered in this subsection. In general, there was a wide difference between the median and mean values for any measurements of time, as well as a large standard deviation. Almost all frequency distributions of timing measurements were positively skewed, showing a curve with an early peak and a long trailing tail.

5.3.1.1 Overall Timings for Sessions, Phases, and Episodes

Table 21 on page 96 presents overall timing distributions for the sessions, phases, constructive episodes, and housekeeping episodes. The median time, along with the mean time and its standard deviation are presented. The median and mean often differ widely, and the standard deviation is sometimes many times the value of the mean. This shows how widely overall times varied.

Table 21. Overall timings for Sessions, Phases, and Episodes

	All 112 Sessions	*70 Non-trivial Sessions*	*42 Trivial Sessions*	*10 Sessions with Writing*
Session Times, in seconds				
median	743 (12 mins.)	1364 (23 mins.)	172 (3 mins.)	3168 (53 mins.)
mean	1447 (24 mins.)	2074 (35 mins.)	402 (7 mins.)	4789 (80 mins.)
std dev	2071 (35 mins.)	2370 (40 mins.)	571 (10 mins.)	3013 (50 mins.)
Phase Times, in seconds				
median	112	112	103	163
mean	322	330	264	541
std dev	589	608	430	939
Constructive Episodes Times, in seconds				
median	11	12	6	14
mean	49	51	19	101
std dev	152	157	36	236
Housekeeping Episode Times, in seconds				
median	21	20	29	21
mean	60	55	122	62
std dev	177	159	311	192

Discussion:

- The wide variation in times can be accounted for by some arbitrarily long factors inherent in the uncontrolled nature of the study:
  - Sessions can be any length; trees can be any size.
  - Housekeeping episodes can be arbitrarily long because they include long pauses.
  - Constructive episodes can be arbitrarily long because of writing time in a text editor.
- Comparing trivial and non-trivial sessions:

   - Non-trivial sessions were much longer in duration than trivial sessions.
   - The phase times were surprisingly close in value for non-trivial and trivial sessions. They differed in how the time was allocated between constructive and housekeeping episodes.
   - Non-trivial sessions had longer constructive episodes than trivial sessions; trivial sessions had longer housekeeping episodes than non-trivial sessions.

- The similar median phase times (about 5 minutes) across the different groups of sessions suggest consistent human behavior for periods lasting several minutes.

  In their *ICARUS* study, Card, Moran, and Newell noted three phases, of duration 7, 14, and 15 minutes. Thus, their mean phase time of 12 minutes can be compared against the mean of 9 minutes among the 10 sessions with writing in this study.

5.3.1.2 How is Session Time Distributed?

Table 22 on page 97 provides more detail on the first row of the preceding table. It shows the distribution of time durations among the 70 non-trivial sessions. This information might be used to answer the question of how long a session normally lasts.

Table 22. Distribution of session times among 70 non-trivial sessions

Session Time (minutes)	*Number of sessions*
14 or less	24
15 to 29	14
30 to 44	14
45 to 59	7
60 to 74	7
75 or more	4

Session times, 70 non-trivial sessions

```
   range:  0.8 minutes (S16R0103) to 218.3 minutes (S17R0101)
  median: 22.7 minutes
    mean: 34.6 minutes
 std dev: 39.5
```

A subset of these 70 sessions are the 10 sessions where a substantial amount of writing was done in the nodes (these session IDs are listed in Figure 33 on page 169). Their duration was more than twice as long as the median and mean duration for all 70 non-trivial sessions.

Session times, 10 sessions with substantial writing

```
   range: 32.5 minutes (S26R0101) to 196.9 minutes (S25R0301)
  median: 52.8 minutes
    mean: 79.8 minutes
 std dev: 50.2
```

Discussion:

- Most sessions without writing were less than twenty-five minutes in duration. Most sessions with writing lasted less than an hour. No session lasted more than four hours; the longest sessions included extensive pauses.

This suggests that controlled tests that study such software systems should consider having session times on the order of thirty minutes to an hour, as opposed to four hours.

In their *ICARUS* study, Card, Moran, and Newell analyzed one session of a circuit layout task that lasted 40 minutes. This data suggests their session time was of typical length.

- The many short sessions might be explained because work on some documents was divided over several sessions.

5.3.1.3 How is Document Time Distributed?

The following table shows the distribution of total time durations among the 45 non-trivial documents. Document times are different from the session times; a document may be composed of more than one session, or many documents may be worked on in a single session.

Table 23. Distribution of 45 document times among 70 non-trivial sessions

Document Time (minutes)	*Number of sessions*
14 or less	7
15 to 29	8
30 to 44	13
45 to 59	6
60 to 74	5
75 or more	6

Document times, 45 documents

```
  range:  2.4 minutes (S18R0102) to 218 minutes (S17R0101)
 median: 39.4 minutes
   mean: 53.7 minutes
std dev: 52.9
```

Discussion:

- Most documents constructed with *Prose II* were completed in less than forty-five minutes of total session time.

5.3.1.4 How Long are the Pauses Between Commands?

One of the hypotheses built into this parser is that pauses of moderate length occur between bursts of similar work. The two graphs that follow chart the duration of the pauses between consecutive commands. The parser identified 7370 pauses across all sessions.

Figure 27 on page 99 shows a "close-up" view of the pause durations; it looks at the frequency distribution of pauses lasting 16 seconds or less, where the distribution has been computed at 0.2 second intervals. Figure 28 on page 100 shows pause durations up to 100 seconds, in this case, distributed using 1.0 second intervals.

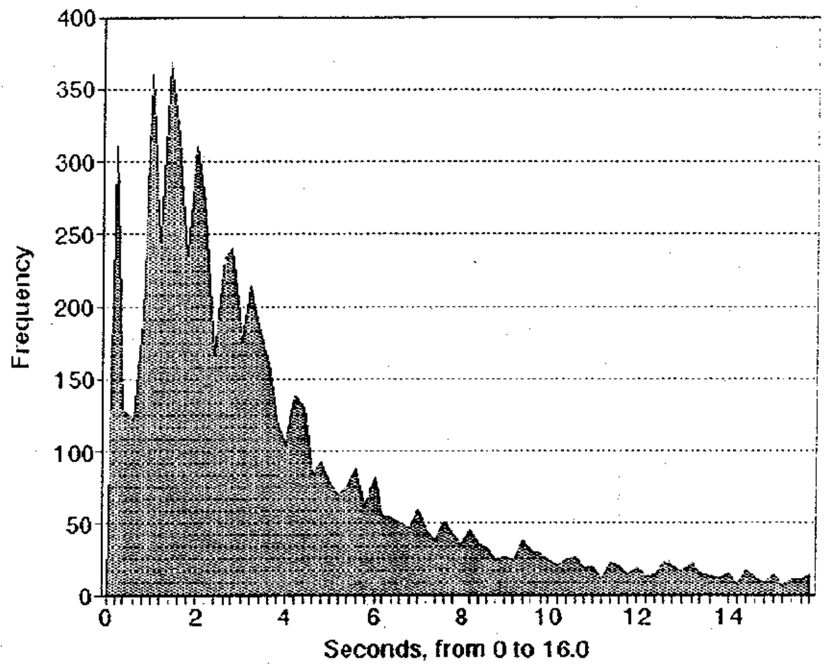


Figure 27. Frequency distribution of pauses between commands. This figure shows the distribution by 0.2 second intervals, from 0 to 16.0 seconds. The first peak in this graph is in the interval from 0 to 0.2 seconds. The next two peaks are in the intervals 0.8 to 1.0 and 1.2 to 1.4 seconds.

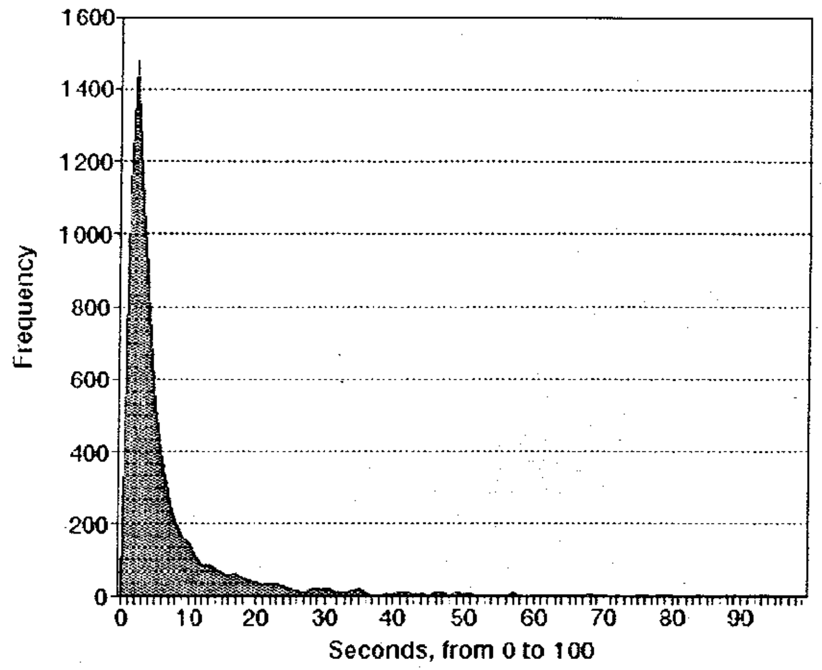


Figure 28. Frequency distribution of pauses between commands. This figure shows the distribution by 1.0 second intervals, from 0 to 100 seconds. The peak in the graph is between 1 and 2 seconds, where 20% of all pauses occur.

Table 24. Distribution of 7370 pauses among the four pause types

Type of pause	*Frequency*
< 6.0 secs	72.6%
6.0 to 8.5 secs	7.7%
8.5 to 100 secs	18.8%
100 secs or more	0.7%

Duration of a single pause, among all 7370 pauses

```
  range:  0.05 seconds to 2820 seconds
 median:  3.0 seconds
   mean: 10.0 seconds
std dev: 69.6
```

Duration of a single pause, excluding the 65 pauses greater than 100 seconds

```
  range:  0.05 seconds to 99.6 seconds
 median:  3.0 seconds
   mean:  6.4 seconds
std dev: 10.4
```

Discussion:

- The median pause duration was 3 seconds. Two-thirds of all pauses were less than 5 seconds.
- Card, Moran, and Newell used 5 seconds as their inter-command cutoff value in their single *ICARUS* protocol.
- In its version of freewriting, Addison Wesley's *Wordbench* puts a timer at the top of the screen and exhorts you to "KEEP WRITING" if you stop for more than 5 seconds. These results corroborate their design.
- When building the parser, my experience with pilot studies showed that there was a natural break in the frequency of pause durations after 6.0 seconds and after 8.5 seconds. This data shows I was close, but suggests 9.5 seconds might have been a better selection point. The handling of pauses by the parser is discussed in "Pauses" on page 74.
- The clock resolution of DOS, and thus the tracker, was 0.055 seconds. 2.6% of the pauses were 0.06 seconds or less, which is at about the resolution of the human cognitive processor that Card, Moran, and Newell describe (they specified the human processor cycle time as 0.07 seconds, with a range from 0.03 to 0.10 seconds).
- More sophisticated parsers can be envisioned where the meaning assigned to pauses of different durations could be adjusted on a sliding scale--as opposed to making decisions based on fixed points as was done here. To determine the classification of each pause, this sliding scale could use information it maintains about the subject, their experience, and the elapsed time so far in a session. This technique might also be used to identify individual differences in attention spans.

5.3.1.5 What Proportion of Session Time was Spent in Pauses?

Of the total time in a session, how much time was spent not actively executing commands with this software? Figure 29 on page 102 shows the frequency distribution of the proportion of total session time spent in pauses.

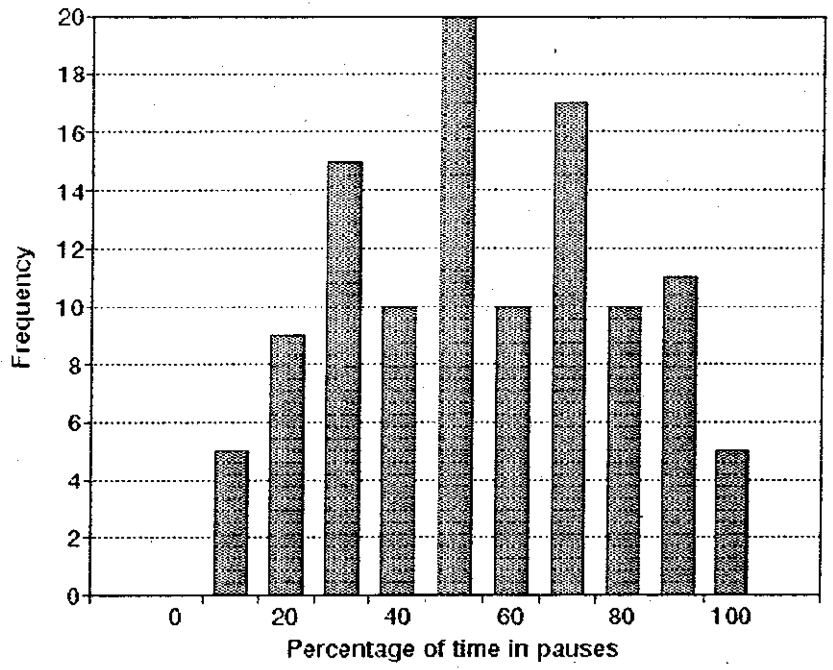


Figure 29. Frequency distribution of total session time spent in pauses. This chart shows how total pause time within a session was distributed among the 112 sessions. For example, in 20 sessions the total pause time was between 40% and 49% of the total session duration.

The proportion of total session time spent in pauses has the following size characteristics:

Proportion of total session time in pauses

```
  range:  1% (S10R0101) to 99% (S27R0108)
 median: 56%
   mean: 50%
std dev: 25
```

Discussion:

- About half of the total session time is spent in pauses, not actually executing commands. Long periods spent in a text editor are not included in the total pause time.

5.3.1.6 Summary: Time Durations

- The frequency distribution of most time durations showed large differences between the median and mean times, as well as large standard deviation values. Graphs of these distributions are positively skewed, showing curves with an early peak and a long trailing tail.
- Sessions were short, about a half hour or less, when no writing was involved.
- Sessions with writing lasted about an hour.
- Overall document times were about an hour.
- About half the total session time was spent in pauses.
- Half of all pauses were less than 3 seconds; two-thirds were less than 5 seconds. The measure of pause duration did not include the times spent for the execution of any commands. The testbed system was fast, with only two commands that did not appear to be instantaneous: opening and saving files. For all commands, the entire command completed before measuring the beginning of the pause.
- Mean and median phase times were close in value when comparing non-trivial and trivial sessions. This could imply that spans of human behavior are of similar length, whether the underlying activity is learning or doing productive work.

5.3.2 How are Parse Elements Distributed in Frequency of Use and Time?

Having looked at the distribution of the parse elements in time duration, this subsection examines how frequently the different types of phases, episodes, and commands occurred within these sessions.

5.3.2.1 Frequency Distribution of 7 Phases

Phases were the highest grouping applied by Card, Moran, and Newell. The parser divided sessions into one or more phases, each lasting several minutes. A single type of activity predominates in each phase. A phase consists of a sequence of cognitive tasks, known as episodes. Within a phase, episodes of constructive work alternate with periods of thought and housekeeping. While phases contain both types of episodes, only constructive episodes are used to characterize the phases.

What were the broad types of activity that writers engaged in? Of the seven types of phases defined by the grammar in this project, which were seen most often? How long did the phases last?

The seven different types of phases were described in detail in Table 17 on page 84. Table 25 on page 104 and Table 26 on page 104 shows the distribution of phase types and their durations.

Table 25. Non-trivial sessions: Distribution of the 7 phases types. This is sorted in order of frequency of the phases observed in the non-trivial sessions

Phase Name	*Distribution among 70 Non-trivial sessions*	*Percentage count among Non-trivial sessions*	*Median Phase time, Non-trivial sessions (seconds)*	*Percentage time among Non-trivial sessions*
Exploration	145	33.0%	106.5	27.2%
Define Hierarchies	103	23.4%	121.7	28.3%
Top Down Construction	101	23.0%	172.2	24.2%
New Workspace	37	8.4%	34.7	2.0%
Document Revision	31	7.0%	330.8	16.5%
Bottom Up Construction	18	4.1%	49.0	1.7%
Tree Structure Revision	5	1.1%	26.4	0.1%
Total	440	100.0%	111.9	100.0%

Table 26. Trivial sessions: Distribution of the 7 phases types. This is sorted in order of frequency of the phases observed in the non-trivial sessions

Phase Name	*Distribution among 42 Trivial sessions*	*Percentage among Trivial sessions*	*Median Phase time, Trivial sessions (seconds)*
Exploration	23	35.9%	165.3
Define Hierarchies	24	37.5%	100.2
Top Down Construction	1	1.6%	1316.9
New Workspace	15	23.4%	123.4
Document Revision	0	0%	0
Bottom Up Construction	1	1.6%	12.4
Tree Structure Revision	0	0%	0
Total	64	100.0%	103.4

Table 27 on page 104 shows the percentage of time spent in each of the types of phases.

Table 27. Distribution of time spent among the 7 phases, among all sessions

Type of phase	*Duration, among total session time*
Exploration	27.3%
Define Hierarchies	30.0%
Top Down Construction	22.5%
New Workspace	3.8%
Document Revision	14.8%
Bottom Up Construction	1.5%
Tree Structure Revision	0.1%
Total	100.0%

Discussion:

- A third of the phases were involved with exploration.

  Exploration includes any initial writing done in a node.

- There was much more top-down construction than bottom-up construction.

  - The number of bottom-up episodes is 18% the number of top-down episodes. By time duration, 15 times more time was spent in phases of top-down construction than in phases of bottom-up construction.

- Although common, this may not be the most desirable strategy. Lansman, Smith, and Weber (1990) noted in their experiments a significant negative correlation between top-down score and the quality of subjects' documents. "Those subjects who tended to generate lower level ideas first wrote higher quality reports."

- These results indicate that *Prose II* was rarely used to revise a document's hierarchical structure.

- About one-sixth of the total time was spent in phases of document textual revision. These were the longest phases (in median time duration).

- Pianko (1979) reported that college freshman devote less that 9% of their composing time to reading and revising. By count, the results of this study show that 8.1% of the phases were used for document and structure revision. By time, 16.7% of the total session time was spent in revision, a result somewhat higher than Pianko's report. This may be accounted for by the difference in strategy between college freshman and technical professionals.

5.3.2.2 Frequency Distribution of 12 Constructive Episodes

The parser characterized the types of constructive episodes in a session. A constructive episode consists of series of distinct user commands with short elapsed time between them. Multiple overlapping goals may be attempted by a subject during a single constructive episode: for example, they might move some nodes and delete some nodes that are no longer needed, so that they might then create, label and link a new set of nodes. Section 4.2.3.4 on page 78 discusses how Pass 3 of the parser produces the characterizations of constructive episodes from the sequence of primary and secondary commands. The twelve different types of constructive episodes are described in Table 12 on page 79. Table 28 on page 105 show the distribution of constructive episode types and their duration.

Table 28. Distribution of 12 constructive episodes among the non-trivial and trivial sessions. This is sorted in order of frequency of the episodes observed in the non-trivial sessions

Constructive Episode Name	*Distribution among 70 Non-trivial sessions*	*Percentage among Non-trivial sessions*	*Median Episode time, Non-trivial sessions (seconds)*	*Distribution among 42 Trivial sessions*	*Percentage among Trivial sessions*	*Median Episode time, Trivial sessions (seconds)*
Created solo nodes	226	17.6%	13.4	14	14.1%	6.9
Edited existing nodes	219	17.0%	37.4	4	4.0%	14.8
Grew existing trees	183	14.3%	25.5	2	2.0%	22.7
Hooked existing nodes	147	11.4%	7.8	1	1.0%	2.8
Moved existing nodes	131	10.2%	3.9	3	3.0%	8.0
Created new trees	82	6.4%	15.4	14	14.1%	9.4
Revised existing nodes	82	6.4%	21.4	4	4.0%	25.7
Deleted nodes	74	5.8%	2.3	5	5.1%	0.1
Unproductive work	64	5.0%	7.8	19	19.2%	14.3
Start over	36	2.8%	3.2	32	32.3%	4.8
Assembled trees	20	1.6%	8.2	1	1.0%	3.2
Broke existing links	20	1.6%	5.8	0	-	-
Total	1284	100.0%	11.6	99	100.0%	6.1

Discussion:

- The median time duration for all constructive episodes was about twice as long in non-trivial sessions as in trivial sessions.

- Much of the time in constructive episodes was spent in iterative refinement: growing existing trees and hooking nodes into them.
- The median times for episodes of creating and editing nodes were longer for non-trivial sessions than for trivial sessions. This is reasonable, since productive words were presumably being written during the non-trivial sessions.
- By count, 5% of the constructive episodes in the non-trivial session was unproductive work, as opposed to 19% in the trivial sessions. The unproductive work episodes were of longer duration in trivial sessions than in non-trivial sessions.
- In their *ICARUS* study, Card, Moran, and Newell found their mean episode time was 25 seconds. Their definition of an episode did not make the distinction between types of episodes—housekeeping and constructive—shown in this study. To get to a comparable measure, the median duration for both types of episodes can be summed. This combination of one housekeeping episode and one constructive episode could be said to last about 32.0 seconds, for non-trivial sessions. A typical episode time for trivial sessions was surprisingly close in duration: about 34.6 seconds.

5.3.2.3 Frequency Distribution of 7 Housekeeping Episodes

Between consecutive constructive episodes are periods of system operations, including pauses to think and housekeeping operations. The parser determined each of these episodes of system activity. Seven different types of housekeeping episodes were described in detail in Table 10 on page 76. Table 29 on page 106 shows the distribution of housekeeping episodes and their duration.

Table 29. Distribution of 7 housekeeping episodes among the non-trivial and trivial sessions. This is sorted in order of frequency of the episodes observed in the non-trivial sessions

Housekeeping and Meta-housekeeping Episode Name	*Distribution among 70 Non-trivial sessions*	*Percentage among Non-trivial sessions*	*Median Episode time, Non-trivial sessions (seconds)*	*Distribution among 42 Trivial sessions*	*Percentage among Trivial sessions*	*Median Episode time, Trivial sessions (seconds)*
Medium & Long Pause	756	51.7%	16.5	66	53.7%	23.1
Refocus	361	24.7%	30.0	21	17.1%	48.0
Cleanup & Take Stock	93	6.4%	30.2	2	1.6%	33.8
Cleanup	88	6.0%	9.6	5	4.1%	7.1
Help Request	72	4.9%	45.0	15	12.2%	74.9
Take Stock	72	4.9%	37.2	7	5.7%	24.6
Tracker Comment	20	1.4%	59.5	7	5.7%	132.4
Total	1462	100.0%	20.4	123	100.0%	28.5

Table 30 on page 107 shows the percentage of time spent in each of the types of episodes. It shows the percentage of time among the total housekeeping episode time, as well as the percentage of time among the total session time.

Table 30. Distribution of time spent among the 7 housekeeping episodes

Type of housekeeping episode	*Duration, among total housekeeping episode time*	*Duration, among total session time (includes constructive episodes)*
Medium & Long Pause	60.8%	35.6%
Refocus	18.9%	11.1%
Cleanup & Take Stock	4.1%	2.4%
Cleanup	1.4%	0.8%
Help Request	7.2%	4.2%
Take Stock	4.5%	2.6%
Tracker Comment	3.0%	1.8%
Total	100.0%	100.0%

Discussion:

- As defined in this study, about half of the housekeeping episodes consist of long pauses.
- There were many housekeeping episodes involving roaming and zooming in the workspace. This suggests that the screen is probably too small. As a software developer, I believe I should be building systems with fewer occurrences of Refocus, and with less time per occurrence. For example, in this study, 11.1% of the total elapsed time among all the session was spent in Refocus episodes (see Table 30).

  An area for a follow-on study is a controlled experiment, where two different screen sizes are used. If the amount of time spent doing Refocus activities decreased with larger screens, one could build a case for doing a cost analysis, comparing the cost in employee time for using a small screen as opposed to using a large one.
- Episodes of Cleanup only were relatively short in duration; their mean duration was about 10 seconds, as opposed to about 20 seconds for the housekeeping episodes as a group.
- Trivial sessions contained a higher proportion of help requests and tracker comments, and the time for these was longer than for non-trivial sessions.

5.3.2.4 Frequency Distribution of 39 Commands

One of the most valuable tools to a software engineer for tuning a system is a code profiler. What code was executed, and with what frequency? Similarly, designers of user interfaces need feedback on which features of the interface are used most frequently. What are the most-frequently-used commands—they should probably be the easiest to use? Are there common subsequences that might be grouped together? Are there commands that are never used? Are there commands used by experts, but not by novices?

The tracker in *Prose II* recorded 39 different commands, in addition to the Pause. Across the 112 sessions, all of the commands were used at least once. Table 31 on page 108 shows how the commands were distributed among the non-trivial and trivial sessions. The table is sorted in order of frequency of the commands used in the non-trivial sessions. A total of 6225 commands were performed among the non-trivial sessions; 447 commands were performed among the trivial sessions.

Table 31. Distribution of the 39 *Prose II* commands

Command Name	*Distribution: 70 Non-trivial sessions*	*Percentage: Non-trivial sessions*	*Distribution: 42 Trivial sessions*	*Percentage: Trivial sessions*
LinkNodes	1079	17.3%	38	8.5%
EditLabel	1075	17.3%	34	7.6%
CreateNode	974	15.6%	47	10.5%
MoveNode	515	8.3%	7	1.6%
MapWindowRoam	359	5.8%	12	2.7%
EditNode	287	4.6%	24	5.4%
TidyWorkspace	238	3.8%	6	1.3%
MapWindowZoom	233	3.7%	6	1.3%
LoseFocus	204	3.3%	51	11.4%
HelpRequest	166	2.7%	47	10.5%
MapWindow	154	2.5%	16	3.6%
DeleteNode	142	2.3%	17	3.8%
SaveWorkspace	122	2.0%	6	1.3%
OutlineWindow	63	1.0%	15	3.4%
MapMove	60	1.0%	9	2.0%
MainWindowZoom	59	0.9%	5	1.1%
SetTidyMode	56	0.9%	2	0.4%
OpenWorkspace	53	0.9%	36	8.1%
BreakLink	48	0.8%	2	0.4%
MainWindowReset	48	0.8%	16	3.6%
SetDeleteMode	38	0.6%	4	0.9%
SystemIcon	31	0.5%	4	0.9%
OutlineWindowSize	31	0.5%	12	2.7%
TrackerComment	28	0.4%	10	2.2%
OutlineWindowMove	27	0.4%	8	1.8%
CopyNode	26	0.4%	0	0.0%
SystemZoom	20	0.3%	5	1.1%
GoTo	19	0.3%	1	0.2%
NewWorkspace	16	0.3%	3	0.7%
ChangeDefault	16	0.3%	3	0.7%
Scramble	12	0.2%	1	0.2%
SystemMove	5	0.1%	0	0.0%
TreeShrink	4	0.1%	0	0.0%
TreeGrow	4	0.1%	0	0.0%
MapSize	4	0.1%	0	0.0%
BreakAllLinks	3	0.0%	0	0.0%
SystemSize	3	0.0%	0	0.0%
ClearDrawing	2	0.0%	0	0.0%
ClipboardCopy	1	0.0%	0	0.0%
Total	6225	100.0%	447	100.0%

Discussion:

- Among the non-trivial sessions, more than 50% of the commands performed were either creating, linking, or labeling of a node, with the other 36 commands occurring far less frequently.
- Among the trivial sessions, the three most frequent commands were 1) leaving *Prose II* for another application, 2) requesting help, and 3) creating a node.
- The bottom eight commands in Table 31 are good candidates for removal, or placement on an "Advanced" menu. They were rarely used in the non-trivial sessions and never even tried in the trivial sessions.

5.3.2.5 Relationship between Commands, Episodes, Phases, and Sessions

What is the relationship among the parse elements? How many commands were there per episode, how many episodes in a phase, and how many phases in a session? I looked at these relationships in two ways: among the group of 27 single-document sessions, and among all 70 non-trivial sessions. These results are unusual among those in this chapter; their standard deviations are small compared to their means and medians!

Table 32. Ratios of commands, episodes, and phases among 27 single-document sessions

	Commands per episode	*Episodes per phase*	*Phases per session*
range:	2.0 to 8.0	2.0 to 11.0	2.0 to 36.0
median:	4.0	4.3	6.0
mean:	4.0	4.8	9.4
std dev:	1.3	2.2	7.6

Table 33. Ratios of commands, episodes, and phases among 70 non-trivial sessions

	Commands per episode	*Episodes per phase*	*Phases per session*
range:	1.2 to 8.0	1.5 to 13.0	2.0 to 36.0
median:	2.9	4.0	4.0
mean:	3.1	4.4	6.3
std dev:	1.3	2.2	6.3

Discussion:

- Subjects worked in longer bursts in the 27 single-document sessions than in the 70 non-trivial sessions. There were more commands per episode and more episodes per phase in these 27 sessions.
- The standard deviation values are small in the commands per episode and episodes per phase. Also, the median and means are relatively close together. This suggests a consistency in the definition of these terms and in how they are parsed. It may further suggest that humans are consistent in short time periods.

  As a topic for follow-up research, it would be interesting to see how these values changed as the time duration values were changed in the parser (*e.g.*, PauseType0—see Section "Pauses" on page 74).

  These chunks (that is, the commands per episode and episodes per phase) are about the size of human Working Memory chunks: in the range of 3 to 5 elements.
- The number of episodes per phase is fairly consistent, even among the different types of sessions. Table 21 on page 96 shows that the timings for phases was consistent among all types of sessions.

  This suggests that the definition of phases, in both time and number of episodes, was consistent among all types of sessions.
- There is large variation in phases per session, since session lengths and document sizes vary considerably.

5.3.2.6 Number of Nodes vs. Time Spent in a Session

What's the cost, in time, of adding new nodes to a document? Is the cost per node higher for large documents or for small documents? What's the time difference between sessions where subjects only labeled nodes, as opposed to sessions where they wrote substantial text for the nodes?

In an effort to understand the "cost" of creating a node, I looked at the number of nodes saved in the single-session documents, compared to their total session time.

Time per node, 22 single-session documents without substantial writing

```
  range:  22 seconds/node (S17R0201) to 631 seconds/node (S10R0104)
 median:  76 seconds/node
   mean: 113 seconds/node
std dev: 124
```

These 22 sessions are listed in the second column of Figure 32 on page 168.

Time per node, 10 sessions with writing

```
  range:  94 seconds/node (S18R0108) to 692 seconds/node (S15R0102)
 median: 386 seconds/node
   mean: 427 seconds/node
std dev: 206
```

These 10 sessions are listed in the two columns of Figure 33 on page 169.

Discussion:

- The data analysis showed that the time per node decreases as the number of nodes increases. The decrease in the time per node as the number of nodes increases may be explained by the fixed startup and takedown costs in a session.
- When writing in the nodes, the time per node increases by a factor of about 4 to 6 times.
- In the sessions of labeling without writing, subjects devoted about a minute to each node. A recent report supports this finding:

  > Psychology students using a new computer program were able to generate 86 ideas in 78 minutes (*i.e.*, 54 seconds/idea), while students who tried brainstorming at random produced 55 ideas in 55 minutes (*i.e.*, 60 seconds/idea). The program, called *IdeaFisher* (from Fisher Idea Systems in Irvine, California) allows creative "navigation" through 370 broad topics, 65,000 words and phrases, and 675,000 cross references. (Roberts, 1989)

  A minute per idea can serve as a predictor of overall session time.
- Slow sessions can be identified. For example, S23R0101 was a relatively long session for the number of nodes, yet there were no exceptionally long pauses and no writing in the nodes. However, the labels were long (a mean of 49.4 characters/label).
- S10R0104, which showed the long rate of 421 seconds per node, consisted simply of building a tree with 6 nodes and labeling them. This points out a deficiency in the parser: inadequate handling for one of the EditLabel commands that lasted 48 minutes.

5.3.2.7 Number of Nodes vs. Number of Commands in a Session

Subjects executed commands in a session for many different reasons, as we have seen: both for housekeeping and for constructive work. What's the relationship between the number of nodes and the number of commands in a session? The next set of statistics compares the number of nodes created in a session with the total number of commands in a session. Counted among the commands are all "non-pauses" recorded by the tracker; this includes LeaveProseII, HelpRequest, and TrackerComment. Nodes were created in 56 non-trivial sessions (in the other 14 sessions, existing nodes were edited, but none were created).

Commands per node

```
   range: 3.6 commands/node (S06R0301) to 48.0 commands/node (S27R0104)
  median: 6.2 commands/node
    mean: 8.8 commands/node
 std dev: 7.8
```

Discussion:

- During a session, subjects performed about 4 to 8 commands per node.

  Since an expected command sequence is CreateNode, EditLabel, and LinkNode for each node, three commands per node would be the reasonable minimum. Thus, session S06R0301 (at the low end of the range) appears straightforward. Examination of this session indeed shows an extended sequence of creating, labeling, and linking node, with little Revision or Refocus activity.

- The commands per node decreases slightly as the number of nodes increases. The decrease in the number of commands per node as the number of nodes increases may be explained by the fixed startup and takedown costs in a session.

5.3.2.8 Summary: Distribution of Commands, Episodes, and Phases

- Almost 80% of the phases involved exploring, defining hierarchies, or constructing trees in a top-down manner. Subjects rarely used *Prose II* for textual revision or structural changes.
- There was a preponderance of top-down construction, as opposed to bottom-up construction. Much of the top-down construction took place as iterative refinement.
- A quarter of the housekeeping episodes were concerned with refocus operations, perhaps because of the small screens used by the subjects.
- Half of the commands can be accounted for with creating, labeling, and linking nodes.
- The counts of commands per episode and episodes per phase were surprisingly consistent, with small variation.
- As a rule of thumb, subjects spent a minute or two per node if writing was not involved in the session; if writing was involved, they spent about 5 to 10 minutes per node.

5.3.3 How Were Particular Commands Used?

Some specific *Prose II* commands are examined here in greater detail. These include the commands for creating, deleting, labeling, and writing text in nodes, as well as requesting help, opening and saving a workspace, and tidying trees. Finally, all of the comments that subjects left in the protocol records are listed.

5.3.3.1 Creating and Deleting Nodes

Counting nodes in a session is an ambiguous problem. At various points in the analysis, I encountered four different ways of counting the nodes created in a session or document:

1. Count the number of create commands in a session.

   This misses the copy command, but CopyNode was only used in 3 of the 112 sessions. This also omits nodes created in a previous session and present in the current session because the subject used the OpenWorkspace command.

2. Count the number of nodes saved in the last save of a session.

   This misses substantial work on nodes that were deleted before the save, and it includes nodes from existing documents created in other sessions.

3. Count the total nodes for a given document, since they are uniquely numbered.

   This counts all deleted nodes (which can be many) and can also include nodes from existing documents created in other sessions.

4. Count the nodes actually touched by any explicit command in a session.

   For example, in one session (S14R0103) a file with 45 existing nodes was opened; the constructive work in that session consisted of writing text in 8 of the nodes.

There was not a straightforward solution to this ambiguity. I have used the first definition consistently in this section, unless it is identified otherwise.

Table 34. Number of nodes created in different types of documents

	Non-trivial Documents	*Single-session documents*	*Multiple-session documents*
median:	17.0	19.0	14.0
mean:	22.4	26.3	14.2
std dev:	17.1	19.4	4.8

Discussion:

- In the multi-session documents, the median and mean value for the number of nodes in the documents is similar (about 14 nodes) and the standard deviation is comparatively small. I cannot directly account for this; I would like to look at a larger sample size.
- In one unusual session (S17R0101), 36 nodes were deleted. Otherwise, in the remaining 111 sessions, the most nodes deleted in a single session was 9; the mean was 1.0 nodes deleted in a session. 70 sessions had no deletes. Of the sessions with deletes, the mean was 3.0 nodes deleted.
- The effectiveness of having a separate Delete Mode would be seen if no nodes were accidentally deleted—this allows *Prose II* to get by without an Undo command. The protocols could be examined for evidence of accidental deletion—but this is hard to determine, since nodes were frequently created soon after a delete. On the contrary, two-thirds of the deleted nodes were the last node created.
- A total of 945 nodes were created using the CreateNode command in all sessions, and 119 nodes were deleted. In aggregate, about 12.6% of the nodes created were deleted. This count of deleted nodes does not include those deleted because of the NewWorkspace, ClearWorkspace, or OpenWorkspace commands; these three commands delete all existing nodes in a workspace. Similarly, this count of created nodes does not include nodes created by opening existing documents using the OpenWorkspace command.

- Subjects used the CopyNode command to create nodes in only 3 of the 112 sessions. It was used in session S17R0101 22 times; in the two other sessions, it was used twice.

5.3.3.2 Labeling Nodes

As a software designer, how should I plan for the labeling of nodes in an outline? How much time is spent labeling a node (*i.e.*, can other background activity be occurring during this period)? How big can node labels become: if my software is allocating internal memory blocks, how big might these labels grow?

Subjects labeled nodes in all but two of the 70 non-trivial sessions.

S25R0101: There was writing in all eight nodes, but none of the nodes were labeled.

S25R0301: The subject built a substantial tree with 21 nodes, wrote text in the files associated with all of them, but labeled only one of them.

How much time was spent labeling a node? Across the 112 sessions, node labels were edited 954 times. The durations for these periods in the EditLabel dialog box are shown below.

Time spent labeling a node

```
  range:  0.3 seconds to 2841 seconds
 median: 11.9 seconds
   mean: 17.8 seconds
std dev: 34.5
```

How long are the labels used for the nodes? Across the 112 sessions, the statistics are shown below:

Label length, in characters

```
  range:  2 characters to 189 characters
 median: 18 characters
   mean: 24.7 characters
std dev: 21.2
```

Running a word-counting program against a file of all the labels showed that the 112 sessions had a total of 3482 words.

Label length, in words

```
mean: 3.6 words/label
      6.8 characters/word
```

Discussion:

- These subjects expressed their concepts with a few choice, long words. Compare 6.8 characters/word with the standard measure mentioned by Card, Moran, and Newell: 4.8 characters per word (for telegraphic data, from 1898).
- There was wide variation in the time spent labeling, but narrower variation in the number of characters in the labels. This suggests some consistency in the size of the character strings used for labels, which may be a side-effect of the displayed size of the nodes and the font

shown in the system. Subjects rarely entered labels that were significantly longer than the size of a node, which would have caused the label to be truncated during normal viewing.

- One session, S22R0101, was interesting in that it had somewhat long labels, yet was efficient in the number of nodes created and labeled during the elapsed session time. A fast typist.

5.3.3.3 Writing Text in Nodes

While most subjects labeled their nodes, few used a text editor to write extensive text to be associated with their nodes. When writing did occur, how much time was spent doing this writing? Hayes and Flower (1986, p. 1109) noted that "Even for the most extensive outliners, the ideas noted in the outline were expanded on the average by a factor of eight in the final essay." Were similar results seen in this study of writers?

In 54 of the 70 non-trivial sessions, subjects edited at least one node; 16 sessions contained no writing at all. However, only 9 documents involved sessions with substantial text editing. 3 of these 9 were additional work on existing documents; 6 were newly-created documents.

How much time was spent in one trip to an editor? For the 54 non-trivial sessions with periods of writing:

Time spent writing with a text editor, per node

```
  range:   0.1 seconds to 2136 seconds
 median:  37.3 seconds
   mean: 119.6 seconds
std dev: 235.1
```

What amount of total session time was spent writing?

Proportion of session time spent writing, 54 sessions with any writing

```
  range:  0.1% (S22R0101) to 84.7% (S15R0102)
 median: 16.7%
   mean: 25.5%
std dev: 26.5
```

Proportion of session time spent writing, 10 sessions with substantial writing

```
  range:  7.8% (S13R0101) to 84.7% (S15R0102)
 median: 44.7%
   mean: 50.8%
std dev: 24.9
```

Total writing time in a session, 54 sessions with any writing

```
  range:  0.1 minutes (S22R0101) to 113.3 minutes (S15R0102)
 median:  1.1 minutes
   mean: 10.6 minutes
std dev: 22.4
```

Discussion:

- Most subjects experimented with using a text editor, but few used it for actual document writing.
- In the 10 session with writing, about half the total session time was spent doing that writing.

  There was wide variation in the time spent writing with a text editor.
- Although *Prose II* could invoke any editor (*e.g.*, a paint program), no subjects used anything other than a text editor for work on an actual document. One reason: *Prose II* is not well suited to embedding graphics in final output. More extensive commercial programs, such as *PageMaker*, extend the idea of mixing text with graphics. *Prose II* was used principally as a 2-dimensional outline processor.
- Hayes and Flower's observation of an expansion by a factor of eight cannot be directly substantiated with the results of this study, because the number of words in the edited text was not collected by the tracker. However, the times for labeling and writing can be roughly compared. For the 54 sessions, both sets of time distributions have early peaks and long trailing tails:

```
Time spent labeling, per node      Time spent writing, per node

 median: 11.9 seconds              median:  37.3 seconds
   mean: 17.8 seconds                mean: 119.6 seconds
std dev: 34.5                     std dev: 235.1
```

  These mean values are consonant with the 8-to-1 ratio reported by Hayes and Flower.
- Subject 25 had two unusual sessions, where writing was done in large trees but with little labeling.

5.3.3.4 Requesting Help

Significant effort is invested in modern interactive software to provide online help. How often are these helps used? How long do people spend looking at the helps? Are they actually helpful?

Ten different help panels were available in *Prose II*. The full text for each of these panels is shown in Table 2 on page 45. Table 35 on page 115 shows how many times each of these helps was requested and how much time was spent viewing them. Help was requested in only 27 of the 70 non-trivial sessions.

Table 35. Distribution of help requests among all sessions

Name of the help panel	*Total number of requests*	*Median time spent viewing this help panel (seconds)*
Introduction	24	6.0
First Time	22	9.2
Mouse	31	11.6
Map Window	18	17.6
Delete/Tidy	30	20.1
File Formats	14	24.4
Editing	32	18.0
Changing	14	15.7
WIN.INI	15	32.4
Help Me	4	21.0

Time per help panel, 112 sessions

```
  range:  2.1 seconds to 205.9 seconds
 median: 12.9 seconds
   mean: 19.2 seconds
std dev: 21.7
```

Requesting help was the tenth most frequently-used command among the 70 non-trivial sessions.

Time per help panel, 70 non-trivial sessions

```
  range:  2.2 seconds to 87.7 seconds
 median: 12.8 seconds
   mean: 16.8 seconds
std dev: 14.9
```

Requesting help was the second most frequently-used command among the 42 trivial sessions.

Time per help panel, 42 trivial sessions

```
  range:  2.1 seconds to 205.9 seconds
 median: 14.2 seconds
   mean: 25.2 seconds
std dev: 34.0
```

Discussion:

- Subjects viewed help panels for about 15 to 30 seconds.
- Time spent in helps was longer in trivial sessions than in non-trivial sessions.
- There was wide variation in the amount of time spent viewing help panels. This variation was widest among the trivial sessions.
- Help for frequent commands (*e.g.*, using the mouse, labeling and editing) was requested more frequently than for less-frequently-used commands (*e.g.*, changing how the nodes are drawn).
- Help for complex operations (*e.g.*, updating the `WIN.INI` file, or dealing with the six file formats) took about twice as long as for others (*e.g.*, using the mouse).

  I can suggest two conflicting reasons why a user spends a long time reading a help panel:

  - The help panel contains lots of valuable information, worth reading to understand the particular problem it addresses, or
  - The help panel is hard to understand. It should be re-written and/or re-organized in a future release of the software.

  The *Prose II* tracker did not contain a mechanism to distinguish these conditions. It could only observe that some helps took longer than others. A simple addition might provide some additional insight:

  > Since the user must push an "OK" button already to exit a help, two buttons could be displayed instead. One could read "This was helpful," while the other could read "This was not helpful." Either selection would exit the help. The tracker could record

the choice, which could be accumulated across many sessions during the protocol analysis.

5.3.3.5 Opening and Saving a Workspace

A variety of file formats were available to users of this system. Each format has a software development cost, in its design, coding, and maintenance, as well as its size in the final product. Which formats were used most and least? Could some of the formats reasonably be omitted from the software, and yet satisfy the target users?

Across the 112 sessions, there were 87 OpenWorkspace and 125 SaveWorkspace commands performed. Table 36 on page 117 shows which file formats were used in these Opens and Saves. The .SCR format is used inside IBM as the basic format for document processing, so most of the existing files that were opened were in that format. The percentage of files saved in the .IND and .PR2 formats was larger than the percentage of files opened in those formats, implying that *Prose II* was sometimes used to translate from one format (particularly the .SCR format) to another.

No subjects used the .LST and .RDY formats, two file formats that I found useful in developing *Prose II*. The .CRD format, compatible with the Microsoft Cardfile application, was also rarely used.

For a review of these file formats, see Section 3.1.10.3 on page 43.

Table 36. Distribution of file formats used for Opens and Saves, among all sessions

File format	*Percentage of Opens*	*Percentage of Saves*
.SCR	57%	42%
.PR2	27%	39%
.IND	5%	17%
.CRD	2%	2%
.RDY	0%	0%
.LST	0%	0%

5.3.3.6 Tidying

The TidyTrees function was new to most users of this software. It offered a new working paradigm for some users, one of working in a rather "sloppy" manner, then choosing a time to ask the system to tidy up the workspace. Was this paradigm a useful one?

Tidying of trees was performed in 48 of the 70 non-trivial sessions. In these 48 sessions, tidying was done with the following frequency:

Number of TidyTree commands in a session

```
  range: 1 command to 41 commands (Session S17R0101)
 median: 4 commands
   mean: 5.7 commands
std dev: 7.2
```

The TidyTree function proved popular among those who used it. Evidence was both anecdotal and in the parser summaries.

On the other hand, the TreeShrink and TreeGrow commands, which I considered powerful functions of *Prose II*, were used in only two sessions, for a total of four times each across all the sessions. TreeShrink and TreeGrow were among the few functions available through the keyboard interface only; they did not have a mouse or menu selection. Thus, to find out about them, users had to read the appropriate help panel or the manual. This may have limited their usage frequency. However, the converse was not true: the five commands that were used even less frequently than these all had mouse/menu interfaces.

5.3.3.7 Comments Collected by the Tracker

How often did subjects use the two "hot keys" to communicate directly with the tracker? In *Prose II*, the F2 function key could be pressed at any time to record a comment in the protocol record; pressing the F3 function key indicated that the subject was taking a break in the session.

10 of the 29 subjects used the "F2" key in *Prose II* to leave comments in their protocol records. A total of 32 comments were left among the 112 sessions. The comments are listed verbatim below, ordered by session ID.

Session	Verbatim comments entered in the tracker
S06R0107	*"This looks like a terrific tool once I get to know it better. One thing is that Windows messes up with the PC Char Set and the ANSI Set doing National Characters. Keep on going John!"*
S06R0110	*"This would be a wonderful tool for keeping minutes of a meeting etc. Especially when a videobeam system were available. Then from a script file a written document would be output."*
S08R0201	*"There doesn't seem to be a way to disassociate filename from node."*
S08R0301	*"May have said this before, would like quick create "button" that would both make a node and open the label text window"*
S10R0104	*"This is the first time I am using Prose II."*
S12R0101	*"trying to find how to preserve ordering of trees after "tidy""*
	"stopping work to send message to author on tidying of trees"
S15R0105	*"I guess i screwed things up by changing file names of text to get my data back..."*
	"System keeps hanging or requires rebooting."
S15R0106	*"I seem to have lost all the text from my previous session..."*
S15R0202	*"Strange things are happening... I added a node and found text from another node in it."*
S17R0101	*"Just returned from lunch (1 hour 45 minutes!)"*
	"Undo would have been nice here"
S18R0103	*"Will continue later"*
S25R0101	*"particularly like using the map window as a way to move around the document"*
	"for brainstorming this reminds of an idea wheel, where you just write the ideas down and worry about the relationships later, very natural"

S26R0201 *"After leaving PROSE2 and restarting, file extension for a node changed."*

"Why are two files created for each node? One without .ext one with."

S27R0104 *"When I go select 'Editor', I'd like to see the node label as a comment in the top line."*

"When I click outside a pulldown menu, I don't want to create a node in the workspace. The workspace should then be inactive."

"'Save' and 'Save as' should confirm the action."

"Outline window should be editable"

"Items in map window should be selectable. If I select the 3rd outside the workspace, that node should be centered in the updated workspace."

"Does not redraw workspace after viewing and sizing map."

"I now have File and Defaults highlighted in the Action Bar! Something odd has happened to the workspace. The arrows are HUGE. Redrawing takes an awful long time."

"Windows messages 'Not enough memory'. Windows falling apart. Will probably have to end session. Colors and borders screwy. Sluggish. Lockouts."

S27R0105 *"Have restarted, with more free RAM (about 1.5MB extra). I have about 6.5MB, but use enormous cache and some 1MB RAM disks under DOS3.3, normally. Different setup for DOS2.01 and OS/2."*

"Closing down for a while to finish shopping, take a break, and a dump."

S27R0106 *"The outline window needs a scroll bar at the bottom, so I can see text that's off the right of the screen."*

S27R0107 *"Now have to stop for the day."*

"Not obvious how to gather the text, which I entered 'behind' each node, into a single document."

S27R0109 *"Something odd: None of the text I enter is saved. Files exist, but length = 0. Giving up and returning to word processor."*

Subjects used the "F3" key to signal explicit pauses in their sessions a total of 17 times across all the sessions.

Time spent in explicit breaks

```
  range:  2.3 seconds to 256 seconds
 median: 27.9 seconds
   mean: 60.8 seconds
std dev: 74.2
```

Discussion:

- Subjects' comments appear to fall into five general classes:

  1. Descriptions of what happened during some elapsed time.
  2. Observations on how they were using the system, or good ways it could be used.
  3. Praise for functions of the system they liked.
  4. Software bugs or lack of understanding of how to accomplish some function; also, requests for additional functions.

5. System or hardware problems.

- Subjects rarely signaled explicit pauses in the tracker using the designated function key. When they did use this mechanism, none of the explicit breaks were longer than 5 minutes; most were less than half a minute.

  Since the Comment function was sometimes used to indicate pauses of a couple of hours, these functions should probably be combined. They should also have a menu interface to be readily accessible.

5.3.3.8 Summary: Commands and Documents

- Documents that were constructed across several sessions were smaller than documents constructed in a single session.
- Most documents had between 15 and 30 nodes. In documents constructed across multiple sessions, the median and mean value for the number of nodes in the documents is close (about 14 nodes), with small variation.
- About 1/8th of all nodes that were created were later deleted. Two-thirds of the deleted nodes were the last node created.
- Subjects spent about 10 to 20 seconds labeling a node. Labels consisted of a few choice words.
- Subjects spent about 1 or 2 minutes writing in a node.
- Most subjects tried running a text editor within *Prose II*, but few used it for extensive writing of a document.
- Help screens were typically viewed for 15 to 30 seconds. Help times were longer in trivial session than in non-trivial sessions.
- Subjects used the TidyTree function frequently.
- Subjects left comments in the tracker for a variety of reasons, which generally fell into five general classes.

5.3.4 What Was Learned About Writing With This System?

This subsection presents a variety of results on some specific aspects of the writing process. For example, how large are the structures that writers construct? When is the main point of a document created? How are nodes laid out spatially when organizing in two dimensions? How do sessions differ, depending on the number of nodes under consideration?

5.3.4.1 How Big are Trees?

Prose II allows users to build immense trees. In actual use, how big did trees grow? As a software designer, what are reasonable data structures to use, based on the expected size of the trees? In studying writers, how many ideas are generated, and to what depth of elaboration are they taken?

One of the previous sets of statistics looked at the number of nodes created in the sessions. Different from that is the number of nodes in the saved trees. For example, a workspace with 40 nodes might be saved, yet the largest tree in the saved workspace has only 5 nodes.

Three different measures of tree size are presented here. The first measure looks at the maximum depth of the saved trees. The second measure looks at the number of nodes in the saved trees. Finally, I looked at the sizes of the trees created in one of the single "CreateNewTrees" episodes.

46 distinct trees were saved among the 70 non-trivial sessions. A tree is defined as having at least two linked nodes. The following table shows the maximum number of levels in these trees.

Table 37. Distribution of maximum tree depths

Maximum number of tree levels	*Frequency*
2	4
3	11
4	10
5	8
6	6
7	2
8	3
9	1
10	1

Maximum number of tree levels

```
  range: 2 to 10 levels (Session S27R0103)
 median: 4 levels
   mean: 4.7 levels
std dev: 1.9
```

How many nodes were in a single tree? This value is often smaller than the number of nodes in a document, since some nodes created in a session may not be linked to others.

Table 38. Distribution of maximum tree sizes

Maximum number of nodes in a tree	*Frequency*
2 to 5	4
6 to 10	11
11 to 15	15
16 to 20	9
21 to 25	2
26 to 30	2
31 to 35	1
36 to 40	1
more than 40	5

Nodes in a single tree

```
  range: 2 to 122 nodes (Session S17R0101)
 median: 14 nodes
   mean: 18.3 nodes
std dev: 18.3
```

One of the constructive episodes, CreateNewTrees, characterized a sequence of commands where a subject built a tree from scratch–by creating nodes and linking them. How big were the trees created in the 96 "CreateNewTrees" episodes?

Nodes created in a CreateNewTrees constructive episode

```
   range: 2 nodes to 12 nodes
  median: 3.0 nodes
    mean: 3.4 nodes
 std dev: 2.0
```

Discussion:

- The distributions of tree depths and tree sizes is similar.
- Small trees differ from large trees in their degree of fanout. There was small variation in the tree depths; there was wide variation in tree sizes.
- The maximum tree depth was 10. *Prose II* was designed to support up to 99 tree levels; this was clearly more than enough. This is a case where the protocol data might be used to solve a design issue: for example, how many internal memory blocks should be set aside to handle tree levels?
- In the constructive episodes where a new tree was created, those trees tended to be small: about 3 to 5 nodes.

5.3.4.2 How Does Order of Creation Correlate with Final Position?

Writer's using this software laid out nodes in two-dimensional trees. How much brainstorming actually took place with this system; were nodes generated in the order they were used, or were they moved and re-ordered until a desired positioning was found?

To determine the relationship between when a node was created and its final tree position, I generated a pre-order walk of each document's final tree structure. The nodes in a session's final tree were numbered according to the order in which they were created. A Pearson correlation was then calculated between the two sets of numbers.

Table 39. Distribution of the ordering correlation values among the 45 documents

Pearson coefficient	*Frequency*
-1.0 to -0.8	0
-0.79 to -0.6	0
-0.59 to -0.4	2
-0.39 to -0.2	2
-0.19 to 0.0	2
0.01 to 0.2	5
0.21 to 0.4	7
0.41 to 0.6	11
0.61 to 0.8	8
0.81 to 1.0	7

Correlation between final position and order of creation

```
   range: -0.53 to 1.0
  median:  0.53
    mean:  0.42
 std dev:  0.39
```

Discussion:

- The results showed strong correlation between the order of node creation and the sequence of nodes found in a pre-order walk of the document tree.
- In 1989, Lansman working in a study with the Textlab group, found correlations among her 17 sessions that ranged from -0.16 to 0.99, with a mean of 0.48. My values, for 45 documents, ranged from -0.53 to 1.0, with a mean of about 0.42. These mean coefficient values are similar; my range may be larger because it had more subjects in less controlled conditions.
- Do longer sessions imply more or less correlation?

  The analysis showed a mild negative correlation between the session times and the preceding coefficient: `r = -0.27`. This suggests that more experience and longer sessions imply more exploration of a tree's structure.
- Sessions with high correlation:

  S25R0301: The subject constructed a tree with 20 nodes and 0.999 correlation between the order of node creation and the tree position. These nodes were not labeled. This is even more unusual, since there were many MoveNode commands.

  S10R0104: In this one case of perfect correlation, the subject left the comment "This is the first time I am using Prose II." Only six nodes were created in this session.
- Pre-ordering was used in this analysis because it is the familiar ordering in Western culture for unraveling a hierarchical document into a linearly sequenced text. A future study might considering examining other orderings, seeing where there are high correlations, and what other factors they may correlate with.

5.3.4.3 Which Node is the Final Root?

When is the main point in a document created? This discussion is motivated by a paper (Bereiter, Burtis, and Scardemelia, 1989) that reported a bimodal distribution of "time until the main point." They found one mode at about 1 minute and the other mode at about 6 minutes.

In 35 documents, complete trees were saved (although 45 documents were saved by the subjects, ten of these involved work on previous documents whose structure was not known by the tracker). For 25 of these, the root of the tree was node number 1. There was no particular pattern among the other 10 documents: 2, 5, 8, 9, 17, 17, 17, 25, 32, 48. For the 10 documents whose root was not the first node, the mean time to creation was 16.8 minutes, with a standard deviation of 10.5 minutes.

Discussion:

- Most subjects knew their main topic before they started.
- Like many other results in this study, the frequency distribution shows an early peak and a long trailing tail.
- The results do not directly support the bimodal distribution reported by Bereiter, Burtis, and Scardemelia. However, they reported on the time until the main point, whereas I looked at which node was the eventual main point. Measuring the time until the first node is created is another test that could readily be added to the parser and examined in a future study.

5.3.4.4 Are Nodes Created Left-to-right and Top-to-bottom?

When subjects create new nodes, where do they tend to position them? After watching my own behavior, my guess was that a new node is generally created below, and to the right of, the last node created. To test this, I considered a simple vector addition scheme of summing the total x and y coordinate values, but realized it would not work, since one oddly-placed node can sway the resulting sum.

I devised a scheme where the parser compared a new node's x and y coordinates with those of the last node created. If the new node's x coordinate exceeded the last node's value, a count was incremented; if it was lower, the count was decremented. A similar count was maintained for the y coordinate. If my conjecture was true, the x-count would be positive and the y-count would be negative, corresponding to moving to the right and down in the Cartesian coordinate system.

For the 27 single-document sessions, the statistics are shown below:

x coordinate count

```
  range: -3 (S20R0101) to 25 (S17R0201)
 median:  3
   mean:  4.0
std dev:  5.2
```

y coordinate count

```
  range: -18 (S22R0101) to 7 (S17R0201)
 median:  -3
   mean:  -4.3
std dev:   5.9
```

Discussion:

- There is a wide variation in node layout patterns (the standard deviation is larger than the mean), but the results still suggest a classic left-to-right and top-to-bottom creation of nodes.
- The X and Y coordinate counts (median, mean, and standard deviation) match up well.
- S17R0201 was unusual; in this session the subject moved strongly up and to the right.

5.3.4.5 Early Planning and Late Writing vs. Alternation?

What proportion of planning preceded writing? Did subjects' writing tend to conform to a two-stage model of writing (*i.e.*, planning first, followed by writing or revising)?

To test this conformance, Lansman (1991) developed an index called the Stage Index. This index was designed to quantify the extent to which planning time preceded the time for writing and revising. In order to understand this index, imagine computing—for every minute of writing time—the proportion of total planning time that preceded that minute of writing. The Stage Index takes an average of these proportions across all the minutes of writing. For example, if a subject completed all planning before beginning to write, then for each minute of writing the proportion of planning that preceded it would be 1.0 and the average, the Stage Index, would be 1.0. The index can vary between close to 0 and 1.0. In practicality, 0.0 and 1.0 are hard to reach, since at least one node must be created before writing occurs, and since pauses can separate the writing periods at the end of a session.

Among the ten sessions with substantial writing (see Figure 33 on page 169), the Stage Index values are shown below:

Stage Index, among 10 sessions

```
  range: 0.40 (S15R0202) to 0.81 (S18R0108)
 median: 0.55
   mean: 0.56
std dev: 0.14
```

Discussion:

- The mean and median Stage Index values tended toward the low value in the range, indicating that planning and writing were generally intermixed in these 10 sessions.
- The standard deviation was relatively low. However, the sample size of ten was small, since few sessions had substantial amounts of writing.
- Lansman, Smith, and Weber (1991) describe Stage Index values for a study with 17 subjects; the values ranged from a minimum of 0.58 to a maximum of 0.98, with a mean of 0.78. In the sessions they observed on the *WE* system, planning appears to precede writing to a much greater extent. This might be explained by the different design of the two systems, where planning is encouraged in distinct windows. These windows lead a user to create and structure their document trees before a text editor becomes readily available.
- In the two sessions at the extremes:

  S15R0202: which had a Stage Index value of 0.40, got that value because several nodes were created and edited early in the session, followed by a pause of 26 minutes before work on the session was completed. Long pauses are not accounted-for well by the Stage Index. Like many other aspects of this study, this index seemed to operate well in controlled, continuous-work environments, but was not robust in some actual settings.

  S18R0108: which had a Stage Index value of 0.81, was unusual in that all the creation, labeling, linking, and movement of the nodes was done before any writing in the nodes. This Stage Index of 0.81 might be as high as can be practically reached using these analysis tools and this software.

5.3.4.6 What Patterns are Seen in the Sessions?

Finally, stepping back and looking at all the productive sessions, were there a set of overall writing schemes that appeared? With the large amount of data available, there were many ways to look for answers to this question. The best answer correlates different patterns with the number of nodes being worked with.

Besides the four categories of non-trivial sessions described in Figure 32 on page 168, the substantial sessions can be grouped by the number of nodes created in that session. Our first conjecture was to divide the sessions by groups of 10 nodes, that is, sessions with 1 to 10 nodes, sessions with 11 to 20 nodes, sessions with 21 to 30 nodes, and so on. Examining all of these sessions, we instead found three natural groupings: those sessions with 10 or fewer nodes, those sessions with 11 to 21 nodes, and those sessions with 22 or more nodes.

Among the features evident in these groupings are some concepts I borrow from the game of chess: the idea of an "opening move," a "middle game," and an "end game." These concepts are discussed below.

Sessions with 10 of Fewer Nodes: There were 13 sessions in this group. For the most part, these sessions look like learning or experimentation by their subjects. For example, in 4 of the sessions, there were no labels or only one label on the nodes. 5 of these sessions involved at least some writing in the nodes.

This was little pattern to the opening moves in these sessions. These sessions were not time-efficient, probably because of 1) the fixed startup and takedown costs, and 2) the ongoing exploration of the *Prose II* features by the subjects.

These sessions were clearly different from the trivial sessions, however. For example, comments were left in the tracker only twice among all these sessions, as opposed to many comments left in the trivial sessions.

Sessions with 11 to 21 Nodes: There were 22 sessions where 11 to 21 nodes were created. As a group, three different patterns can be seen in these sessions.

Brainstorm ideas, then link
Brainstorming was evident in 4 of these sessions. The subjects created and labeled all or most nodes, then linked them together into trees.

Incremental development
In 9 of these sessions, the subjects created and labeled a "critical mass" of nodes, then linked them into a tree. The rest of the session consisted of creating new nodes and incrementally linking them into the existing tree. The tree was frequently tidied (almost compulsively) after each set of new nodes was linked into the tree.

Unstructured exploration
9 of these sessions evinced undirected exploration of the problem: many Roam, Zoom, Move, and BreakLinks commands were interspersed among brief constructive episodes.

An opening move was apparent in many of these sessions: usually 3 to 6 nodes were created in the first constructive episode. Only 2 of these 22 sessions could be considered efficient in the number of commands performed or the elapsed time, given the number of nodes they created.

Sessions with 22 or More Nodes: In 9 of the sessions, 22 or more nodes were created. Most of these sessions start with an opening episode where 6, 7, 8, or 9 nodes are created and labeled. After that, the general pattern was incremental development.

Subjects appeared to be experienced with *Prose II* before tackling these larger documents. This was indicated, for example, by the fact that little help was requested among these sessions; subjects did not tend to tackle large documents without experience with the system. Also, 4 of these 9 sessions were efficient in the number of commands performed or the elapsed time, given the number of nodes they created.

There were a pair of unusual sessions in this group.

S18R0108: In this session, 27 nodes were created, labeled, and linked, then all of them were written in. The document was all laid out and organized before any writing occurred.

S02R0101: In this session, 28 nodes were created and labeled, then all of them were linked into a tree. All the ideas were laid out onto the workspace before they were hierarchically organized.

5.3.4.7 Summary: Planning and Writing

- Trees generally had 3 to 5 levels, consistent with the conventional structuring of documents into chapters, sections, subsections, and paragraphs. No tree had more than 10 levels.
- Most trees had between 15 and 30 nodes. Most saved documents consisted of a single, complete tree.
- A node's order of creation generally correlated with its final pre-order tree position. Nodes created early in a session ended up high in the structure trees; nodes created late were low in the trees. Two-thirds of the time, the first node created was the root of the eventual tree.
- In terms of spatial positioning, a node is generally created below and to the right of the node created before it.
- Planning and writing were generally intermixed. Only one session had all the exploration and planning before any of the writing.
- Sessions showed natural groupings according to how many nodes were operated upon during the sessions. The groupings appeared to reflect the subjects' experience with *Prose II*. Sessions with 22 or more nodes looked similar: a critical mass of nodes were created and linked into a tree, which was incrementally developed. Sessions with 11 to 21 nodes showed three kinds of behavior, possibly because the number of nodes was manageable enough to be manipulated in several ways. As with the sessions with 22 or more nodes, trees were incrementally developed in some of these sessions. In other sessions, all the nodes were created and labeled before any were hierarchically linked. And in the third group, subjects' behavior looked almost turbulent, as they searched for the right ideas and relationships among them. Sessions with 10 or fewer nodes showed little overall patterns.

5.4 A Portrait of a Session

With the information learned in this analysis, we can construct the story of a typical user and session (although there was no session exactly like the one portrayed here).

> The user starts work on a document by launching *Prose II*, opening a new workspace. The session lasts about 35 minutes. About half of the total time is spent in pauses between commands; the pauses are around three seconds or less in duration, although at some point in the session, a long pause of about two minutes is taken.
>
> The session is spent creating a document tree, that is, an outline of the elements of the document being constructed. Most of the commands that are executed are creating, labeling, and linking nodes, although the user writes some extended text in a few of the nodes. The labels are short and to the point; they average four words with seven letters per word. It takes about 15 seconds to type and save each label.
>
> The document tree has about 20 nodes, in a tree four levels deep. The tree is laid out in a top-down manner; the title of the document is the first node created, followed by the nodes in the first chapter, then the nodes in the second chapter, and so on. A few of the nodes are moved, but nodes are generally put in the right place to begin with. The user operates in a manner where they are creating and labeling nodes quickly, then linking them into the growing tree. Several times in the session, the user tidies up the tree. Since it is hard to see all 20 nodes on the screen along with their labels, the user must roam to different corners of the workspace to find more room. This is done about five times before the document tree is complete.
>
> An analysis of the session shows about 120 commands were executed by the user during the session. These fell into natural groupings, with about four commands in each episode.

Most of these episodes consisted of creating solo nodes, editing existing nodes, and linking nodes into existing trees. Over the course of the session, these episodes formed into phases of similar activity. There were about six phases in the session, with about five episodes per phase. Two of these phases were exploration: creating and labeling new nodes and moving them into position. The other phases involved constructing the document tree and touching up its structure. At the end of the session, the tree is saved once, in the Script format favored by the IBM employees who were the users in this study.

CHAPTER 6. OBSERVATIONS ON AUTOMATING PROTOCOL COLLECTION AND ANALYSIS

This project was designed to examine the feasibility of automating both the collection and analysis of protocols. A goal in automating these steps is to allow naturalistic studies of the interactions between humans and computers. Reports of naturalistic studies are important if systems developers and researchers in human-computer interaction are to draw informed conclusions on tasks, strategies, and software usage.

This chapter reports on the costs, problems, and suggested improvements related to automating protocol collection and analysis. In the first section in this chapter, I discuss the issues I encountered collecting and managing the protocol record files in this study. The second section discusses my observations on building and using the parser and protocol analysis tools. The next two sections list the enhancements to the tracker and parser that I recommend for future research efforts.

6.1 Collecting and Managing Protocol Records

How big were the files that were collected? How big were the parse trees generated by the parser? How much data can be expected for this type of a study? These statistics are presented below. The second subsection examines the significant problem of managing the session recordings and analysis data.

6.1.1 Protocol Data Statistics

112 protocol record files were collected in this study. The 112 files together took 1,053,000 bytes, small enough to fit on one high-density personal-computer diskette.

Protocol record file sizes

```
  range:    736 bytes (S21R0101) to 102,428 bytes (S17R0101)
 median:  6,476 bytes
   mean:  9,405 bytes
std dev: 12,889
```

A file about the size of the mean contained 150 lines, including the seven-line header in each file.

Because the tracker captures the secondary commands, I did not assume the size of a protocol record file would correlate with the amount of work done in a session. For example, small moves of the root of a large tree cause many secondary commands to be recorded, making the size of the protocol file out of proportion to the number of commands done. However, the file size and number of commands were surprisingly well correlated: `r = 0.967`.

With a mean session time of 1447 seconds, the tracker produced an average of about 6.5 bytes of protocol data per second, which is 23,400 bytes per hour. This is about 373 lines of protocol data per hour, or between 6 and 7 pages of protocol record per hour.

The parser produced parse tree files that were about 30% smaller than the protocol record files. The total size of all 112 parse tree files was 704.820 bytes. Including the intermediate files produced by passes 0, 1, 3, and 4, the parser produced 560 files, with a total size of 3,290,154 bytes—about three times the size of the protocol records. The range of parse tree sizes is shown in Figure 30 on page 130.

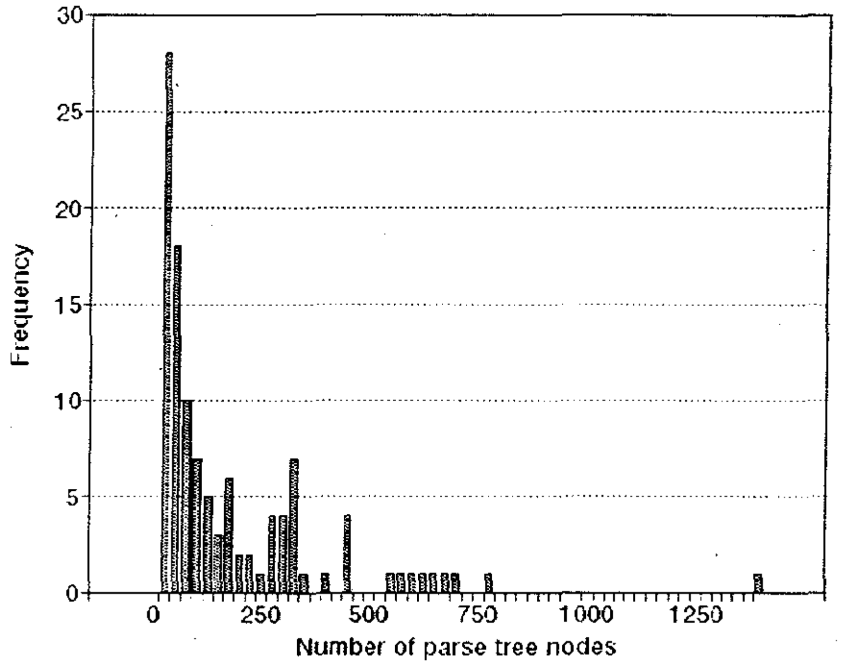


Figure 30. Frequency distribution of parse tree sizes across 112 sessions. This chart shows the number of nodes in the parse trees generated by the parser.

The automated parse generated 112 parse trees, with the following size characteristics:

Parse tree sizes

```
  range:  7 nodes (S21R0101) to 1398 nodes (S17R0101)
 median: 112 nodes
   mean: 166 nodes
std dev: 212
```

On a 16Mhz 80386 personal computer, it took 35 minutes to run the parser against the 112 protocol record files. Thus, it took about 20 seconds to parse a single protocol record.

6.1.2 Managing Session Recordings

Soon after I sent the *Prose II* package to the potential subjects, I began receiving their protocol files on the network. The first unanticipated problem was the naming convention for the files. I implemented *Prose II* to use a *Windows*-specific call to generate a unique name for each recording file on the machine it was being run on. These names look like `~RCD1300.TMP` in DOS. I did not anticipate the name conflict among files from different subjects. When uploaded to an IBM VM host, the initial tilde was not accepted as a valid character. Subjects took this as an opportunity to rename their files or to combine them into larger packages. When I received the files, I had to carefully account for each file by each subject; the filenames I received differed from the filename strings the tracker had recorded in the header of each protocol record.

I quickly settled on a naming convention that identified each subject, document by that subject, and session for that document. For example, the session entitled S08R0201 indicates subject number 08, document number 02, session number 01. This naming convention was implemented by looking inside each protocol file. I determined not only the chronological sequence of files received from a subject (by looking at the date and timestamp), but whether this session contained work on a new workspace or on an existing workspace. My quick evaluations were sometimes later proven wrong; also, they did not account for trivial sessions and sessions consisting of work on multiple documents. Nonetheless, this naming convention proved stable. I kept the filenames throughout the study, as shown in Table 40 on page 164.

This hand-assignment of names was part of a major issue uncovered in this study: the management of session recordings. Because a single session could be efficiently analyzed, I did not anticipate a new class of problems: management of the massive amounts of raw and processed data. All this data needed unique names and some type of attributes to identify it. The 112 sessions in this study were managed by hand. Thus, the groupings of attributes such as subjects, documents, and sessions, had to be done by hand. Scaling up to thousands of sessions will require more sophisticated tools.

For example, I produced Table 40 on page 164 and Figure 32 on page 168 by hand. Over the months of analysis, I found numerous transcription errors in these despite the care I took. Finding a bookkeeping error (such as missing a session) involved the painstaking itemization of all 112 sessions.

The sequencing of the parser passes was automated using the software utility *Make*. I constructed each *Make* input file by hand. The *Make* files were rarely touched once I felt they were right, implying a loss of flexibility.

Another unanticipated problem was the interweaving of subjects, documents, and sessions.

- Subjects worked on documents that spanned one or more sessions.
- Subjects worked in sessions that included several documents.
  - Some subjects worked on a single document during several sessions in a day. Other subjects left their *Prose II* window active but unused for long periods of time. Could the analysis account better for the difference between a very long pause (*e.g.*, an hour), and exiting the system and entering it again within a short time period?
  - Some subjects worked with *Prose II* for a while, exited it, worked on the document with a real text editor for a few days, then came back and used *Prose II* for a session with the updated document. How should the analysis account for the discontinuity in the work on a document?

- Some subjects revised documents that had already been written using a different text editor. *Prose II* proved amenable to this sort of work. How should the nodes in such a document be accounted for?
- It did not appear that multiple people shared work on a session or a document, but that situation could occur often in many environments. The current collection techniques were not prepared for multiple sequential or concurrent authors.

6.2 Automating Protocol Analysis

The parser and associated analysis tools developed in this project proved suitable for generating answers to the many questions discussed in Chapter 5. The questions, answers, and statistics presented in the previous chapter explored a wide range of areas, providing a composite view of this group of subjects, how they spent their time with this software, and what kinds structures they worked with. At the outset of the project, a key research question was "What kinds of questions is the protocol analysis designed to answer"?

The parse output provides researchers with the same kind of information that it provided Card, Moran, and Newell in their *ICARUS* study: a guide to how time was spent in a session with a software system. Besides segmenting a session and characterizing the sequences of commands, the parser also produced a set of statistics, summarizing many aspects of the session. An example of a session summary is shown in Figure 24 on page 90.

6.2.1 Devising and Testing the Parser

The parser described in Chapter 4 evolved from the protocol analysis described by Card, Moran, and Newell in their *ICARUS* study. We designed each pass of the parser by first taking protocol records and analyzing them by hand. How should a given sequence of commands be parsed? We came up with consistent answers in our manual analysis, then had to find a way to generate the same answers with the parser.

I first completed Pass 0 (converting from the protocol record text to computer-oriented 16-tuples), since its success involved a close marriage between the syntax analyzer and the tracker. I worked next on Pass 2 (characterization of housekeeping episodes), since its success was necessary to define the constructive episodes, which are the crux of the analysis. Pass 3 (characterization of constructive episodes) evolved over a year. It requires a great deal of context information, and a lot of experimentation was required to determine just what that information was and how it should be carried. Pass 4 (characterization of phases) also had a long gestation period, since I was trying to understand exactly what the output symbols should be and what sequences of episodes determine each phase. The collection of session statistics in Pass 5 is a process that continued throughout the project. Finally, several inconsistencies caused problems that had to be solved somewhere; Pass 1 resulted, late in the project.

The parser describes how time was spent in a session, in addition to looking at the products produced over time. A helpful enhancement would be a fool-proof way to identify the different groups of session types (as shown in Figure 32 on page 168), particularly trivial sessions. Card, Moran, and Newell did not have to deal with these groupings in their single analysis. With about 100 sessions, the categorization could be accomplished by hand. With more session data, however, the categorization should be automated.

The way that this parser was developed and refined depends on a human analyst examining the input and deciding how well each of the various output sequences describe the input. People must do the learning and convert the learning into changes in the parser. It may be that refining the parser is a job well suited for technologies such as neural networks or expert systems. These technologies were not used in this study because they generally provide little information about their internal operation. However, in future automated protocol analysis software, I would consider using a technology that decreases the amount of human time needed to construct the parser.

6.2.2 Key Decisions

What key decisions were made in formulating the grammar and parser? What hypotheses about cognitive processes are built into the parser?

To use pauses to separate identifiable episodes.

I chose to depart from the Textlab version of "modes" in favor of more detailed analysis in terms of episodes. Thus, the use of pauses in this parser contrasts with the *WE* system, where modes are associated with specific windows in the workspace. In *WE*, changes in modes are distinguished by changes in which window a subject is working.

Having chosen to use pauses to separate periods of activity, I proposed that the periods of activity between these pauses could be identified and reasonably characterized.

To separate subjects' activities into two groups: housekeeping episodes and constructive episodes.

In this grammar I proposed two types of episodes. The episodes occur sequentially, not simultaneously.

To assume that cohesive periods of cognitive work are ended when a person pauses or does a housekeeping operation.

I chose to delimit each constructive episode by the periods of housekeeping or long pauses that precede and follow it. This software system is designed to allow a human to perform useful work, not necessarily to spend their time doing housekeeping. Stated another way, in the periods between this housekeeping and long pauses, constructive work is accomplished.

To allow housekeeping episodes to occur in sequences.

The long pauses and periods of housekeeping can be of indefinite length. Identifiably different types of activity can occur during those periods.

To form composite descriptions for some constructive episodes.

During the periods of work, a human might pursue multiple, simultaneous goals. These can be complexly interwoven, yet executed in rapid succession. The characterizations of the constructive episodes reflect this complexity, with their multi-part descriptions.

To characterize episodes and phases with a small group of names.

I proposed that the set of identifiable episodes and phases could be small enough that they can be easily explained to other humans, that they correspond to known type of cognitive tasks, and that they can be recognized and characterized by an automated parser.

To rewrite the internal representation for dense state machines.

Dense state machine are difficult to maintain and comprehend. The state matrices I originally developed for the parser passes were easy to understand when they were sparse (*i.e.*, when, for a given state or input, there were few transitions to other states). They soon developed paths through them that were difficult to anticipate and visualize. Re-writing them using YACC source was helpful in clarifying the sequences. In doing this, many parallel sequences were found and described as such; several sequences clearly in error were found.

To vary the grammar powers among the passes of the parser.

For any task, you generally want to use the simplest tool with sufficient power to complete the task and not waste the resources of the human user. For each pass of the parser, I used what I considered the simplest technique with sufficient power to characterize the sequence under consideration--all within the framework of an ATN. Breaking down the parser into smaller components made its design, implementation, and incremental refinement more modular and intellectually manageable. Section 4.2.2 on page 61 describes the power of each pass and how the passes were used.

6.2.3 Parser Robustness

How robust is the parser? Did the parser converge? How is this measured?

The parser is constructed to handle any syntactically-valid input string. Therefore, the question of robustness became more a question of looking for weak and wrong characterizations of command sequences. In contrast, anyone can create a syntactically invalid input file by deleting a line or sorting the lines in a protocol record file. The parser does not make any attempt to handle an invalid file. One of the 112 files from the study was invalid, since the subject had attached a paragraph of text to the bottom of the ASCII file. Pass 0 of the parser failed, and I fixed the file by hand.

When this study was originally designed, our plan was to parse 20% of the recordings, fix any problems, then try on the next 20% of the recordings. Presumably, fewer problems would be found if the parser was converging. How many of the session protocols could not be parsed with the initial grammar? Could we characterize the problem in terms of what changes had to be made in order to parse all the protocol records?

That approach works only if all the recordings are homogenous in character, however. For example, only a few of the recordings include the Scramble command; problems with that command would only be evident in the recordings that contained it. Also, by automating the parser operation using *Make* it was as easy to process ALL the files as a group as it was to parse just one.

I had the parser tested and operating for more than a year before I began this study. For example, I showed its operation at the *Hypertext '87* conference in November 1987. I assumed that no further changes would be made to the parser by March 1989 when all the 112 protocol records had been collected. I was wrong!

The following list describes the bugs and modifications made to the parser since the end of the protocol collection period.

Date	**Description**
3/7/89	Added missing Format parameter for SaveWorkspace in the Pass 0 YACC input file.
3/7/89	Corrected the state transition for OutlineWindow when in state 1 of Pass 2.
3/7/89	Added handling for the BreakAllLinks and Scramble commands, missing from Passes 2 and 3 altogether.
3/7/89	Added handling for the ChangeDefault command, missing from the Pass 0 YACC input file.
3/9/89	Added handling for the BreakAllLinks command, missing from Pass 0 YACC input file.
3/9/89	Corrected the state transition for OutlineWindow when in state 5 of Pass 2.
3/13/89	Corrected the parameters for the CopyNode command. CopyNode was expecting a NodeID and a set of coordinates in the Pass 0 YACC input file. Instead, the ID of the source node and the newly-created node ID are the parameters.

3/13/89	Corrected the parameters for the CopyNode command. CopyNode lists ParentID(0) in the recording. This was changed after the system was first shipped, because of a bug. Both forms are now accepted.
3/13/89	Modified input symbols by hand, changing them to an uppercase X (so they could later be distinguished from the lowercase x used in all labels). LEX rules did not handle symbols above ASCII 127, which includes characters from foreign languages, such as Greek and Finnish.
3/14/89	Updated a few older recordings to match the latest command names and parameters.
3/14/89	Generated the end of the text parameter by hand in some protocol records that I prematurely clipped. This occurred for three labels in S19R0101, and for 2 comments. The IBM Personal Editor II text editor clips text strings longer than 255 characters.
3/14/89	Removed the note left by Subject 20 in the bottom of the ASCII recording file.
3/14/89	Updated the Pass 0 YACC input file. It had always converted MapWindowZoom to MapWindowRoam.
3/16/89	Added missing TreeGrow and TreeShrink commands to from the Pass 2 state table.
3/17/89	Corrected the handling of Canceled and Invalid commands in Pass 3.
3/17/89	Added the missing input symbol META_COMMENT to Pass 4.
3/18/89	Separated OutlineWindow into OutlineWindowOpen and OutlineWindowClose (analogous to the MapWindow commands) in Pass 2.
3/18/89	Corrected an invalid state transition in Pass 2. MainWindowReset went to state 2 from state 4 (should go to state 5).
3/19/89	Integrated the handling of very long pauses into Pass 2.
5/13/89	Converted Pass 2 from a hand-tailored state machine coded in *C* to a YACC input file with a lexical analyzer.
5/14/89	Corrected the handling in Pass 3 that occurred when nodes were deleted because of starting over.
5/14/89	Modified the output of all passes to allow it to be directly imported into any DOS spreadsheet.
5/14/89	Added tracking for the accumulation of the "number of commands" to the output of all passes.
5/14/89	Fixed the Pass 4 FSM to correct the output for Document Revision (was being bundled under "Top Down Construction").
5/16/89	Modified Pass 3 and Pass 4 output to produce single characters—to simplify spreadsheet graphs.
5/16/89	Modified Pass 5 to produce IBM BookMaster-compatible output. This made it easier to print all of the parse trees on a mainframe laser printer.
6/19/89	Modified the Pass 3 to output the workspace tree each time a SaveWorkspace command is done.
6/26/89	Added Pass 1 to handle its unique cases: 1. Maps the following into the single symbol: EditNode. The subject has actually brought up an editor, which caused an exit from *Prose II* proper.

- EditNode
- LeaveProseII

2. Map the following into the single symbol: TidyWorkspace.

   - SetTidyModeOn
   - pause
   - SetTidyModeOff

3. Maps the following into the single symbol: DeleteNode.

   - SetDeleteModeOn
   - pause
   - DeleteNode
   - pause
   - SetDeleteModeOff

These sequences were by-products of using modes and how the system interacted with other windows. Updated Pass 5 to read symbols from Pass 1.

6/26/89 Modified the Pass 3 output of workspace trees—to output them in a standard spreadsheet format. This allowed easy running of the Pearson correlation, to correlate the order of node creation to the node's position in the final tree.

7/9/89 Fixed bugs in the summary pass caused by looking at the flags of Pass 3 output symbols. Rebuilt the summary output to be much more extensive: it now has 30 summary fields. There are still some fields that will need to be added by hand, such as average label length.

7/26/89 Added an additional check to Pass 1. Some editors caused the ordering of the commands to be reversed. Pass 1 now correctly maps the following commands into the single symbol: EditNode.

- LeaveProseII
- EditNode

8/7/89 Cleaned up the file names and *Make* files to clearly identify the passes to the uninitiated.

8/18/89 Fixed Pass 1 to recover correctly from concatenated sequences.

8/18/89 Adjusted Pass 3, so CreateNode and DeleteNode in a constructive episode maps to the Unproductive Work symbol.

11/12/89 Added a field to the Pass 5 summary information: total pause time.

12/23/89 Added a field to the Pass 5 summary information: Stage Index.

6.3 Tools Used in the Protocol Analysis

Besides the basic six parser passes, several other tools and techniques were used to further automate the analysis and to make it faster and simpler. These are discussed in this section.

I used the following software as the base environment for the parser development and protocol analysis:

- DOS 3.3 operating system
- FileCommand 3 DOS shell
- Personal Editor II 1.01b text editor

- GSEE general filename searcher

I used the following software tools to develop the parser:

- Microsoft *C* compiler 5.1
- Microsoft *Make* 4.07 program maintenance utility
- LEX
- YACC

I used the following software tools for the protocol analysis:

- Quattro 1.0 spreadsheet
- MYSTAT statistical analysis package
- Ready! 1.0 outline processor
- GFIND general string searcher

6.3.1 Helpful Tools and Techniques

The following tools and techniques were particularly helpful for developing the parser and doing the protocol analysis:

Use ERRORLEVEL to control program interaction.

DOS (and other operating systems) allow programs to return an integer value to their callers upon exit. In DOS, this is known as the ERRORLEVEL. I defined unique ERRORLEVEL values for each of the fatal errors in each of the passes of the parser. These allowed for easy identification of an error during parser development.

Use stdin and stdout for file input and output.

The generic file streams available in DOS (and other operating systems) made the connection of the parser passes simple and easy to test. The output of one pass was easily piped into the input of the next. For testing and collection of statistics, the output could be sent to a file and examined later. These proved good object-oriented tools, as espoused by Cox (1986).

The liabilities of using **stdin** and **stdout** are two: no simple checkpointing and no passing of complex data structures between programs. These did not pose any problem in this study. Checkpointing might be a necessity in a future system that does online parsing. Complex data structures were accommodated by using a single template to read the input and write the output for Passes 1 through 4.

Save Intermediate files in spreadsheet format.

I initially invented a unique format for the intermediate parse files, but later realized the many benefits of being able to look at the output of any pass or any session summary using a standard DOS spreadsheet:

- Easy to examine proportions, for example, the proportion of total session time spent in pauses.
- Easy to do frequency analysis.
- Easy to calculate mean, standard deviation, median.
- Easy to replicate complex equations.
- Easy to break data into groups, for example, all sessions with a non-zero Stage Index.
- Easy to sort large body of data with several keys simultaneously.
- Easy to produce and tailor graphs.

I used the *Quattro* spreadsheet, which is compatible with the popular *Lotus 1-2-3*.

Use *Make* to automate parsing

The parsing process for all the passes and all the sessions is driven by a single *Make* input file. This completely automated the job of analyzing a large number of sessions. Some advantages of using *Make* to automate the task of parsing are:

- Error checking is simplified. The parser stopped if any error is found in any parsing pass. The returned `ERRORLEVEL` value is displayed when *Make* completes.
- It is easy to run a parse against a subset of sessions. For example, a certain factor could be generated in all of the "trivial" sessions. Once the set of trivial sessions was determined, they could all be put in one Makefile, and only the programs run against them need be changed.
- Consistency is guaranteed across all sessions being analyzed: "Apply this operation to all recordings." The same set of programs and options was run identically against all sessions in a Makefile. Out-of-date analyses were overwritten. Whenever a modification is made to one of the parser passes, it is easy to test it.
- It was simple to start it up and let it run to completion.
- It was simple to stop a parse at any time, and then to later resume right where it left off.

The liability of using *Make* was that the Makefiles had to be created and maintained by hand. As always, any time I worked with such files, I introduced human errors. For example, I would accidentally duplicate a filename in one place, but not in another. These errors often took a long time to track down.

6.3.2 Tool Limitations and Problems

I encountered the following limitations with the tools I was using:

- The DOS version of *Make* shipped with the Microsoft C compiler could not handle more than 70 files. With 112 recording files, this implied breaking the protocol record files into two groups in order to parse all of them. Since Makefiles were constructed by hand, this was another possible source for errors.
- The *Ready!* outline processor would not handle outlines or trees with more than 700 records or 77 characters per line. This limited its useful for looking at parse trees—although only two of the parse trees had more than 700 nodes, many labels were longer than 77 characters.
- A parse of all 112 sessions produced about 800 files. While the DOS shell I was using, FileCommand 3, could hold all 800 file directives in its memory simultaneously, it did not have enough memory left to run the *Quattro* spreadsheet.
- The SORT command in DOS can only sort files whose size is less than 64 Kbytes. This was not a significant problem whenever the data was in a format that could be used as input to a spreadsheet, since spreadsheets can sort files larger than 64 Kbytes.
- The *Quattro* spreadsheet sorts spreadsheet data with a maximum of 5 sort keys.
- The *Quattro* spreadsheet accommodates 8192 rows. The study included 7370 pauses, which were sorted for analysis and looked at in various ways. So, although this was not a limitation encountered in this study, a study that was slightly larger would have needed other tools to look at a list of this approximate size.
- *Quattro* spreadsheet graphs are limited to a maximum of 175 items on the x-axis. I used *Quattro* to look at the time duration of episodes in a session; this restriction caused a problem when there were more than 175 episodes in a session.

- Scanners generated by LEX restrict their input character set to be the seven-bit ASCII symbols, numbered in decimal from 0 to 127. The scanner used for Pass 0 of the parser balked with labels entered by subjects in Greece and Finland.
- The IBM Personal Editor II text editor could only edit lines with 255 or fewer characters. I truncated a few long labels and comments unexpectedly.

6.4 Enhancements to the Tracker

While using the tracker and its generated protocol recordings, I compiled this list of future enhancements that could improve its generality and usefulness.

1. Add checkpointing of protocol record files.

   If a session is ended abruptly by re-booting or powering-off the machine, the entire protocol record for that session is lost. By adding periodic checkpointing, a significant portion of the protocol record could be saved.

   I did not plan for this in the design of the tracker, but did not receive any reports of whether it would have been helpful to any of the subjects. I did encounter this situation myself during a power failure, and realized the value of preserving as much as possible of each protocol record.

   The checkpointing could be done periodically, or it could be done during a long pause—the range of which has been suggested in this study (*e.g.*, see Section 5.3.1.4 on page 98).

2. Have the tracker ship sessions and protocol records directly to the researchers, where desirable and possible.

   In some network environments, users of this kind of protocol collection system could eliminate the step of uploading and sending the protocol records over the network. Starting a session could open a communications link directly with the computer collecting the protocols, and the records could be written directly to the target (collecting) computer. Entire screen images could also be sent over communications lines.

   There are some problems with this kind of approach, however. This raises again the issue of user confidence: if the protocol is not visible, the user does not know what kinds of secret data might be being collect. The software also needs to handle the occurrence of line outages and communications failures; if the network or protocol collection fails, the primary software should continue uninterrupted.

3. Handle indefinitely long pauses better.

   The current tracker records nothing longer than one hour, since there is no hour (or day, week, or month) field in the recordings. Pauses could conceivably last for many hours or days. In actual use studies such as these, machines might often be left on overnight. Card, Moran, and Newell showed only minutes and seconds in their protocol records; significantly longer periods should also be unambiguously recorded

4. Capture additional system information in the header of each protocol record (*e.g.*, screen size, CPU speed).

   With the state of personal computer technology at the time of this study, I guess that most users had display screens with resolutions of 640x350 or 640x480 pixels. While *Prose II* will work with any screen resolution, it cries out for a larger/denser screen. I personally used a display of 1280x800 pixels, which felt about right. Most subjects were somewhat crippled by smaller screens, I suspect, indicated by the fact that 25% of the housekeeping episodes were Refocus.

For example, the tracker could have found the screen size in pixels; this would have been helpful in determining how many nodes were present on the screen and how much cost was involved in roaming and zooming. It would also help in targeting GUI software to computers with screen real estate of the most prevalent size. If an overwhelming number of target users have screens that are 500x500, writing software that works best on a 1000x1000 pixel screen may be counterproductive.

Additional strategy questions, such as, "Is tree size correlated with screen size?" could have also been answered.

5. Add feedback options when exiting help panels.

   When viewing a help panel, a user is at an "interrupt"; they have stepped out of the context of their work to understand more about the tool at hand. With a simple modification to the tracker, we could could identify help panels that are not sufficiently helpful. Currently, users press an "OK" button when they have finished reading a help panel. Instead, they could press one of two buttons to exit: "This was helpful," or "This was not helpful." The tracker could collect these responses, allowing the parser to find the degree of usefulness for all the help panels in a system.

   This could obviously be extended with various degrees of obtrusiveness: instead of two buttons, there could be a sliding scale. Also, whenever "not helpful" is selected, the tracker could request additional information from the user.

6. Add a word count to the parameters recorded by the EditNode command.

   When tracking the information for the EditNode command—which records the time spent writing with a text editor—the tracker should also record a word count for the file that was edited. This would allow comparison between the number of words in labels and the number of words in the associated text.

6.5 Enhancements to the Analysis Tools

I compiled my longest list of enhancements in the area of improving and simplifying the analysis of protocols. While this study advanced the state-of-the-art in this area, it is still not clear how to answer best questions like "Tell me what user X did with this software," and so on. Many suggestions are offered here, collected with the help of many researchers who saw or used these analysis tools.

1. Add additional visualization and graphical aids to analysis.

   For example, the visual playback ability of *WE* lets a human analyst get a better feel for what occurred in a session by actually seeing each step executed by a user.

2. Do parsing in real-time, so it is available as a tool to the user.

3. Allow human intervention to guide or alter the parse.

   The parser works without intervention, making assumptions about what occurred during a sessions. An enhancement would allow a human analyst to mark landmarks in a session, to be accommodated by the parser. One set of landmarks would help in resolving the ambiguity among multi-document sessions and multi-session documents.

   Gordon Ferguson of the Textlab suggested the idea of a parse-tree editor. It provides a form of guided parsing: here is the string, now what kind of parse tree do I want to see for this? Given that, now go about defining the rules.

   In addition to comments by analysts, techniques are needed to annotate the machine-recorded protocols and the parse trees: what do people say that they are doing? This helps get at the issue of verification. The subject can help verify and adjust the correctness of the resulting analysis.

4. Consider formulating a standard protocol record format.

   Is there some base format that would allow protocols to be traded among "application-aware" tracking systems? If possible, we could convert these protocols to a format usable by the Textlab group, to be analyzed by their cognitive grammar.

5. Redesign the parser to recognize the four different categories of sessions, in addition to trivial sessions.

   Culling the many trivial sessions was unexpected and labor-intensive. Especially with trivial sessions, it is important that when this is automated, there are no false positive or false negatives.

6. Modify the parser to avoid skewing of results because of varying time durations.

   The parser should find something interesting to do with command durations that fall out of the bounds of reason. For example, S10R0104 contained an EditLabel command that lasted 48 minutes. Although the subject clearly stopped the task they were involved in, this lengthy period was not accounted for as a long pause or writing in the node. This skewed the overall times in this session for constructive episodes, phases, time per node, and so on. The same thing could happen with Help, for example.

7. Handle the parser's key decisions about pause durations with a sliding scale.

   A more sophisticated parser can be envisioned where the meaning assigned to pauses of different durations could be adjusted on a sliding scale—as opposed to making decisions based on fixed points as was done here (see Section "Pauses" on page 74). To determine the classification of each pause, this sliding scale could use as its input history about the subject, their experience, and the elapsed time so far in a session. This technique might also be used to identify individual differences in attention spans.

   As a topic for follow-up research, it would be interesting to see how the mean, median, and standard deviation values for "commands per episode" and "episodes per phase" changed as the pause duration values were changed in the parser (see Table 32 on page 109).

8. Avoid ambiguity in distinguishing among long pauses and consecutive sessions.

   The parser and overall analysis needs to handle better the issue of the difference between a long pause (*e.g.*, an hour), and a subject who simply ends a session and then comes back in an hour and continues where they left off. Perhaps a future preprocessor pass of the parse might review sets of protocol records from a subject and determine if they should be concatenated. Similarly, if a pause is sufficiently long (*e.g.*, a day), a single session should be divided into multiple pieces, because significantly different cognitive activities may have taken place between the use of the system. Finally, it may be reasonable to divide sessions where a subject works on several different documents sequentially (*i.e.*, multi-document sessions).

9. Modify the Stage Index definition to handle some typical session-duration situations.

   The Stage Index, as devised by Lansman, was designed to quantify the extent to which planning time preceded the time for writing and revising. (The Stage Index is described in detail in Section 5.3.4.5 on page 124.) It gave a suitable indication of the intermixing between planning and writing in controlled studies. However, in the varied behaviors seen in the protocols associated with this study, it needs modification to better handle indefinitely long pauses, multiple-session documents, multiple-document sessions, and so on.

10. Account for the Power Law of Practice.

    Prominent in the second chapter of Card, Moran, and Newell (1983) is the Power Law of Practice: "The time to do a task decreases with practice." With a well-designed user interface, this behavior should be observable over time, with practice. With the tracker implemented here, this could be observed with longitudinal studies, with users who submit many protocol records over time. This behavior could not be observed in this study, because the numbers were too small; few subjects submitted multiple protocols, with little elapsed time.

6.6 Summary

The richness of results documented in Chapter 5 speaks for the feasibility of automating protocol collection and analysis. The parser was costly to develop, and many iterative refinements to it are indicated. However, once it was debugged, it provided useful and reliable analyses of the protocols. This analysis was truly automated, including construction of parse trees, summaries, and direct input into spreadsheets. In fact, the amount of data handled by the parser led to constraints in the accompanying software utilities.

When we designed this project, I assumed that the only required tools to automate protocol analysis would be the new parser. The task of protocol collection and analysis evolved into a much richer process. Database management, spreadsheets, statistical analysis software, tree editors, make, and file comparison utilities also figured prominently in constructing an overall picture of the users and sessions. The diversity of these tools, combined with the value of their interoperation, leads to the need for a protocol analysts' workbench. Such a suite of tools is discussed in the "Future Research" section of Chapter 8.

CHAPTER 7. OBSERVATIONS ON SOFTWARE BUILDING

We had two intentions for the parser and analysis tools constructed for the project. First, they should help illuminate how the software is used in actual field settings. These tools should presumably offer an effective way to do such studies. Second, the analysis should provide feedback on the design of the software system itself, as a way to guide in its further refinement. This chapter is devoted to the second point.

The following pages first discuss general lessons learned in this software engineering exercise. Specific feedback, in terms of change orders to the testbed software, are discussed next; these changes were indicated by the protocol analysis. This chapter closes with a list of additional modifications to the testbed system that were collected during its implementation and usage.

7.1 General Software Engineering Lessons

This section could be viewed as a general overview of some classic gems of sound software engineering. They are stated in the context of the issues that arose in designing and implementing the 25,000 lines of code that comprise the software used in this project.

Develop complex software incrementally.

Throughout the design and development of *Prose II*, its tracker, and the parsing tools, I always had a running system. This afforded me the following advantages:

- Problem determination and debugging was simpler, since the system was grown from the original *PROSE* system and a working demo shipped with the Microsoft *Windows* toolkit. Bugs I found were almost always in code that had been recently added, and I could go back to the last version and see my changes.

- I always had a system in a condition where I could give a demonstration. People who saw the system, even in its earliest stages, could easily give me feedback on the tangible system they had seen and manipulated, not on a document of specifications. For example, early in its development it was stable enough to run experiments on to study navigation techniques (Beard and Walker, 1989).

- Modifications to the system were easily staged. These were normally isolated to the addition of new function, although a full re-write of the internal memory management was done once.

- It was easy to prototype new functions inside the framework of a system that was already running. I invented several new tree-positioning algorithms because I always had a testbed system within which to test them (Walker 1989).

- This was an experimental system. I did not know ahead of time exactly what the result would look like. It continues to evolve, with new features I never foresaw when the project started.

Use the tools you are developing.

The parse trees that were generated by the parser were in a format that could be used by *Prose II*. This let me read in parse trees and manipulate them. The large size of some of the trees turned out to be a good driving problem: could *Prose II* display and manipulate the thousands of nodes in the parse trees with reasonable response time?

As another example, whenever I was using *Prose II* myself as an editor, I generally kept a second *Prose II* session active, to keep track of bugs I was finding or other gripes I had. *Prose II* is well-suited as a place to just stick unrelated ideas. The ideas can later be grouped and organized--in this case, fixes to the system could be staged.

Plan for international character translation.

I did not explicitly plan for national language support as I was building the system, and hence made some assumptions about the characters and character sets being used. This was not a problem with the *Windows* portion of the system, but did cause a problem in the tracker and analysis tools when *Prose II* was distributed worldwide. The lexical analyzer I used dealt with the one-byte ASCII characters represented numerically by 0 to 127. I did not anticipate that characters out of this range would be entered in node labels or search strings, since the US English version of Microsoft *Windows* I was using filters these symbols as part of its function. However, subjects in Finland and Greece submitted protocol records that contained characters that were out of this range, causing the lexical analyzer to fail.

Plan to throw one away.

This is a favorite suggestion from *The Mythical Man-Month* (Brooks, 1968). *Prose II* was a second system. Every line of the original, character-based *PROSE* was touched in my original port from the VAX to the PC. This let me know all the good and bad points in its implementation. Then I started over from scratch with *Prose II*.

In *The Mythical Man-Month*, Brooks notes problems with second systems, particularly with system architects attempting to exceed their design constraints. I encountered a few problems with *Prose II* as a second system: it included some commands that were fun to implement, but clearly not of value its users. However, *Prose II* was implemented from a strong research base, and it was complimentary with a pair of concurrent development efforts, *Storyspace* and *WE*, which somewhat limited the unusual inventions that were incorporated.

Nonetheless, the protocol analysis showed several commands that were rarely used. Architectural fluff?

Use the power of computer networking.

Networking played a key role in this study, in much the way it did for Mead and Conway in the development of their VLSI book (1980).

- I was an early developer of a Microsoft *Windows* application, before there were adequate published examples or books on the topic. Much interaction with Microsoft and the *Windows* developers themselves took place over the GEnie network, where Microsoft supported a bulletin board service. This helped me a great deal in learning how to solve many problems in *Windows* programming.

- *Prose II* was distributed for a year via IBM's internal network before protocols were solicited. This allowed me to fix bugs and make improvements based on users' feedback over the network. I was confident the system was stable when I commenced the study.

- The study was distributed to subjects in their actual field setting via the network. The protocols were returned to me over the network.

Automate testcases for visual environments.

When developing non-interactive applications, it is easy to amass a large set of regression testcases, to be run whenever fixes or changes are made. This is much harder when programming in a graphical, interactive environment, where the correctness of the program must be painstakingly verified by a human entering sequences and watching the display.

I did not build any automated testcases to test the visual environment. I wish I had. To provide any kind of a regression test, I had to manually execute all commands in the visual language, including invalid and canceled commands. I did not explicitly do this with every version. One problem this caused is that some items in the list of recorded actions were overlooked by the tracker, and were not captured.

Performance counts.

Much of the acceptance of *Prose II* may be attributed to its fast performance in a graphical environment. Many users commented positively on its speed. Early in its development, I identified the key sections of code that were performance-critical, and prominently labeled them in the source code. I spent much continued effort on making those sections as fast as possible.

The tracker implementation was simpler because it was able to operate synchronously. This was possible only because of the fast performance; having the tracker always active did not cause a perceptible slowdown in users' commands.

Buy, rather than build.

This is another favorite maxim, from Brooks' "No Silver Bullet" article (1987). Large savings on this project resulted from the use of a spreadsheet in the protocol analysis. Once I had modified the parser to produce spreadsheet-compatible output, any questions concerning frequency and correlation could be reasonably asked and easily answered. A separate statistical analysis package, which read input in the same format, assisted in the more-complex queries. Customized programs did not have to be written to analyze the parser output.

Other significant savings was provided by the *Make* utility, which was used to automate the operation of the individual parser passes and to manage the groups of protocol records.

Have an early scout blaze the trail.

The interchange between me and the Textlab group was a good one. I came up with ideas that they implemented and vice versa. By allowing diverse teams to explore the same set of problems, we found interesting solutions that could be quickly included in the current running systems.

For example, *WE* and *Storyspace* both employ the tree-positioning algorithm I first developed for *Prose II*. The researchers on the *WE* project have taken a lead in developing tools to help manage large numbers of protocol records.

Assure conceptual integrity with a single designer.

I think the conceptual integrity of *Prose II* was enhanced by having a single designer and implementer. I conjecture that this accounted for high productivity and fast, reliable code.

7.2 Using the Protocol Analysis Tools to Guide System Refinement

A six-pass parser served as the primary tool for examining the 112 protocol records. The parser automated and extended the work laid out by Card, Moran, and Newell in their *ICARUS* study. What did Card, Moran. and Newell use their parse for?

- To identify three broad phases, lasting 5 ~ 15 minutes.
- To classify unit tasks, lasting 10 ~ 30 seconds.
- To identify command frequency; they found that six different commands accounted for 85% of the command executions.
- To see that their user processed elements in small groups; for example, he transcribed about three circuit elements at a time.
- To look at the nature of user errors in the session.
- To predict the overall execution time, using their Keystroke Model.

Because of the higher-level and unstructured nature of the task reported in this current project, the last item could not be examined with the current study. The tracker used in this project did not count individual keystrokes, since they would have greatly increased the size of the protocol record files. Also, they were not considered essential to the higher-level user strategies we were trying to ascertain.

Similarly, errors, in an exploratory task such as writing, are not readily identifiable in an execution sequence without knowing a user's intentions. However, the analysis did reveal situations where a node was deleted, and then a new node was created soon after that. This could be considered a user error.

In their study, Card, Moran, and Newell observed one subject who subject completed one session of consistent, structured work. Automating that approach directly showed some problems not experienced by these original researchers:

- Handling of multiple documents in one session
- Handling of multiple sessions for one or more documents
- Handling of long, unaccounted-for pauses
- Handling for undirected work, or users inexperienced with the software

The value of the parser in this project can be seen to have a wider scope than simply automating the analysis done by Card, Moran, and Newell, since its use was extended across many subjects and sessions:

1. It provides a consistent classification of sequences of commands and episodes.
2. It reduces large volumes of protocol data to a manageable size.
3. It avoids many human data-handling errors during collection and analysis.
4. It simplifies frequency analysis of all kinds; an analyst can tell which activities were performed most frequently and where the most time was spent.
5. It focuses on activities by types and sequence. In particular, this parser was not particularly useful for making judgement about time allocations. It was also not sensitive to fast versus slow work and to long pauses

The parser indicated the following items as areas for further refinement of the software used in this project.

Identify the frequency of use for the commands within specific types of sessions.

The automated parse analysis made it straightforward to see how often each of the commands was used. Among the non-trivial sessions, more than 50% of the commands were CreateNode, EditLabel, or LinkNodes. This kind of information is a clear indication to developers of which commands to make easiest to use, which commands to have dedicated mouse buttons or accelerator keys for, and which commands should have the fastest performance.

I had received early user feedback that suggested that the LinkNodes commands should be easier to use. I changed its mouse interface so that a single mouse clink creates a node, while a double click both creates a node and positions the cursor to begin labeling it.

At the other extreme of frequency distribution, there were several commands that were used rarely. As a developer, I must question whether the added complexity involved to implement and document these commands is worth the cost to the eventual users. For example, there were functions that I found interesting that were not used by many subjects: ClipboardCopy, ClearDrawing, BreakAllLinks, TreeGrow, and TreeShrink.

Further, would this frequency distribution have been different if the subjects had more experience or were working in a different environment? It can reasonably be argued that most subjects were new users of the system, and therefore tended to use those commands central to its operation. More powerful and complex commands are used by more experienced users, it is often claimed. (This approach begins to mirror the RISC versus CISC discussion among computer architects.)

Identify the frequency of use for the file formats.

The protocol analysis showed that two of the six workspace file formats were not used by any subjects. The formats never used were `.RDY` and `.LST`. Several hundred lines of the internal code of *Prose II* are incorporated to handle these two formats; this code could have been removed without affecting this set of subjects. This is one example where the tracker served to "profile" the subjects and how they used the software, identifying sections of code that were never executed—and are thus overhead to the system, its user interface, and its documentation.

The analysis also showed that *Prose II* was frequently used as a file format translator; workspace files were opened in one format and saved with a different format.

Determine appropriate system defaults.

For example, the tracker can easily record the size and position that users specify for their map and outline windows. Should the initial default be changed? Should the current position be remembered across sessions?

Unfortunately for this particular question, the tracker information was incomplete, because of a bug in my implementation. However, this kind of information should be available with this type of tracker, and could provide helpful information for software developers who are designing default values into their systems.

Determine which Helps need attention.

The automated analysis provides straightforward direction on which Helps are most frequently used and in which the most time is spent. In some of the Help panels, users spent more than twice as much time as in others. For example, help for complex operations (*e.g.*, updating the `WIN.INI` file, or dealing with the six file formats) took about twice as long as for others (*e.g.*, using the mouse). This may show that these panels provide too much information, or are not clear in the information they present. These Helps should be made easily available; the information they provide should be also explained clearly in the user's manual.

Also, some of the Helps were requested much more than others. Help for frequent commands (*e.g.*, using the mouse, labeling and editing) was requested more frequently than for less-frequently-used command (*e.g.*, changing how the nodes are drawn).

Determine how often a Save is necessary.

When we started this study, we did not have a good feel for how often users save their work. Should *Prose II* do saves automatically? The protocol records showed that the maximum number of saves in a session was 5, but the median number of saves was 1. Subjects tended to save once at the end of a session: until then, they were not ready to save their work. They did not appear to fear system crashes or power failures (although such records would never be seen, since the tracker did no checkpointing.)

Determine the size of nodes and their labels.
I designed the size of the nodes with only an intuitive feel about how large they should be. With a 6-point font, a node of the default size held three lines of text with about 20 character per line. Up to 250 characters of text could be written in the node labels; long labels could be seen in the Edit dialog box or if the size of the node is zoomed. Analysis of the protocol records showed that most node labels consisted of a few choice words—exactly what they were intended for.

Determine the size of displayed trees.
Most trees contained fewer than 20 nodes, although a few trees had about 40 to 50 nodes. The tree size, combined with the size of the nodes, determined how much a subject could see in a normal *Prose II* window at a time.

Determine the necessary tree depth.
When I designed *Prose II*, I arbitrarily chose a maximum number of levels in displayed trees since one of the internal data structures was a compiled array. I decided that a maximum tree depth of 99 levels was a reasonable limit, based on my own use and on my observation of some pilot users. The protocol analysis showed that no tree in the study was deeper than 10 levels. This is an example of how usability tests can assist developers in making implementation decisions.

Minimize the number of Refocus episodes.
About 25 percent of all housekeeping episodes were spent playing with navigation: roaming and zooming to new areas of focus within a workspace. This suggests that the screen was often too small for the trees being displayed. *Prose II* users could probably be more productive if they spent less total time in Refocus episodes.

It would be interesting to study this with a controlled experiment, where two different screen sizes are used. If the amount of time spent doing Refocus activities decreased with larger screens, one could build a case for doing a cost analysis, comparing the cost in employee time for using a small screen as opposed to using a large one.

Expect the tracker to find unexpected sequences.
The tracker showed sequences of commands that I do not know how to manually reproduce. An example of this is the following:

```
 ⋮
50:55.50  50:55.55    0.05  MapWindow         OPEN
 ⋮
52:17.39  52:17.45    0.06  MapWindow         OPEN
 ⋮
55:44.96  55:45.01    0.05  MapWindow         OPEN
 ⋮
```

Figure 31. Multiple opens of the Map Window without intervening closes. When I tried to reproduce this, closing of the Map Window was shown. How did this occur?

In this example actual users performed a sequence that I never saw during development and testing, and could not reproduce. Additional pilot testing might have determined what this unexpected code path was and whether it was causing problems for users.

Offer a means of leaving comments in the recording.

As implemented in this project, the tracker captures one primary aspect of the human-computer interaction: its command interface. The tracker in *Prose II* also added a way for humans to record their comments in context. This was a way to offer the unstructured feedback available with think-aloud protocols. Many subjects found this to be a useful way to correspond with the protocol analyst(s) and software developer(s). Subjects used this mechanism to describe what was occurring during long elapsed time periods, to comment on how they were using the software now or would use it in the future, to praise functions they liked, to complain about functions they did not like or were missing, and to report bugs.

As both the software developer and protocol analyst, I found the feedback helpful, and encourage incorporating such a mechanism in similar software.

7.3 Enhancements to the Testbed System

I had a strong desire to continue modifying the software used in this study, especially when analysis results showed areas of immediate interest. Ever the programmer!

Since the testbed system, *Prose II*, was stabilized and used in this study, many hypertext writing systems for personal computers have appeared on the market. These systems are becoming quite sophisticated in their use of graphics and multimedia, and are proving useful in many professions. Fersko-Weiss (1991) gives a comprehensive overview of these current systems.

Certainly *Prose II* can be updated with the functions and features contained in the latest hypertext systems. I compiled the following list of enhancements to *Prose II* during the study; these are in addition to enhancements driven by competition in the marketplace.

1. Allow multiple, concurrent workspaces.

   Multiple copies of *Prose II* can be active at any time on a machine. However, you cannot drag nodes and trees among *Prose II* windows. This would allow parts of documents to reside in separate workspace files. As a user of *Prose II*, I found that I frequently created many more nodes than I needed during a session of brainstorming and exploration. As I developed a document further, I found that I wanted to drag the nodes I chose not use into another document for safekeeping. This was a tedious process with the current implementation.

2. Allow collaboration on work, as groupware, over networks.

   As readily as I can create, link, and write in nodes in my own workstation, I would like to have the ability to do this same thing cooperatively among two or more users in a network. Many users could view the same workspace, adding and writing in nodes which could be viewed in real-time by other members of the group.

3. Port the *Prose II* source code from *Windows* to run in the OS/2 Presentation Manager environment.

   The OS/2 operating system for personal computers did not exist when *Prose II* was first designed and implemented. Although not yet as widely installed as DOS and *Windows*, OS/2 proves to be a more suitable environment for a system of this type on personal computers.

   - Automated trackers are well-suited to OS/2's background thread structure.
   - Online parsing is well-suited to OS/2's background thread structure. The tracker could feed its output directly to the parser, which could display the current parser tree for a session in real time.
   - Checkpointing and quick file saves are well-suited to OS/2's background thread structure.
   - Having programmed extensively in both environments, OS/2 is a more productive and robust environment in which to develop and run programs.

4. Provide handling for general directed graphs (*i.e.*, nodes that have more than one parent)

   Prose II allowed a node to have no more than one parent, since this was algorithmically "safe" and explainable within the original target usage. Given an outline as input, a two-dimensional tree could be drawn; given a tree drawing, an outline could be generated.

   Allowing general directed graphs is a more difficult problem, but one that could find many uses. Suitable algorithms might be invented through iterative refinement: initially work within some arbitrary restrictions, such as no more than two parents per node.

5. Indicate additional visual information about the file underlying a node.

   The nodes in *Prose II* showed, by their shading, whether a file was attached to them. An additional indicator to show the size of the associated files would have been useful.

7.4 Summary

Chapter 1 noted that feedback to software designers is increasingly important if they are to match the software tools they create to the complex mental tasks the software is designed to support. Many types of feedback were available at each stage of this project; this paper attempts to capture the flavor and detail of that feedback. Feedback to the software refinement process was outlined in this chapter. In particular, it described specific "change-orders"–modifications to the application program directly indicated by the analysis results–as well as steps to make the ongoing software development and refinement process more efficient. It also described some enhancements that became apparent through long experience in developing, testing, and using the application program myself.

I played several roles in this project, roles often divided among several people: software designer, software implementer, protocol analyst, and human-computer-interaction researcher. In all of these roles, I gained insight into how this type of software–the testbed application program and the protocol collection and analysis tools–might be improved, both in their functional content and in their development process. The final chapter summarizes the highlights of this feedback, and discusses important areas for future researchers to focus.

CHAPTER 8. CONCLUSIONS AND FUTURE WORK

This project was a first step in automating the process of analyzing and describing usage of computer software. As such, it probably raised more questions than were answered—often the result of an exploratory project. This chapter summarizes what was learned about the behavior of users of this particular testbed system. One of the goals of the analysis was to point to elements of the testbed system that could be improved; these are discussed next. This is followed by conclusions on the methodology used in obtaining and analyzing the data for the study.

Finally, as an exploratory project, many follow-up items were tracked. The chapter closes with an agenda for use by future researchers in the area of human-computer interaction.

8.1 User Behavior

The backdrop for this project was an examination of subjects in their actual field setting, using a new software system to assist them in writing. Chapter 5 provides extensive detail on the results of the analysis of the 112 session recordings received from the 29 subjects. It also contrasts the results obtained in this study with those of previous researchers. Some of the interesting findings of this study are summarized below in three broad categories: what types of activities took place across the sessions, how users spent their time and how long events lasted, and what kinds of groupings and sequences were observed.

8.1.1 What Kinds of Activities Took Place?

Much use of the system, 38% of the sessions, was trivial or unproductive. I did not anticipate this, and hence did not generate specific information about why this occurred. Reasonable conjectures on the high number of trivial sessions are:

- This an artifact of using a new system.
- These were the type of sessions returned by subjects who return recordings.
- This is how people work.

Determining why so many trivial sessions were received is an interesting topic for future exploration. What variables in the setup of the study or in the system itself would reduce or increase the number of trivial sessions?

The observations that follow provide insight on phases, episodes, and commands in the non-trivial sessions.

- By count, about 80% of the phases involved exploration, defining hierarchies, or top-down construction. There were frequent alternations among these types of phases.

- Although 9 different types of housekeeping episodes were identified by the parser, about a quarter of the housekeeping episodes were concerned with refocus operations. This was probably because of the small screen sizes available to most subjects.
- 39 commands were available in *Prose II*. Half of the commands in the non-trivial sessions can be accounted for with just three of these commands: creating, labeling, and linking nodes.
- About 1/8th of all nodes that were created were explicitly deleted. This was unexpected, since nodes could have been easily moved or re-labeled.
- Most subjects tried running a text editor within *Prose II*, but few used it for extensive writing of a document.
- Subjects frequently used the *Prose II* function that tidied their drawings of trees. This novel function proved popular. Other novel functions (for example, Scramble) were rarely used.
- Some subjects found the novel ability to leave comments in the tracker appealing. They left comments in the tracker for a variety of reasons that generally fell into five general classes:
  - Descriptions of what happened during some elapsed time
  - Observations on using the system
  - Praise for functions of the system
  - Software bugs, requests for additional functions, queries
  - System or hardware problems.

8.1.2 How was Time Spent in the Sessions?

The frequency distribution of the time duration for most commands, episodes, phases, and sessions showed large differences between the median and mean times, as well as large standard deviation values. Graphs of these distributions show curves with an early peak and a long trailing tail. Out of this, some generalizations were identified.

- Sessions were generally short, about a half hour or less, when no extensive writing in the nodes was involved. Sessions with extensive writing lasted about an hour.

  Previous studies of users doing writing typically lasted two to four hours. When working on their own, it appears subjects write for much briefer periods.
- Overall document times, for documents composed over one or more sessions, were about an hour.
- Dividing total document time by the number of nodes, subjects spent about one minute per node if nodes were merely labeled; if extensive writing in the nodes was involved, they spent about 5 to 10 minutes per node.
- Subjects spent about 10 to 20 seconds each time they labeled a node. Labels consisted of a few choice words, averaging 4 words of 7 characters each.
- Help screens were typically viewed for 15 to 30 seconds.
- The time duration of phases was consistent between non-trivial and trivial sessions: about two minutes. However, the sequences and distributions of episodes within the phases were quite different between trivial and non-trivial sessions.
- One-sixth of the session work time was spent in phases of revision of the document text.
- About half the total session time is spent in pauses. Most pauses were less than 3 seconds. Two-thirds of all pauses were less than 5 seconds.

8.1.3 What Kinds of Patterns were Observed?

Throughout the analysis, we found that work tended to fall into classes. The first distinction was between trivial and non-trivial sessions: the trivial sessions were later isolated from most of the protocol analysis. Non-trivial sessions fell into groups of single-document sessions, multi-document sessions, and multi-session documents. Some sessions also showed work on documents that already existed.

Differences in strategy became most apparent in the differences among small, medium, and large documents. Sessions showed natural groupings according to how many nodes were operated on during the sessions. The groupings appeared to reflect the subjects' experience with *Prose II*.

- Sessions with 22 or more nodes looked similar: a critical mass of nodes were created and linked into a tree, which was incrementally developed.
- Sessions with 11 to 21 nodes showed three kinds of behavior, possibly because the number of nodes was manageable enough to be manipulated in several ways. As with the sessions with 22 or more nodes, trees were incrementally developed in some of these sessions. In other sessions, all the nodes were created and labeled before any were hierarchically linked. And in the third group, subjects' behavior looked almost turbulent as they searched for the right ideas and relationships among them.
- Sessions with 10 or less nodes showed little overall patterns.

Some other observations involved the size of documents, sessions, phases, and episodes, where the meaning of size depends on the topic being discussed.

- Documents that were constructed across several sessions were smaller (*i.e.*, had fewer nodes) than documents constructed in a single session.
- The counts of commands per episode and episodes per phase were nearly the same, with small variation. These counts are consistent with the size of human Working Memory chunks: about 3 to 5 commands/episode and episodes/phase.
- Trees generally had 3 to 5 levels, consistent with the conventional structuring of documents into chapters, sections, subsections, and paragraphs.
- Most trees had between 15 and 30 nodes. Subjects generally saved their work when it was at some state of completeness: most saved documents consisted of a single, complete tree, as opposed to a forest of trees or unlinked nodes.
- There was a preponderance of top-down construction, as opposed to bottom-up construction. The top-down construction took place in the form of iterative refinement.
- A node's order of creation generally correlated with its final, pre-order tree position. Nodes created early in a session finished high in the structure trees; nodes created late were low in the tree. Two-thirds of the time, the first node created was the root of the eventual tree.
- In terms of spatial positioning, a node is generally created below and to the right of the node created before it.
- Planning and writing were generally intermixed. Only one session had all the exploration and planning before any of the writing.

Ultimately, sessions varied from one to another among many variables. Sessions could be grouped by looking across any of these variables, or they could be characterized by looking across many variables in one session. For example, all the sessions where "trees with more than 30 nodes were created" could be readily compared.

8.2 Software Development Feedback

One of the important goals of this study was to understand the type of feedback available to software designers and developers using these types of analysis tools. This section lists modifications to the *Prose II* testbed software indicated by the protocol analysis.

8.2.1 Frequent Refocus Episodes

The analysis showed that 25% of the housekeeping episodes were Refocus. Users spent much effort navigating to other areas of the workspace and zooming for better focus on what they were working on.

With the state of personal computer technology at the time of this project, I assume that most users had display screens with resolutions of 640x350 or 640x480 pixels. While Prose II will work with any screen resolution, it cries out for a larger/denser screen. I personally used a display of 1280x800 pixels, which felt about right. Most subjects were somewhat constrained by smaller screens, resulting in frequent refocusing.

A tracker of this type can show software designers how many nodes are present on the screen at a time. Similarly, it could show software buyers how much cost is involved in roaming and zooming. It could help in targeting GUI software to computers with screen real estate of the most prevalent size. If an overwhelming number of target users have screens that are 500x500, writing software that works best on a 1000x1000 pixel screen may be counterproductive.

By also collecting extra information with the tracker, additional strategy questions could have also been answered, such as, "Is tree size correlated with screen size"?

8.2.2 Concatenated Function

The protocol analysis showed that half of the more than 6000 commands executed by subjects during the study were creating nodes, labeling nodes, and linking nodes. These three commands therefore needed the closest attention:

- they should perform well,
- they should be easy to execute, and
- they frequently occur together in sequences.

In early feedback, the focus on these three commands became obvious from studying protocol records and from direct comments from users. The mouse buttons were designed so that these commands were readily available with single click, double click, and click and drag, respectively. The Edit dialog box was re-designed so it was easy to enter the text for a node's label. Editing of a node's text was also simplified.

The large number of times these three commands were executed suggests further improvements may be warranted. Some ideas to be studied further: pre-built templates of common tree structures, and full-time icons for sequences of create, label, and write.

8.2.3 Extra Function

The protocol analysis readily identified commands were used with several orders of magnitude less frequency than others. Thus, the system may have had elements of functional overkill: some of the features were rarely used.

Because it appeared there were many trial and initial sessions with *Prose II*, I am initially reluctant to remove the rarely-used functions without some further longitudinal studies of experienced users. Clearly some functions were not used by novices with this system; are the functions, however, helpful to experts?

In particular, functions without a direct mouse/menu interface (that is, they could done only with the keyboard) were rarely used by the subjects in this study. If these functions were now given mouse interfaces, would their frequency of use increase (assuming they had the same utility as before)? Or, would the additional menu size and complexity change the overall comfort with the system?

8.2.4 Tidy, Save, and Help Commands

Subjects used the novel TidyTree command frequently: this allowed them to work cluttered and then ask the system to clean up. This appears a good paradigm for user behavior with such software, where complex structures are being created and modified.

Subjects rarely saved their work during a session; an ongoing automatic save function could avert disasters for some users. The analysis showed mean and standard deviations for inter-command pauses; to reduce disruption to users, a suitable (and possibly self-adjusting) interval for automatic saves could be implemented.

The protocol analysis readily identified help panels that were used frequently, and showed on which helps the most time was spent. The context where help was needed could also be easily seen. Frequently-requested helps point to obvious candidates for functions that need to be simplified.

The observations in this section were highlights of the extended descriptions found in Chapter 7. They indicate that the project was fruitful; it showed areas for improvement that were not expected when the software was designed. The project met this goal: useful feedback was obtained on how to improve the specific software under study.

8.3 Tools and Methodology

Aside from learning lessons about peoples' strategies with this particular software, this project was a platform to describe how we examined a group of software users and what we learned from doing that.

The automated techniques explored in this project are intended to supplement, not supplant, other techniques for acquiring and analyzing protocols, such as think-aloud verbalization and videotaping. These methods still offer unique advantages of their own: namely the ability to capture users' attitudes and emotions. Moments of delight, frustration, or bewilderment, as well as interruptions and distractions cannot be effectively captured with an automated tracker. Perhaps most importantly, it has proved difficult to capture users' intentions with automated techniques.

8.3.1 Advantages of These Tools

By contrast, the tools and methods presented here are intended to address issues of users' styles and strategies across a large sample of sessions and subjects. They offers several advantages over previous protocol collection and analysis methods.

First, it allows unobtrusive collection of protocols from actual users in the field. The protocol collection is not done in an experimental setting. Users can perform tasks they consider representative and use the system as they wish without the intrusion of monitoring personnel or equipment.

Second, because the protocol collection can occur "in parallel" among many users in their naturalistic settings, a great deal of protocol data can be collected in a short period. The shortened period potentially allows quicker software design feedback. The large volume of data offers a broad look at the use of the system and the variety of strategies employed in its use. The automated analysis makes the volume of data manageable.

Third, after protocol records have been collected, the grammar and parser can be modified as understanding of system usage increases. The protocol records can then be rerun with the modified parser to see if the new decisions reflected in the parser still hold for the large cross-section of existing protocol records.

Fourth, it allows researchers to see how peoples' software usage changes over time. Introspection says that the ways we use a piece of software changes as we gain more experience with it. How is that change manifested in users' usage records? The internal tracker and consistent analysis tools make such longitudinal studies possible.

In general, automating the tracker and protocol analysis was a convenient way to observe many users. This method of software distribution and protocol collection allowed a large volume of data to be collected in a short period. The cost of collecting and analyzing each session is relatively low, after preparing the tools.

8.3.2 Observations on Protocol Collection

Chapter 6 contains an extensive discussion on constructing the automated tracker and collecting the protocol record files it generates. A summary of the protocol collection aspects of the study follows.

- 14% of the users that were contacted returned their recordings. Anecdotal data says this was a reasonable reply rate for a voluntary experiment.
- The tracker was designed to avoid writing any secret or hidden information; users could browse through any protocol file with a simple text editor. This gave users confidence that private information was not being collected about them.
- With few experimental controls, many sessions reflect merely learning and exploration of the software itself. Much like any harvest, the ease of this method of protocol collection requires additional methods to filter out the chaff.
- It would have been helpful to have some supplement to the protocols, indicating users' intentions. The amount of experience each user had with the system would have also been helpful, to discern novice sessions from expert sessions, and to watch a progression of sessions as a user moves from novice to expert.
- Giving users a mechanism for inserting comments into the protocol record proved valuable. It allowed users to report problems and comments in the exact context where they occurred.

This feature could be expanded upon considerably (up to the point of obtrusiveness) and should be considered for addition in similar software.

- Subjects produced about 23 Kbytes or about 7 pages of protocol data per hour with this system.

The automated tracker proved to be a simple, but powerful addition to the existing software system. It was easy to incorporate a tracker in a software system without degrading performance.

- The internal tracker used a synchronous interface, which did not perceptibly slow response time and simplified the amount of state information needed to identify each event.
- The tracker recorded events at the command level, *e.g.*, "Create a node." If every keystroke and mouse movement had been tracked, system performance may have been slowed and recording files could have become needlessly long.
- Communications networks proved a powerful tool for software development, distribution, and protocol collection.
  - I was a developer of this Microsoft *Windows* application before there was adequate documentation. I exchanged programming questions with Microsoft *Windows* experts via the *Genie* network, where Microsoft supported a bulletin board service.
  - The testbed software used in this project was distributed for a year via IBM's internal network before protocols were solicited. This allowed me to fix bugs and make improvements based on users' feedback over the network. I was confident the system was stable when I started the study.
  - The study was distributed to subjects in their actual field setting via the network. The protocols were returned to me over the network.

While it was simple to collect many protocol record files, their management became time consuming. For example, file management became a problem as protocol files with arbitrary names were collected from different user's machines. As the tracker was implemented, it invented filenames unique to the user's machines. These names were no longer unique when all the files from all the users were stored on one machine for analysis. Also, users had occasion to change their protocol filenames before returning them to me for analysis.

In summary, many users were comfortable with generating protocol records of their sessions, and returning them for analysis. It is easy to collect a large volume of actual-use protocol data! Collecting and analyzing protocols as done in this project scaled up to about 100 sessions, but cataloging and file management was becoming a predominant problem. Easy categorization of groups of sessions, documents, and subjects is necessary as the number of sessions increases. A database management system is needed to manage large numbers of session protocols.

8.3.3 Observation on Using a Parser for Protocol Analysis

The multi-pass parser used to analyze the protocol records allowed intellectual manageability and programming refinement that was bounded in scope. For example, difficulties in the analysis were eased late in the project by adding an additional pass to the parser. Also of value was the parser's ability to write its intermediate files in spreadsheet format. Complex analyses and graphing could readily be performed on any factor or variable discovered in the analysis. This ease of analysis led, in turn, to the addition of more variables to the parser output, to satisfy requests for specific views of the sessions.

The parser was labor-intensive to build, but straightforward to use once it was stabilized. Having a large number of valid "testcases" (in the form of actual protocol record files) made regression testing of parser changes easy to automate.

Playback of user sessions was not incorporated into the analysis. Playback is a difficult problem across platforms: it requires a great deal of contextual information. Many other activities, besides the one being studied, are possible in a multi-windowed environment, which compounds the understanding of what the user is actually doing when not directly using the target software.

Several innovations in the collection and analysis helped me to uncover useful information from the subjects and to work more efficiently:

- The tracker captured commands, not just the keystrokes. This makes the recordings shorter and saves the step (and possible ambiguity) of recombination during the analysis. Both Card, Moran, and Newell and Smith *et al.* have captured all the keystrokes, and then have had to go to the effort of encoding groups of these as commands.
- I distributed the system over the network and getting the recordings back the same way. The recordings could stand alone, and were compact enough that they did not need to be compressed—to avoid network performance issues.
- This project introduced a group of statistics that can be used to examine a similar task or software system. These are described in detail throughout Chapter 5.
- *Prose II* contained hot keys, allowing users to communicate directly with the tracker. Users could relay their thoughts to the analysts and presumably the software developers in the exact context where the comment occurs. Subjects used this for several purposes in this study, including describing bugs at the place where they occurred.

With these innovations and those of others, this project progressed confidently in its quest to understand how we use computer software and how it can be improved. The initial *ICARUS* study of Card, Moran, and Newell provided an excellent basis for automating the tracker and parser. Additional input from the concurrent Textlab research supplied a rich environment in which to experiment with implementation ideas. The resulting tools and techniques succeeded in shedding light on user behavior, on methods for studying user behavior, and on ways to improve human-computer software interfaces.

8.4 Future Research

In this exploratory project, ideas for future research and modifications were frequent throughout all stages of design, implementation, testing, analysis, and summarization. Specific suggestions for changes and extensions to the analysis tools and the testbed system are described in the final sections of Chapters 6 and 7, respectively.

This section identifies four areas where additional studies and overall changes seem warranted. First, with these software tools in place, numerous ideas for fruitful longitudinal studies are described. Next, many of the analysis summaries involve rates, types, and sizes; ways this knowledge could be used to build better human-computer interfaces are discussed. Third, media other than text were rarely used in this study; how would the results change with advanced hypermedia systems? Finally, using the analysis tools constructed for this project suggested yet more tools that a designer or analyst would like to have. Issues involved with building an integrated suite of analysis tools are described.

8.4.1 Conducting Longitudinal Studies

This project laid the groundwork for observing users with a particular software system over an extended period of time. The tools and techniques demonstrated here make those types of studies tractable.

- Studying individual users over time: effects of learning

  The tools introduced in this project make it simple to watch an extended sequence of user sessions over time, as individual users move from being novices with the software to being experts. How many sessions or how much elapsed time does it take to become an expert? What identifies an expert (*e.g.*, fewer errors, longer pauses, larger documents, shorter sessions)? What is the mix and sequence of commands an expert uses, as opposed to a novice? (We have an introduction to this answer in this study, with the mix of trivial versus non-trivial sessions.)

  For software designers, can alternate implementations be objectively compared with two scales: how fast users can move from novice to expert, and how efficiently experts can perform given tasks?

- Studying experienced users: how do expert writers work?

  I conjecture that most of the subjects in this study were neither experienced users of this software, nor were they experienced writers. What would the data analysis have shown about writing behavior if all the subjects had been skilled writers, experienced with this software system? Future studies with such a focused group of users will likely change the protocol analysis results summarized in Chapter 5.

- Characterizing users by what they were trying to do

  The protocol data that was collected contained no background information on what goals its users were attempting to accomplish. Thus, the major categorizations that were made in Chapter 5 were made by the kind of session: single-document sessions, multi-session documents, and so on. Without becoming too obtrusive, a better categorization might take into account verbal feedback and summaries from users. Users could guide how analysts determine highlights of sessions and categories for them.

- Understanding the short, unproductive sessions

  Nearly 40% of the sessions were trivial or unproductive. A follow-up study could examine further why this occurred, and how much semblance this bears to normal behavior. For example, I know of no similar data with respect to the use of text editors or word processors. Are unproductive sessions in fact helpful, part of the exploration and learning process? Do certain software systems encourage unproductive sessions? Is similar behavior manifest in other, non-computer-related activities?

8.4.2 Utilizing Rates, Times, Types, and Sizes

One way to improve the overall performance of a computer system is to take steps that bridge the speed mismatch between processor memory and disk memory. Much computer science research has focused on ways to use caches and similar technology to optimize a computer's throughput.

A similar mismatch exists between the speed of human operations and underlying computer speeds. Certainly, human speed seems so much slower than raw CPU speed that this shouldn't be a concern, but advanced software functions and current GUIs have a tendency to consume computing resources. To bridge this gap, we need to understand the rates, types, and sizes of human input operations. With this knowledge, future systems might be tuned to adjust to the performance characteristics of their human users.

Card, Moran, and Newell (1983) organized a large body of diverse data on how the general "human information processor" operates. For specific tasks such as writing, additional data is needed to adjust software to its users. The protocol analysis tools used in this project illuminated many attributes of how users worked with the testbed software. For example, they showed

- the distribution of the duration of pauses,
- the distribution of writing times in nodes,
- the distribution of time spent reading helps,
- the frequency of file saves, and
- the distribution of label lengths.

Future software designers should consider capitalizing on the rhythms of their human users, making their software appear more responsive. For example, highly-interactive computer software can use the rhythms and sizes found through protocol analysis to find opportune times for I/O, backup, checkpointing, and garbage collection.

Similarly, resource allocation can be optimized to utilize practical human sizes, instead of large, fixed sizes. Items far outside the expected range of sizes can be treated on an exception basis; the protocol analysis gives a basis for understanding the frequency of the exceptions.

8.4.3 Using Media Other Than Text

White (1985) proposed that we work with three types of concepts in our human cognitive systems: propositions, algorithms, and images. All subjects in this study used propositions (represented as labeled nodes) almost exclusively, even though *Prose II* nominally supports all three types of concepts. A current challenge for computer scientists is the utilization of algorithms, video and audio images, and their mix: how do we make them more inviting to express in a hypermedia system, and how do we integrate them into "hypermedia documents"?

I assume that the rarity of algorithms and images as types of nodes was a by-product of their inadequate support in *Prose II* (and computer systems, in general). Recent hypertext systems, however, are beginning to offer broad, integrated support for multimedia.

Strongly-integrated hypermedia systems introduce new questions for this kind of study. With different types of nodes (instead, or in addition to, labeled nodes), how do many of the behavior results obtained in this study change? What is a common mix among the types of nodes? A follow-on study could look at such things as the time spent per node, the duration of pauses, the type and duration of episodes and phases, and the size of the resulting multimedia documents in an attempt to better understand how humans behave in their interaction with these new systems.

In *Prose II*, a node consists of only a single type; its label is always text. This restriction was an implementation decision which reduced its underlying complexity. However, a full hypermedia system should allow, for example, nodes composed of text, with audio annotation and imbedded graphics. Although the current tracker design is sound, extensions are required to accommodate this richness of media. One feature of text-only systems is that the data storage of text is relatively compact compared to audio and video. It was easy to capture node labels in a text system; it would be more costly to capture a graphic icon or audio message and store it in a protocol record. Hypermedia systems potentially offer a rich development environment for multimedia documents. The challenge presented to their developers is to make them so easy to use that they actually get used. The tools introduced in this project offer designers and developers a means to get consistent feedback on their systems' usage.

8.4.4 Developing an Integrated Suite of Tools

The parsing tools developed for this project were designed to make protocol analysis less expensive and more consistent. Using them over an extended period suggested, in turn, other software tools that could help make protocol analysis more accurate, more useful, and faster.

To improve accuracy, analysis tools could better match protocol analysis to users' intentions and experience. This can be done by allowing user input to guide or influence the parse. For example, the parser built for this project could be adapted to operate concurrently with actual user tasks; it could be shown in an adjacent window. Future enhancements could be envisioned that allow a viewer to give a running audio commentary and to modify an ongoing parse. This ability could be exercised by users concurrently during a task, as a think-aloud protocol, or by analysts later while viewing a recorded session.

Another way to improve accuracy is to provide analysts with tools to help visualize one or more sessions. Visualization of sessions through playback has proved a powerful tool for researchers using the *WE* analysis tools. As mentioned, playback can be technically challenging in an arbitrary, multi-tasking machine environment. Additionally, the nodes in this project and the Textlab studies consisted almost exclusively of text. With a richer mix of hypermedia components, playback of complex sessions will need new algorithms for representation and retrieval.

A parse tree can provide visual results for a single protocol analysis, as shown in this project, but additional tools would strengthen the capabilities of this tool. For example, the parse trees illustrated by *Prose II* do nothing to illustrate the passage of time. Parse tree illustrators and editors can be imagined that picture different factors of a session with different colors and icons. Additional tools are also necessary for looking across a groups of session analyses for various features.

Visualization tools to look across groups of sessions point to a general class of protocol analysis tools to manage and query session analysis results. In this study, the number and size of the sessions allowed me to manage them by hand. This should be automated with a database management system; we should be able to put the protocol records in a standard database with search keys and relations. Currently, the Textlab group is looking at constructing extensive headers for protocol records. They have given the protocols attributes, which become an abstract description of the protocol.

Analysis tools are needed to look across the landmarks and features of vast amounts of protocol data. "Show me the prominent portions of a single session, of multiple sessions, or of all the session with a certain category. Or, show me all the sessions containing some factor, and show me other features the they have in common."

The scale of the problems allowed by analysis tools needs to increase, as well. Working in DOS would bound many questions had my sample size been much larger, because of its constraint of only 640 Kbytes of main memory. Modern workstations could have increased the practical study size by an order of magnitude.

Finally, if parsing of protocol records is to be used more widely, tools are needed to build better parsers faster. To be used across different kinds of software systems, skeletons of operating parsers should be readily available and modifiable. Simple modifications to an existing parser should be simple to make. In this project, they require knowledge of C programming, plus extensive testing to assure they have not changed the results of previous protocol records that were parsed "correctly." Alternate parser implementations might make them easier to develop and modify. For example, technologies such as neural networks or expert systems might be used to create parsers that learn how to parse by being guided by a human through a series of representative parses.

I assumed that an automated tracker and parser would be a panacea for studying user behavior with software. What it allowed was to scale up the types of questions I could ask about broad behavior patterns. To obtain answers to the new spectrum of questions suggested by this study, additional tools are needed to coordinate the resulting analysis data. Integrating these tools into a workbench for protocol analysis would make working with large sets of protocol data much more tractable, and hence more timely and useful.

APPENDIX A. COVER LETTER AND SESSION SUMMARY

This appendix has two sections. The first is a copy of the cover letter sent to potential subjects. The second section contains the exhaustive itemization of each of the 112 sessions. It also shows, by session name, the five categories of sessions, and the 10 sessions that contained extensive writing.

A.1 The Cover Letter

I sent a copy of the following cover letter to each of the 210 potential subjects on January 19, 1989.

Cover Letter

I'd like your help for a study I'm conducting on human-computer interaction, part of my Ph.D. dissertation. I recently sent you the current version of my structure editor, Prose II. You probably noticed that it makes recordings (ASCII files) describing the actions you took while using it. I'm building some sophisticated tools to analyze these recordings, and need a number of good recordings to analyze.

Please use Prose II to design and write a paper, an article, a set of class notes, or some similar document of between 2 and 15 pages, and then send me the recording (I don't need the actual document). In fact, I'd really like two different recordings from you, if possible--maybe two different kinds of documents? The recordings will be most useful if you use the system from beginning to end in the process of writing the document.

You will not be judged in this study. There will be no public association made between your name and any recording. The recordings contain no confidential information.

I'm hoping to appeal to your desire to advance the state-of-the-art, but I can sweeten the deal a bit with the following:

- I will acknowledge you in any papers or articles that arise from this research (let me know if you don't want me to).
- I can send you a copy of your analyzed recording.
- I can send you a copy of my research proposal.
- I can send you a copy of any papers that arise from this research.
- I can send you a copy of Prose II with the recorder "compiled out" (thus, a bit smaller and faster).

My target date to get your recordings is February 25th. Do what you can (sorry--I know you really don't need another schedule!).

Thanks in advance, - John Q. Walker

A.2 The 112 Sessions

The 112 session protocols are summarized below in Table 40 on page 164, grouped in chronological order by the person who submitted them. I assigned each of these protocol records a unique name, which is the session ID shown as the first column in the table. The column entitled "Number of Commands" is a count of the primary commands executed in a session. The rightmost four columns identify, for a given subject, which document a given session applies to. This is because two subjects returned sessions showing work on four documents; the other subjects showed evidence of work on three or fewer documents. Other symbols in the table are described by the following legend.

Legend of Symbols:

Symbol	Description of this symbol
E	Existing file: the subject opened and worked with an existing file. The file was not necessarily created using *Prose II*.
N	No save: the subject created several nodes and worked with them, but did not save the file.
R	Revision session: the subject edited the labels and content of existing nodes, but did not structurally change the document.
T	Trivial session: the subject was learning the system or just playing around with the features. No constructive work was observed. Trivial sessions are discussed further in Section 5.2.1 on page 94.

Table 40 (Page 1 of 4). Subjects, Sessions, and Documents.

Session ID (file name)	*Total Session Time (seconds)*	*Number of Commands*	*Nodes in Doc 1*	*Nodes in Doc 2*	*Nodes in Doc 3*	*Nodes in Doc 4*
Subject 1						
S01R0101	1973	226	56			
S01R0201	1128	82		17		
S01R0301	2293	153			29	
Subject 2						
S02R0101	1407	115	28			
S02R0201	875	72		19		
Subject 3						
S03R0101	2446	130	21			
Subject 4						
S04R0101	3245	230	60			
Subject 5						
S05R0101	1016	24	T			
S05R0201	999	45	14-E			
Subject 6						
S06R0101	2863	84	6			
S06R0102	121	4	1-E			
S06R0103	169	6	2-E			
S06R0104	1	2	T			
S06R0105	452	18	6			
S06R0106	557	10	3-E			

Table 40 (Page 1 of 4). Subjects, Sessions, and Documents.

Session ID (file name)	*Total Session Time (seconds)*	*Number of Commands*	*Nodes in Doc 1*	*Nodes in Doc 2*	*Nodes in Doc 3*	*Nodes in Doc 4*
S06R0107	337	9	2-E			
S06R0108	1180	21		4		
S06R0109	1719	143		18		
S06R0110	283	10		T		
S06R0201	902	67			12	
S06R0301	359	40				11
Subject 7						
S07R0101	365	16	T			
Subject 8						
S08R0101	1454	103	19			
S08R0201	2210	140	23	14		
S08R0301	1311	112			17	
Subject 9						
S09R0101	1516	27	T			
S09R0102	369	17	T			
Subject 10						
S10R0101	793	3	T			
S10R0102	5	2	T			
S10R0103	780	2	T			
S10R0104	3789	58	9			
S10R0105	1178	123		14	11	
Subject 11						
S11R0101	460	35	T			
S11R0102	286	18	T			
S11R0103	831	35	T			
S11R0104	119	12	2			
Subject 12						
S12R0101	2219	79	17			
Subject 13						
S13R0101	3803	68	7			
Subject 14						
S14R0101	18	2	T			
S14R0102	24	2	T			
S14R0103	1241	21	45-E			
S14R0104	630	5	T			
S14R0105	4124	83	46			
S14R0106	289	14	48			
S14R0107	5	2	T			
S14R0108	34	3	T			
S14R0109	4	2	T			
Subject 15						
S15R0101	1320	36	8			
S15R0102	8023	52	17			
S15R0103	282	10	5-E			
S15R0104	90	7	3-E			
S15R0105	100	4	T			

Table 40 (Page 2 of 4). Subjects, Sessions, and Documents.						
Session ID (file name)	*Total Session Time (seconds)*	*Number of Commands*	*Nodes in Doc 1*	*Nodes in Doc 2*	*Nodes in Doc 3*	*Nodes in Doc 4*
S15R0106	58	3	T			
S15R0107	25	2	T			
S15R0108	39	5	T			
S15R0201	2770	24		6		
S15R0202	6567	70		17		
S15R0203	2562	22		18		
S15R0204	100	6		T		
S15R0205	223	13		19		
Subject 16						
S16R0101	222	21	T			
S16R0102	2106	118	9			
S16R0103	47	8	1-E			
S16R0104	337	7	1-E			
S16R0105	221	9	2-E			
S16R0106	233	17	2-E			
S16R0107	1217	68	T			
S16R0108	1773	30		T		
S16R0109	363	17		T		
S16R0110	211	15		7-N		
S16R0111	3065	101			15	
S16R0112	203	9			T	
S16R0113	56	3			T	
S16R0114	219	15			T	
Subject 17						
S17R0101	13096	522	176			
S17R0201	1846	335		89		
Subject 18						
S18R0101	23	3	T			
S18R0102	146	11	4			
S18R0103	1296	66		11		
S18R0104	19	2		T		
S18R0105	54	4		1-E		
S18R0106	379	33		24		
S18R0107	705	29			6	
S18R0108	2356	153				27
Subject 19						
S19R0101	2555	265	47			
Subject 20						
S20R0101	653	66	16			
Subject 21						
S21R0101	42	1	T			
Subject 22						
S22R0101	2038	248	41			
S22R0201	1124	102		13		
S22R0301	2543	303			19	
Subject 23						

Table 40 (Page 3 of 4). Subjects, Sessions, and Documents.						
Session ID (file name)	*Total Session Time (seconds)*	*Number of Commands*	*Nodes in Doc 1*	*Nodes in Doc 2*	*Nodes in Doc 3*	*Nodes in Doc 4*
S23R0101	4308	138	20			
Subject 24						
S24R0101	2188	123	17			
Subject 25						
S25R0101	2974	43	8			
S25R0201	55	3	T			
S25R0301	11811	198		21		
Subject 26						
S26R0101	1953	105	20			
S26R0201	350	23	2-E			
Subject 27						
S27R0101	68	3	T			
S27R0102	825	5	T			
S27R0103	3815	143	21			
S27R0104	2750	48	5-E			
S27R0105	700	6	T			
S27R0106	4140	71	25			
S27R0107	142	4	T			
S27R0108	2825	2	T			
S27R0109	140	4	T			
Subject 28						
S28R0101	35	4	T			
S28R0102	3362	164	31			
S28R0103	8	2	T			
Subject 29						
S29R0101	1204	27	6			
S29R0102	441	14	8			
S29R0103	4440	91	12			

Trivial sessions	1 session/ doc	N sessions/ doc	Work on an existing doc	Multiple docs/session
S05R0101	S01R0101	S08R0101	S05R0201	S08R0201
S06R0104	S01R0201	S08R0201	S14R0103	S10R0105 (2 docs)
S06R0110	S01R0301	S16R0102	S14R0105	S29R0103
S07R0101	S02R0101	S16R0103	S14R0106	
S09R0101	S02R0201	S16R0104	S18R0102	
S09R0102	S03R0101	S16R0105		
S10R0101	S04R0101	S16R0106		
S10R0102	S06R0201	S16R0111		
S10R0103	S06R0301	S16R0113		
S11R0101	S08R0301	S18R0103		
S11R0102	S10R0104	S18R0105		
S11R0103	S12R0101	S18R0106		
S11R0104	S17R0101	S29R0101		
S14R0101	S17R0201	S29R0102		
S14R0102	S18R0107	S29R0103		
S14R0104	S19R0101			
S14R0107	S20R0101			
S14R0108	S22R0101	S06R0101		
S14R0109	S22R0201	S06R0102		
S15R0105	S22R0301	S06R0103		
S15R0106	S23R0101	S06R0105		
S15R0107	S24R0101	S06R0106		
S15R0108		S06R0107		
S16R0101		S06R0108		
S16R0107	S13R0101	S06R0109		
S16R0108	S18R0108	S15R0101		
S16R0109	S25R0101	S15R0102		
S16R0110	S25R0301	S15R0103		
S16R0112	S28R0102	S15R0104		
S16R0114		S15R0201		
S18R0101		S15R0202		
S18R0104		S15R0203		
S21R0101		S15R0204		
S25R0201		S15R0205		
S27R0101		S26R0101		
S27R0102		S26R0201		
S27R0105		S27R0103		
S27R0107		S27R0104		
S27R0108		S27R0106		
S27R0109				
S28R0101				
S28R0103				
42 sessions	27 sessions	37 sessions	5 sessions	3 sessions
0 documents	27 documents	11 documents	3 documents	4 documents

Figure 32. Categorization of the 112 sessions into 5 groups. Among the 112 sessions, a total of 45 different, substantial documents were constructed or edited. Sessions in the righthand four categories are considered "non-trivial." The columns in this table are explained further below.

S06R0101	S18R0108
S13R0101	S25R0101
S15R0102	S25R0301
S15R0201	S27R0106
S15R0202	S28R0102

Figure 33. 10 sessions that contained substantial writing. Among the 112 sessions, these 10 sessions were significant in that the subjects wrote substantial text in the file behind the nodes.

APPENDIX B. EXAMPLE PARSER INPUT AND OUTPUT

Session S18R0108 was a classic among stand-alone sessions. All the nodes were labeled and linked together, the structure of the tree was touched up, then the contents of the nodes were edited. This session dealt with 27 nodes during 2356 seconds (39.3 minutes), and it consisted of 175 commands. The mean length of the labels was 11.5 characters.

The following is my summary of the session.

- (create and label) 4 nodes
- link into a tree
- tidy
- create/label, create/label, 2 link
- create/delete
- create/label, create/label, 2 link
- tidy
- create, label, link, tidy
- (create, label, and link) 2 nodes, tidy
- ((create and label) 2 nodes, links, tidy) 5 times
- (create, label, link, and tidy) 3 nodes
- breaks and links
- Edit: 5, 6, 5, 8, 9, 12, 13, 10, 14, 15, 16, 17, 22, 23, 19, 20, 21, 25, 26, 27

B.1 A Sample Protocol Recording

Including the header, the protocol record file is 456 lines in length.

```
+---------------------------------------------------------------------------+
| Prose II Session Recording (v2.09)              Mon Feb 13 22:52:46 1989 |
| File C: RCD3B06.TMP                                                       |
+---------------------------------------------------------------------------+
| Start     Stop       Time  Operator            Parameters                 |
|min:sec   min:sec     sec                                                  |
+---------------------------------------------------------------------------+
52:46.04  52:48.85    2.81  PAUSE
52:48.85  52:49.62    0.77  LeaveProseII        ID(0)
52:49.62  52:50.05    0.43  PAUSE
52:50.05  52:52.58    2.53  PAUSE
52:52.58  53:05.71   13.13  OpenWorkspace       File(`QUEUEMGR.PR2`) Format(`.PR2`)
                            + CreateNode        ID(1)
                            + CreateNode        ID(2)
                            + LinkNodes         ParentID(1) ChildID(2)
                            + CreateNode        ID(3)
                            + LinkNodes         ParentID(2) ChildID(3)
                            + CreateNode        ID(4)
                            + LinkNodes         ParentID(2) ChildID(4)
                            + CreateNode        ID(5)
                            + LinkNodes         ParentID(2) ChildID(5)
                            + CreateNode        ID(6)
                            + LinkNodes         ParentID(2) ChildID(6)
                            + CreateNode        ID(7)
                            + LinkNodes         ParentID(2) ChildID(7)
                            + CreateNode        ID(8)
                            + LinkNodes         ParentID(2) ChildID(8)
                            + CreateNode        ID(9)
                            + LinkNodes         ParentID(2) ChildID(9)
                            + CreateNode        ID(10)
                            + LinkNodes         ParentID(2) ChildID(10)
                            + CreateNode        ID(11)
                            + LinkNodes         ParentID(2) ChildID(11)
                            + CreateNode        ID(12)
                            + LinkNodes         ParentID(2) ChildID(12)
                            + CreateNode        ID(13)
                            + LinkNodes         ParentID(2) ChildID(13)
                            + CreateNode        ID(14)
                            + LinkNodes         ParentID(2) ChildID(14)
                            + CreateNode        ID(15)
                            + LinkNodes         ParentID(2) ChildID(15)
                            + CreateNode        ID(16)
                            + LinkNodes         ParentID(2) ChildID(16)
                            + CreateNode        ID(17)
                            + LinkNodes         ParentID(1) ChildID(17)
                            + CreateNode        ID(18)
                            + LinkNodes         ParentID(1) ChildID(18)
                            + CreateNode        ID(19)
                            + LinkNodes         ParentID(1) ChildID(19)
                            + CreateNode        ID(20)
                            + LinkNodes         ParentID(19) ChildID(24)
                            + CreateNode        ID(25)
                            + LinkNodes         ParentID(19) ChildID(20)
                            + CreateNode        ID(26)
                            + LinkNodes         ParentID(19) ChildID(21)
                            + CreateNode        ID(27)
                            + LinkNodes         ParentID(19) ChildID(22)
                            + CreateNode        ID(28)
                            + LinkNodes         ParentID(19) ChildID(23)
53:05.71  53:12.41    6.70  PAUSE
53:12.41  53:20.70    8.29  PAUSE
53:20.70  53:20.76    0.06  MapMove             StartRect(0, 0, 550, -216) EndRect(10, 36, 210, 136)
53:20.76  53:21.09    0.33  PAUSE
53:21.09  53:21.14    0.05  MapWindow           OPEN
53:21.14  53:24.05    2.91  PAUSE
53:24.05  53:25.70    1.65  MapWindowZoom       StartRect(0, 0, 550, -216) EndRect(-40, 494, 768, -480)
53:25.70  53:47.07   21.37  PAUSE
53:47.07  53:47.12    0.05  NewWorkspace
                            + DeleteNode        ID(1)
```

```
70                               + DeleteNode       ID(2)
71                               + DeleteNode       ID(3)
72                               + DeleteNode       ID(4)
73                               + DeleteNode       ID(5)
74                               + DeleteNode       ID(6)
75                               + DeleteNode       ID(7)
76                               + DeleteNode       ID(8)
77                               + DeleteNode       ID(9)
78                               + DeleteNode       ID(10)
79                               + DeleteNode       ID(11)
80                               + DeleteNode       ID(12)
81                               + DeleteNode       ID(13)
82                               + DeleteNode       ID(14)
83                               + DeleteNode       ID(15)
84                               + DeleteNode       ID(16)
85                               + BreakLink        ParentID(2) ChildID(16)
86                               + BreakLink        ParentID(2) ChildID(15)
87                               + BreakLink        ParentID(2) ChildID(14)
88                               + BreakLink        ParentID(2) ChildID(13)
89                               + BreakLink        ParentID(2) ChildID(12)
90                               + BreakLink        ParentID(2) ChildID(11)
91                               + BreakLink        ParentID(2) ChildID(10)
92                               + BreakLink        ParentID(2) ChildID(9)
93                               + BreakLink        ParentID(2) ChildID(8)
94                               + BreakLink        ParentID(2) ChildID(7)
95                               + BreakLink        ParentID(2) ChildID(6)
96                               + BreakLink        ParentID(2) ChildID(5)
97                               + BreakLink        ParentID(2) ChildID(4)
98                               + BreakLink        ParentID(2) ChildID(3)
99                               + DeleteNode       ID(17)
100                              + DeleteNode       ID(18)
101                              + DeleteNode       ID(19)
102                              + DeleteNode       ID(24)
103                              + DeleteNode       ID(20)
104                              + DeleteNode       ID(21)
105                              + DeleteNode       ID(22)
106                              + DeleteNode       ID(23)
107                              + BreakLink        ParentID(19) ChildID(23)
108                              + BreakLink        ParentID(19) ChildID(22)
109                              + BreakLink        ParentID(19) ChildID(21)
110                              + BreakLink        ParentID(19) ChildID(20)
111                              + BreakLink        ParentID(19) ChildID(24)
112                              + BreakLink        ParentID(1) ChildID(19)
113                              + BreakLink        ParentID(1) ChildID(18)
114                              + BreakLink        ParentID(1) ChildID(17)
115                              + BreakLink        ParentID(1) ChildID(2)
116                              + MainWindowReset
117  53:47.12  54:07.33   20.21  PAUSE
118  54:07.33  54:07.44    0.11  CreateNode         ID(1) StartPt(293, -29)
119  54:07.44  54:13.43    5.99  EditLabel          ID(1) NewText(`xxxxx xxxxxxx`)
120  54:13.43  54:19.20    5.77  PAUSE
121  54:19.20  54:19.36    0.16  CreateNode         ID(2) StartPt(99, -84)
122  54:19.36  54:26.78    7.42  EditLabel          ID(2) NewText(`xxxxxxxxxxxxxx xxxxxxxxx`)
123  54:26.78  54:29.25    2.47  PAUSE
124  54:29.25  54:29.36    0.11  CreateNode         ID(3) StartPt(257, -80)
125  54:29.36  54:46.83   17.47  EditLabel          ID(3) NewText(`xxxxxxx xxxxxxxxxxx xxxxxxxxx`)
126  54:46.83  54:49.02    2.19  PAUSE
127  54:49.02  54:49.13    0.11  CreateNode         ID(4) StartPt(396, -81)
128  54:49.13  54:59.02    9.89  EditLabel          ID(4) NewText(`xxxxxxx xxxxxxxxxxxx xxxxxxxxx`)
129  54:59.02  55:18.02   19.00  PAUSE
130  55:18.02  55:18.79    0.77  LinkNodes          ParentID(1) ChildID(2)
131  55:18.79  55:19.67    0.88  PAUSE
132  55:19.67  55:20.22    0.55  LinkNodes          ParentID(1) ChildID(3)
133  55:20.22  55:20.93    0.71  PAUSE
134  55:20.93  55:21.65    0.72  LinkNodes          ParentID(1) ChildID(4)
135  55:21.65  55:24.29    2.64  PAUSE
136  55:24.29  55:24.34    0.05  TidyWorkspace
137  55:24.34  55:28.40    4.06  PAUSE
138  55:28.40  55:28.57    0.17  CreateNode         ID(5) StartPt(24, -179)
139  55:28.57  55:34.06    5.49  EditLabel          ID(5) NewText(`x_xxxx`)
140  55:34.06  55:36.59    2.53  PAUSE
141  55:36.59  55:42.47    5.88  PAUSE
142  55:42.47  55:42.63    0.16  CreateNode         ID(6) StartPt(152, -176)
143  55:42.63  55:51.86    9.23  EditLabel          ID(6) NewText(`x_xxxx_xxx`)
144  55:51.86  55:53.18    1.32  PAUSE
```

```
145  55:53.18  55:53.78   0.60  LinkNodes          ParentID(2) ChildID(5)
146  55:53.78  55:54.60   0.82  PAUSE
147  55:54.60  55:55.37   0.77  LinkNodes          ParentID(2) ChildID(6)
148  55:55.37  55:56.75   1.38  PAUSE
149  55:56.75  55:56.86   0.11  CreateNode         ID(7) StartPt(279, -181)
150  55:56.86  56:06.30   9.44  EditLabel          ID(7) -- cancelled --
151  56:06.30  56:10.15   3.85  PAUSE
152  56:10.15  56:10.20   0.05  DeleteNode         ID(7)
153  56:10.20  56:12.29   2.09  PAUSE
154  56:12.29  56:12.40   0.11  CreateNode         ID(8) StartPt(279, -180)
155  56:12.40  56:23.00  10.60  EditLabel          ID(8) NewText(`x_xxx`)
156  56:23.00  56:24.54   1.54  PAUSE
157  56:24.54  56:24.65   0.11  CreateNode         ID(9) StartPt(399, -175)
158  56:24.65  56:32.72   8.07  EditLabel          ID(9) NewText(`x_xxxxxx`)
159  56:32.72  56:38.71   5.99  PAUSE
160  56:38.71  56:39.42   0.71  LinkNodes          ParentID(3) ChildID(8)
161  56:39.42  56:40.14   0.72  PAUSE
162  56:40.14  56:40.69   0.55  LinkNodes          ParentID(3) ChildID(9)
163  56:40.69  56:43.16   2.47  PAUSE
164  56:43.16  56:43.21   0.05  TidyWorkspace
165  56:43.21  56:49.04   5.83  PAUSE
166  56:49.04  56:49.09   0.05  CreateNode         ID(10) StartPt(557, -187)
167  56:49.09  57:11.72  22.63  EditLabel          ID(10) NewText(`x_xxxx`)
168  57:11.72  57:13.64   1.92  PAUSE
169  57:13.64  57:14.58   0.94  LinkNodes          ParentID(3) ChildID(10)
170  57:14.58  57:17.60   3.02  PAUSE
171  57:17.60  57:17.65   0.05  TidyWorkspace
172  57:17.65  57:21.06   3.41  PAUSE
173  57:21.06  57:21.11   0.05  MapWindow          OPEN
174  57:21.11  57:23.31   2.20  PAUSE
175  57:23.31  57:24.13   0.82  MapWindowZoom      StartRect(0, 0, 608, -258) EndRect(140, -9, 692, -238)
176  57:24.13  57:34.84  10.71  PAUSE
177  57:34.84  57:36.11   1.27  MapWindowZoom      StartRect(140, -47, 692, -276) EndRect(-92, 64, 876, -311)
178  57:36.11  57:41.60   5.49  PAUSE
179  57:41.60  57:41.65   0.05  CreateNode         ID(11) StartPt(717, -182)
180  57:41.65  57:47.59   5.94  EditLabel          ID(11) NewText(`x_xxxx_xxx`)
181  57:47.59  57:49.40   1.81  PAUSE
182  57:49.40  57:50.28   0.88  LinkNodes          ParentID(3) ChildID(11)
183  57:50.28  57:51.87   1.59  PAUSE
184  57:51.87  57:51.98   0.11  CreateNode         ID(12) StartPt(839, -177)
185  57:51.98  58:05.11  13.13  EditLabel          ID(12) NewText(`x_xxxx`)
186  58:05.11  58:06.59   1.48  PAUSE
187  58:06.59  58:07.52   0.93  LinkNodes          ParentID(3) ChildID(12)
188  58:07.52  58:10.54   3.02  PAUSE
189  58:10.54  58:10.60   0.06  TidyWorkspace
190  58:10.60  58:13.73   3.13  PAUSE
191  58:13.73  58:15.05   1.32  MapWindowZoom      StartRect(-92, 64, 876, -311) EndRect(5, 128, 1196, -311)
192  58:15.05  58:18.07   3.02  PAUSE
193  58:18.07  58:18.12   0.05  CreateNode         ID(13) StartPt(926, -173)
194  58:18.12  58:25.05   6.93  EditLabel          ID(13) NewText(`x_xxxxxxxx`)
195  58:25.05  58:26.53   1.48  PAUSE
196  58:26.53  58:26.64   0.11  CreateNode         ID(14) StartPt(1063, -180)
197  58:26.64  58:43.45  16.81  EditLabel          ID(14) NewText(`x_xxxxx`)
198  58:43.45  58:45.42   1.97  PAUSE
199  58:45.42  58:46.52   1.10  LinkNodes          ParentID(4) ChildID(14)
200  58:46.52  58:47.51   0.99  PAUSE
201  58:47.51  58:48.28   0.77  LinkNodes          ParentID(3) ChildID(13)
202  58:48.28  58:51.35   3.07  PAUSE
203  58:51.35  58:51.41   0.06  TidyWorkspace
204  58:51.41  58:54.81   3.40  PAUSE
205  58:54.81  58:54.92   0.11  CreateNode         ID(15) StartPt(1016, -182)
206  58:54.92  58:58.49   3.57  EditLabel          ID(15) NewText(`x_xxxx`)
207  58:58.49  59:00.20   1.71  PAUSE
208  59:00.20  59:00.31   0.11  CreateNode         ID(16) StartPt(1122, -178)
209  59:00.31  59:03.88   3.57  EditLabel          ID(16) NewText(`x_xxxx`)
210  59:03.88  59:05.36   1.48  PAUSE
211  59:05.36  59:06.02   0.66  LinkNodes          ParentID(4) ChildID(15)
212  59:06.02  59:06.90   0.88  PAUSE
213  59:06.90  59:07.89   0.99  LinkNodes          ParentID(4) ChildID(16)
214  59:07.89  59:10.08   2.19  PAUSE
215  59:10.08  59:10.14   0.06  TidyWorkspace
216  59:10.14  59:18.65   8.51  PAUSE
217  59:18.65  59:18.82   0.17  CreateNode         ID(17) StartPt(1223, -180)
218  59:18.82  59:22.44   3.62  EditLabel          ID(17) NewText(`x_xxxx`)
219  59:22.44  59:24.42   1.98  PAUSE
```

```
220  59:24.42  59:24.53   0.11  CreateNode          ID(18) StartPt(1364, -180)
221  59:24.53  59:52.93  28.40  EditLabel           ID(18) NewText(`xxxxx xxxxxxxxxxx xxxxxxxxx`)
222  59:52.93  59:56.00   3.07  PAUSE
223  59:56.00  59:56.94   0.94  MapWindowZoom       StartRect(300, 128, 1491, -311) EndRect(-39, 46, 1605, -302)
224  59:56.94  00:04.02   7.08  PAUSE
225  00:04.02  00:05.45   1.43  LinkNodes           ParentID(1) ChildID(18)
226  00:05.45  00:08.03   2.58  PAUSE
227  00:08.03  00:08.09   0.06  TidyWorkspace
228  00:08.09  00:11.88   3.79  PAUSE
229  00:11.88  00:12.81   0.93  LinkNodes           ParentID(4) ChildID(17)
230  00:12.81  00:14.79   1.98  PAUSE
231  00:14.79  00:14.84   0.05  TidyWorkspace
232  00:14.84  00:20.94   6.10  PAUSE
233  00:20.94  00:21.05   0.11  CreateNode          ID(19) StartPt(1329, -187)
234  00:21.05  00:30.71   9.66  EditLabel           ID(19) NewText(`x_xxxxx`)
235  00:30.71  00:32.31   1.60  PAUSE
236  00:32.31  00:32.42   0.11  CreateNode          ID(20) StartPt(1467, -183)
237  00:32.42  00:37.58   5.16  EditLabel           ID(20) NewText(`x_xxxx`)
238  00:37.58  00:40.44   2.86  PAUSE
239  00:40.44  00:41.15   0.71  LinkNodes           ParentID(18) ChildID(19)
240  00:41.15  00:42.03   0.88  PAUSE
241  00:42.03  00:43.07   1.04  LinkNodes           ParentID(18) ChildID(20)
242  00:43.07  00:45.76   2.69  PAUSE
243  00:45.76  00:45.82   0.06  TidyWorkspace
244  00:45.82  01:01.64  15.82  PAUSE
245  01:01.64  01:02.90   1.26  MapWindowZoom       StartRect(79, 46, 1723, -302) EndRect(597, -55, 1677, -320)
246  01:02.90  01:33.06  30.16  PAUSE
247  01:33.06  01:33.16   0.10  CreateNode          ID(21) StartPt(1422, -179)
248  01:33.16  01:35.86   2.70  EditLabel           ID(21) NewText(`x_xxxx`)
249  01:35.86  01:37.23   1.37  PAUSE
250  01:37.23  01:37.89   0.66  LinkNodes           ParentID(18) ChildID(21)
251  01:37.89  02:01.51  23.62  PAUSE
252  02:01.51  02:01.62   0.11  CreateNode          ID(22) StartPt(1033, -137)
253  02:01.62  02:06.28   4.66  EditLabel           ID(22) NewText(`x_xxxx_xxx`)
254  02:06.28  02:07.66   1.38  PAUSE
255  02:07.66  02:08.37   0.71  LinkNodes           ParentID(4) ChildID(22)
256  02:08.37  02:11.17   2.80  PAUSE
257  02:11.17  02:11.23   0.06  TidyWorkspace
258  02:11.23  02:14.69   3.46  PAUSE
259  02:14.69  02:14.80   0.11  CreateNode          ID(23) StartPt(1077, -145)
260  02:14.80  02:18.81   4.01  EditLabel           ID(23) NewText(`x_xxxx_xxx`)
261  02:18.81  02:20.18   1.37  PAUSE
262  02:20.18  02:20.95   0.77  LinkNodes           ParentID(4) ChildID(23)
263  02:20.95  02:23.97   3.02  PAUSE
264  02:23.97  02:24.03   0.06  TidyWorkspace
265  02:24.03  02:32.76   8.73  PAUSE
266  02:32.76  02:33.31   0.55  MapWindowZoom       StartRect(420, -55, 1500, -320) EndRect(773, 128, 1887, -247)
267  02:33.31  02:37.37   4.06  PAUSE
268  02:37.37  02:37.43   0.06  CreateNode          ID(24) StartPt(1724, -97)
269  02:37.43  02:44.29   6.86  EditLabel           ID(24) NewText(`xxxxx xxxxxxxxxxxxx xxxxxxxxx`)
270  02:44.29  02:48.08   3.79  PAUSE
271  02:48.08  02:48.96   0.88  MapWindowZoom       StartRect(773, 128, 1887, -247) EndRect(-18, 119, 1967, -347)
272  02:48.96  02:51.82   2.86  PAUSE
273  02:51.82  02:53.19   1.37  LinkNodes           ParentID(1) ChildID(24)
274  02:53.19  02:56.98   3.79  PAUSE
275  02:56.98  02:57.04   0.06  TidyWorkspace
276  02:57.04  03:01.59   4.55  PAUSE
277  03:01.59  03:01.65   0.06  CreateNode          ID(25) StartPt(1657, -179)
278  03:01.65  03:06.70   5.05  EditLabel           ID(25) NewText(`x_xxxx`)
279  03:06.70  03:07.97   1.27  PAUSE
280  03:07.97  03:08.68   0.71  LinkNodes           ParentID(24) ChildID(25)
281  03:08.68  03:09.34   0.66  PAUSE
282  03:09.34  03:09.45   0.11  CreateNode          ID(26) StartPt(1817, -181)
283  03:09.45  03:16.53   7.08  EditLabel           ID(26) NewText(`x_xxxxx`)
284  03:16.53  03:18.46   1.93  PAUSE
285  03:18.46  03:18.62   0.16  CreateNode          ID(27) StartPt(1885, -118)
286  03:18.62  03:26.31   7.69  EditLabel           ID(27) NewText(`x_xxxxxx`)
287  03:26.31  03:27.63   1.32  PAUSE
288  03:27.63  03:28.56   0.93  LinkNodes           ParentID(24) ChildID(26)
289  03:28.56  03:29.55   0.99  PAUSE
290  03:29.55  03:30.54   0.99  LinkNodes           ParentID(24) ChildID(27)
291  03:30.54  03:33.95   3.41  PAUSE
292  03:33.95  03:34.00   0.05  TidyWorkspace
293  03:34.00  03:40.98   6.98  PAUSE
294  03:40.98  03:41.97   0.99  MapWindowZoom       StartRect(-18, 119, 1967, -347) EndRect(-1423, -46, 1932, -265)
```

```
295  03:41.97  04:27.94   45.97  PAUSE
296  04:27.94  04:28.98    1.04  MapWindowZoom        StartRect(-1423, -46, 1932, -265) EndRect(-1314, -46, 244, -311)
297  04:28.98  04:33.43    4.45  PAUSE
298  04:33.43  04:34.80    1.37  MapWindowZoom        StartRect(-1314, -46, 244, -311) EndRect(-1379, 275, 201, -311)
299  04:34.80  04:45.73   10.93  PAUSE
300  04:45.73  04:46.56    0.83  MapWindowZoom        StartRect(-1379, 275, 201, -311) EndRect(-1011, -27, 309, -256)
301  04:46.56  05:04.63   18.07  PAUSE
302  05:04.63  05:04.68    0.05  BreakLink            ParentID(3) ChildID(10)
303  05:04.68  05:06.11    1.43  PAUSE
304  05:06.11  05:06.77    0.66  BreakLink            ParentID(3) ChildID(11)
305  05:06.77  05:11.82    5.05  PAUSE
306  05:11.82  05:11.93    0.11  SetDeleteMode        ON
307  05:11.93  05:12.70    0.77  PAUSE
308  05:12.70  05:12.76    0.06  DeleteNode           ID(11)
309  05:12.76  05:18.47    5.71  PAUSE
310  05:18.47  05:18.52    0.05  SetDeleteMode        OFF
311  05:18.52  05:21.65    3.13  PAUSE
312  05:21.65  05:22.81    1.16  MapWindowZoom        StartRect(-1011, -27, 309, -256) EndRect(-860, 0, 1131, -293)
313  05:22.81  05:26.65    3.84  PAUSE
314  05:26.65  05:27.26    0.61  LinkNodes            ParentID(4) ChildID(10)
315  05:27.26  05:30.06    2.80  PAUSE
316  05:30.06  05:30.11    0.05  TidyWorkspace
317  05:30.11  06:04.66   34.55  PAUSE
318  06:04.66  06:13.23    8.57  SaveWorkspace        File(`QMGR`) Format(`.PR2`)
319  06:13.23  06:24.43   11.20  PAUSE
320  06:24.43  06:27.13    2.70  MapWindowZoom        StartRect(-860, 0, 1131, -293) EndRect(-1318, 211, -87, -366)
321  06:27.13  06:39.21   12.08  PAUSE
322  06:39.21  06:39.32    0.11  CreateNode           ID(28) StartPt(-1063, -321)
323  06:39.32  06:44.32    5.00  PAUSE
324  06:44.32  06:44.37    0.05  DeleteNode           ID(28)
325  06:44.37  06:56.84   12.47  PAUSE
326  06:56.84  06:59.26    2.42  MainWindowZoom       StartRect(-1318, 211, -87, -366) EndRect(-1146, -10, -779, -259)
327  06:59.26  07:01.89    2.63  PAUSE
328  07:01.89  07:06.45    4.56  PAUSE
329  07:06.45  07:11.56    5.11  PAUSE
330  07:11.56  07:11.62    0.06  PAUSE
331  07:11.62  08:19.12   67.50  LeaveProseII        ID(0)
332  08:19.12  08:19.83    0.71  EditNode             ID(5) Editor(`c:\windows\pif\t.pif`) File(`QMGR005.SCR`)
333  08:19.83  08:24.56    4.73  PAUSE
334  08:24.56  08:28.18    3.62  PAUSE
335  08:28.18  08:32.96    4.78  PAUSE
336  08:32.96  10:32.92  119.96  LeaveProseII        ID(0)
337  10:32.92  10:33.85    0.93  EditNode             ID(6) Editor(`c:\windows\pif\t.pif`) File(`QMGR006.SCR`)
338  10:33.85  10:37.92    4.07  PAUSE
339  10:37.92  10:40.61    2.69  PAUSE
340  10:40.61  10:45.28    4.67  PAUSE
341  10:45.28  10:45.33    0.05  PAUSE
342  10:45.33  11:47.01   61.68  LeaveProseII        ID(0)
343  11:47.01  11:47.89    0.88  EditNode             ID(5) Editor(`c:\windows\pif\t.pif`) File(`QMGR005.SCR`)
344  11:47.89  12:03.00   15.11  PAUSE
345  12:03.00  12:07.44    4.44  PAUSE
346  12:07.44  12:12.28    4.84  PAUSE
347  12:12.28  12:12.33    0.05  PAUSE
348  12:12.33  12:40.29   27.96  LeaveProseII        ID(0)
349  12:40.29  12:41.00    0.71  EditNode             ID(8) Editor(`c:\windows\pif\t.pif`) File(`QMGR008.SCR`)
350  12:41.00  12:45.51    4.51  PAUSE
351  12:45.51  12:49.57    4.06  PAUSE
352  12:49.57  12:54.41    4.84  PAUSE
353  12:54.41  13:47.90   53.49  LeaveProseII        ID(0)
354  13:47.90  13:48.73    0.83  EditNode             ID(9) Editor(`c:\windows\pif\t.pif`) File(`QMGR009.SCR`)
355  13:48.73  13:54.22    5.49  PAUSE
356  13:54.22  13:57.57    3.35  PAUSE
357  13:57.57  14:02.13    4.56  PAUSE
358  14:02.13  15:26.16   84.03  LeaveProseII        ID(0)
359  15:26.16  15:26.88    0.72  EditNode             ID(12) Editor(`c:\windows\pif\t.pif`) File(`QMGR00C.SCR`)
360  15:26.88  15:35.12    8.24  PAUSE
361  15:35.12  15:40.12    5.00  PAUSE
362  15:40.12  15:45.33    5.21  PAUSE
363  15:45.33  16:19.00   33.67  LeaveProseII        ID(0)
364  16:19.00  16:19.72    0.72  EditNode             ID(13) Editor(`c:\windows\pif\t.pif`) File(`QMGR00D.SCR`)
365  16:19.72  16:24.22    4.50  PAUSE
366  16:24.22  16:27.96    3.74  PAUSE
367  16:27.96  16:33.23    5.27  PAUSE
368  16:33.23  17:29.31   56.08  LeaveProseII        ID(0)
369  17:29.31  17:30.02    0.71  EditNode             ID(10) Editor(`c:\windows\pif\t.pif`) File(`QMGR00A.SCR`)
```

```
370  17:30.02  17:34.64   4.62  PAUSE
371  17:34.64  17:39.47   4.83  PAUSE
372  17:39.47  17:44.47   5.00  PAUSE
373  17:44.47  18:12.59  28.12  LeaveProseII       ID(0)
374  18:12.59  18:13.63   1.04  EditNode           ID(14) Editor(`c:\windows\pif\t.pif`) File(`QMGR00E.SCR`)
375  18:13.63  18:22.42   8.79  PAUSE
376  18:22.42  18:28.90   6.48  PAUSE
377  18:28.90  18:34.23   5.33  PAUSE
378  18:34.23  19:29.16  54.93  LeaveProseII       ID(0)
379  19:29.16  19:29.92   0.76  EditNode           ID(15) Editor(`c:\windows\pif\t.pif`) File(`QMGR00F.SCR`)
380  19:29.92  19:36.13   6.21  PAUSE
381  19:36.13  19:39.92   3.79  SaveWorkspace      File(`QMGR.PR2`) Format(`.PR2`)
382  19:39.92  19:42.34   2.42  PAUSE
383  19:42.34  19:43.22   0.88  PAUSE
384  19:43.22  19:46.73   3.51  PAUSE
385  19:46.73  19:51.78   5.05  PAUSE
386  19:51.78  20:14.96  23.18  LeaveProseII       ID(0)
387  20:14.96  20:16.17   1.21  EditNode           ID(16) Editor(`c:\windows\pif\t.pif`) File(`QMGR010.SCR`)
388  20:16.17  20:23.09   6.92  PAUSE
389  20:23.09  20:26.11   3.02  PAUSE
390  20:26.11  20:31.22   5.11  PAUSE
391  20:31.22  21:22.14  50.92  LeaveProseII       ID(0)
392  21:22.14  21:23.02   0.88  EditNode           ID(17) Editor(`c:\windows\pif\t.pif`) File(`QMGR011.SCR`)
393  21:23.02  21:27.74   4.72  PAUSE
394  21:27.74  21:30.49   2.75  PAUSE
395  21:30.49  21:35.81   5.32  PAUSE
396  21:35.81  22:16.24  40.43  LeaveProseII       ID(0)
397  22:16.24  22:17.01   0.77  EditNode           ID(22) Editor(`c:\windows\pif\t.pif`) File(`QMGR016.SCR`)
398  22:17.01  22:22.06   5.05  PAUSE
399  22:22.06  22:25.52   3.46  PAUSE
400  22:25.52  22:30.85   5.33  PAUSE
401  22:30.85  23:32.69  61.84  LeaveProseII       ID(0)
402  23:32.69  23:33.52   0.83  EditNode           ID(23) Editor(`c:\windows\pif\t.pif`) File(`QMGR017.SCR`)
403  23:33.52  23:38.35   4.83  PAUSE
404  23:38.35  23:50.22  11.87  PAUSE
405  23:50.22  23:55.43   5.21  PAUSE
406  23:55.43  24:41.90  46.47  LeaveProseII       ID(0)
407  24:41.90  24:42.62   0.72  EditNode           ID(19) Editor(`c:\windows\pif\t.pif`) File(`QMGR013.SCR`)
408  24:42.62  24:46.62   4.00  PAUSE
409  24:46.62  24:49.65   3.03  PAUSE
410  24:49.65  24:54.64   4.99  PAUSE
411  24:54.64  24:54.70   0.06  PAUSE
412  24:54.70  26:05.17  70.47  LeaveProseII       ID(0)
413  26:05.17  26:06.32   1.15  EditNode           ID(20) Editor(`c:\windows\pif\t.pif`) File(`QMGR014.SCR`)
414  26:06.32  26:12.80   6.48  PAUSE
415  26:12.80  26:16.10   3.30  PAUSE
416  26:16.10  26:21.59   5.49  PAUSE
417  26:21.59  27:45.35  83.76  LeaveProseII       ID(0)
418  27:45.35  27:46.07   0.72  EditNode           ID(21) Editor(`c:\windows\pif\t.pif`) File(`QMGR015.SCR`)
419  27:46.07  27:50.79   4.72  PAUSE
420  27:50.79  27:53.43   2.64  PAUSE
421  27:53.43  27:58.64   5.21  PAUSE
422  27:58.64  28:26.82  28.18  LeaveProseII       ID(0)
423  28:26.82  28:27.75   0.93  EditNode           ID(25) Editor(`c:\windows\pif\t.pif`) File(`QMGR019.SCR`)
424  28:27.75  28:32.20   4.45  PAUSE
425  28:32.20  28:35.66   3.46  PAUSE
426  28:35.66  28:40.77   5.11  PAUSE
427  28:40.77  29:19.33  38.56  LeaveProseII       ID(0)
428  29:19.33  29:20.26   0.93  EditNode           ID(26) Editor(`c:\windows\pif\t.pif`) File(`QMGR01A.SCR`)
429  29:20.26  29:24.00   3.74  PAUSE
430  29:24.00  29:26.74   2.74  PAUSE
431  29:26.74  29:32.02   5.28  PAUSE
432  29:32.02  29:32.07   0.05  PAUSE
433  29:32.07  29:32.13   0.06  PAUSE
434  29:32.13  30:27.16  55.03  LeaveProseII       ID(0)
435  30:27.16  30:27.99   0.83  EditNode           ID(27) Editor(`c:\windows\pif\t.pif`) File(`QMGR01B.SCR`)
436  30:27.99  30:33.42   5.43  PAUSE
437  30:33.42  30:33.48   0.06  MapMove            StartRect(-1318, 211, -87, -366) EndRect(418, 40, 618, 140)
438  30:33.48  30:33.81   0.33  PAUSE
439  30:33.81  30:33.86   0.05  MapWindow          OPEN
440  30:33.86  30:36.12   2.26  PAUSE
441  30:36.12  30:38.26   2.14  MapWindowZoom      StartRect(1482, -10, 1849, -259) EndRect(-1276, 183, 1832, -329)
442  30:38.26  30:44.08   5.82  PAUSE
443  30:44.08  30:45.34   1.26  MapWindowZoom      StartRect(-1276, 183, 1832, -329) EndRect(-1276, -110, 1936, -256)
444  30:45.34  30:49.30   3.96  PAUSE
```

```
445  30:49.30  30:50.12    0.82  MapWindowZoom        StartRect(-1276, -110, 1936, -256) EndRect(-650, 19, -171, -293)
446  30:50.12  30:57.98    7.86  PAUSE
447  30:57.98  31:02.10    4.12  SaveWorkspace        File(`QMGR.PR2`) Format(`.PR2`)
448  31:02.10  31:08.74    6.64  PAUSE
449  31:08.74  31:08.80    0.06  OutlineWindowMove    StartRect(-1276, -110, 1936, -256) EndRect(10, 36, 245, 236)
450  31:08.80  31:09.24    0.44  PAUSE
451  31:09.24  31:09.29    0.05  OutlineWindow        OPEN
452  31:09.29  31:20.88   11.59  PAUSE
453  31:20.88  31:37.80   16.92  SaveWorkspace        File(`QMGR`) Format(`.SCR`)
454  31:37.80  31:41.92    4.12  PAUSE
455  31:41.92  31:59.11   17.19  SaveWorkspace        File(`QMGR`) Format(`.PR2`)
456  31:59.11  32:02.79    3.68  PAUSE
```

B.2 Output from Pass 0

Pass 0 does not generate the header, so the output from Pass 0 is 449 lines in length.

```
0257,  0316604,  0316885,  0000281, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0275,  0316885,  0316962,  0000077, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0257,  0316962,  0317005,  0000043, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0257,  0317005,  0317258,  0000253, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0286,  0317258,  0318571,  0001313, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 1 0 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 2 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 1 2 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 3 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 3 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 4 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 4 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 5 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 5 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 6 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 6 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 7 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 7 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 8 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 8 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 9 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 9 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 10 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 10 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 11 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 11 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 12 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 12 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 13 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 13 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 14 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 14 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 15 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 15 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 16 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 16 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 17 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 1 17 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 18 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 1 18 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 19 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 1 19 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 20 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 19 24 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 25 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 19 20 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 26 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 19 21 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 27 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 19 22 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 28 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 19 23 0 0 0 0 0 0 0 0
0257,  0318571,  0319241,  0000670, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0257,  0319241,  0320070,  0000829, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0279,  0320070,  0320076,  0000006, 0001, 0000 0 0 0 0 550 -216 10 36 210 136
0257,  0320076,  0320109,  0000033, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0281,  0320109,  0320114,  0000005, 0001, 0002 0 0 0 0 0 0 0 0 0 0
0257,  0320114,  0320405,  0000291, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0283,  0320405,  0320570,  0000165, 0001, 0000 0 0 0 0 550 -216 -40 494 768 -480
0257,  0320570,  0322707,  0002137, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0285,  0322707,  0322712,  0000005, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 1 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 2 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 3 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 4 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 5 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 6 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 7 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 8 0 0 0 0 0 0 0 0 0
```

```
 77  0265, -0000001, -0000001, -0000001, 0001, 0000 9 0 0 0 0 0 0 0 0 0
 78  0265, -0000001, -0000001, -0000001, 0001, 0000 10 0 0 0 0 0 0 0 0 0
 79  0265, -0000001, -0000001, -0000001, 0001, 0000 11 0 0 0 0 0 0 0 0 0
 80  0265, -0000001, -0000001, -0000001, 0001, 0000 12 0 0 0 0 0 0 0 0 0
 81  0265, -0000001, -0000001, -0000001, 0001, 0000 13 0 0 0 0 0 0 0 0 0
 82  0265, -0000001, -0000001, -0000001, 0001, 0000 14 0 0 0 0 0 0 0 0 0
 83  0265, -0000001, -0000001, -0000001, 0001, 0000 15 0 0 0 0 0 0 0 0 0
 84  0265, -0000001, -0000001, -0000001, 0001, 0000 16 0 0 0 0 0 0 0 0 0
 85  0258, -0000001, -0000001, -0000001, 0001, 0000 2 16 0 0 0 0 0 0 0 0
 86  0258, -0000001, -0000001, -0000001, 0001, 0000 2 15 0 0 0 0 0 0 0 0
 87  0258, -0000001, -0000001, -0000001, 0001, 0000 2 14 0 0 0 0 0 0 0 0
 88  0258, -0000001, -0000001, -0000001, 0001, 0000 2 13 0 0 0 0 0 0 0 0
 89  0258, -0000001, -0000001, -0000001, 0001, 0000 2 12 0 0 0 0 0 0 0 0
 90  0258, -0000001, -0000001, -0000001, 0001, 0000 2 11 0 0 0 0 0 0 0 0
 91  0258, -0000001, -0000001, -0000001, 0001, 0000 2 10 0 0 0 0 0 0 0 0
 92  0258, -0000001, -0000001, -0000001, 0001, 0000 2 9 0 0 0 0 0 0 0 0
 93  0258, -0000001, -0000001, -0000001, 0001, 0000 2 8 0 0 0 0 0 0 0 0
 94  0258, -0000001, -0000001, -0000001, 0001, 0000 2 7 0 0 0 0 0 0 0 0
 95  0258, -0000001, -0000001, -0000001, 0001, 0000 2 6 0 0 0 0 0 0 0 0
 96  0258, -0000001, -0000001, -0000001, 0001, 0000 2 5 0 0 0 0 0 0 0 0
 97  0258, -0000001, -0000001, -0000001, 0001, 0000 2 4 0 0 0 0 0 0 0 0
 98  0258, -0000001, -0000001, -0000001, 0001, 0000 2 3 0 0 0 0 0 0 0 0
 99  0265, -0000001, -0000001, -0000001, 0001, 0000 17 0 0 0 0 0 0 0 0 0
100  0265, -0000001, -0000001, -0000001, 0001, 0000 18 0 0 0 0 0 0 0 0 0
101  0265, -0000001, -0000001, -0000001, 0001, 0000 19 0 0 0 0 0 0 0 0 0
102  0265, -0000001, -0000001, -0000001, 0001, 0000 24 0 0 0 0 0 0 0 0 0
103  0265, -0000001, -0000001, -0000001, 0001, 0000 20 0 0 0 0 0 0 0 0 0
104  0265, -0000001, -0000001, -0000001, 0001, 0000 21 0 0 0 0 0 0 0 0 0
105  0265, -0000001, -0000001, -0000001, 0001, 0000 22 0 0 0 0 0 0 0 0 0
106  0265, -0000001, -0000001, -0000001, 0001, 0000 23 0 0 0 0 0 0 0 0 0
107  0258, -0000001, -0000001, -0000001, 0001, 0000 19 23 0 0 0 0 0 0 0 0
108  0258, -0000001, -0000001, -0000001, 0001, 0000 19 22 0 0 0 0 0 0 0 0
109  0258, -0000001, -0000001, -0000001, 0001, 0000 19 21 0 0 0 0 0 0 0 0
110  0258, -0000001, -0000001, -0000001, 0001, 0000 19 20 0 0 0 0 0 0 0 0
111  0258, -0000001, -0000001, -0000001, 0001, 0000 19 24 0 0 0 0 0 0 0 0
112  0258, -0000001, -0000001, -0000001, 0001, 0000 1 19 0 0 0 0 0 0 0 0
113  0258, -0000001, -0000001, -0000001, 0001, 0000 1 18 0 0 0 0 0 0 0 0
114  0258, -0000001, -0000001, -0000001, 0001, 0000 1 17 0 0 0 0 0 0 0 0
115  0258, -0000001, -0000001, -0000001, 0001, 0000 1 2 0 0 0 0 0 0 0 0
116  0276, -0000001, -0000001, -0000001, 0001, 0000 0 0 0 0 0 0 0 0 0 0
117  0257,  0322712,  0324733,  0002021, 0001, 0000 0 0 0 0 0 0 0 0 0 0
118  0264,  0324733,  0324744,  0000011, 0001, 0000 1 0 293 -29 0 0 0 0 0 0
119  0270,  0324744,  0325343,  0000599, 0001, 0000 1 0 0 0 0 0 0 0 0 0
120  0257,  0325343,  0325920,  0000577, 0001, 0000 0 0 0 0 0 0 0 0 0 0
121  0264,  0325920,  0325936,  0000016, 0001, 0000 2 0 99 -84 0 0 0 0 0 0
122  0270,  0325936,  0326678,  0000742, 0001, 0000 2 0 0 0 0 0 0 0 0 0
123  0257,  0326678,  0326925,  0000247, 0001, 0000 0 0 0 0 0 0 0 0 0 0
124  0264,  0326925,  0326936,  0000011, 0001, 0000 3 0 257 -80 0 0 0 0 0 0
125  0270,  0326936,  0328683,  0001747, 0001, 0000 3 0 0 0 0 0 0 0 0 0
126  0257,  0328683,  0328902,  0000219, 0001, 0000 0 0 0 0 0 0 0 0 0 0
127  0264,  0328902,  0328913,  0000011, 0001, 0000 4 0 396 -81 0 0 0 0 0 0
128  0270,  0328913,  0329902,  0000989, 0001, 0000 4 0 0 0 0 0 0 0 0 0
129  0257,  0329902,  0331802,  0001900, 0001, 0000 0 0 0 0 0 0 0 0 0 0
130  0274,  0331802,  0331879,  0000077, 0001, 0000 1 2 0 0 0 0 0 0 0 0
131  0257,  0331879,  0331967,  0000088, 0001, 0000 0 0 0 0 0 0 0 0 0 0
132  0274,  0331967,  0332022,  0000055, 0001, 0000 1 3 0 0 0 0 0 0 0 0
133  0257,  0332022,  0332093,  0000071, 0001, 0000 0 0 0 0 0 0 0 0 0 0
134  0274,  0332093,  0332165,  0000072, 0001, 0000 1 4 0 0 0 0 0 0 0 0
135  0257,  0332165,  0332429,  0000264, 0001, 0000 0 0 0 0 0 0 0 0 0 0
136  0297,  0332429,  0332434,  0000005, 0001, 0000 0 0 0 0 0 0 0 0 0 0
137  0257,  0332434,  0332840,  0000406, 0001, 0000 0 0 0 0 0 0 0 0 0 0
138  0264,  0332840,  0332857,  0000017, 0001, 0000 5 0 24 -179 0 0 0 0 0 0
139  0270,  0332857,  0333406,  0000549, 0001, 0000 5 0 0 0 0 0 0 0 0 0
140  0257,  0333406,  0333659,  0000253, 0001, 0000 0 0 0 0 0 0 0 0 0 0
141  0257,  0333659,  0334247,  0000588, 0001, 0000 0 0 0 0 0 0 0 0 0 0
142  0264,  0334247,  0334263,  0000016, 0001, 0000 6 0 152 -176 0 0 0 0 0 0
143  0270,  0334263,  0335186,  0000923, 0001, 0000 6 0 0 0 0 0 0 0 0 0
144  0257,  0335186,  0335318,  0000132, 0001, 0000 0 0 0 0 0 0 0 0 0 0
145  0274,  0335318,  0335378,  0000060, 0001, 0000 2 5 0 0 0 0 0 0 0 0
146  0257,  0335378,  0335460,  0000082, 0001, 0000 0 0 0 0 0 0 0 0 0 0
147  0274,  0335460,  0335537,  0000077, 0001, 0000 2 6 0 0 0 0 0 0 0 0
148  0257,  0335537,  0335675,  0000138, 0001, 0000 0 0 0 0 0 0 0 0 0 0
149  0264,  0335675,  0335686,  0000011, 0001, 0000 7 0 279 -181 0 0 0 0 0 0
150  0270,  0335686,  0336630,  0000944, 0001, 0256 7 0 0 0 0 0 0 0 0 0
151  0257,  0336630,  0337015,  0000385, 0001, 0000 0 0 0 0 0 0 0 0 0 0
```

```
152  0265,  0337015,  0337020,  0000005, 0001, 0000 7 0 0 0 0 0 0 0 0 0
153  0257,  0337020,  0337229,  0000209, 0001, 0000 0 0 0 0 0 0 0 0 0 0
154  0264,  0337229,  0337240,  0000011, 0001, 0000 8 0 279 -180 0 0 0 0 0 0
155  0270,  0337240,  0338300,  0001060, 0001, 0000 8 0 0 0 0 0 0 0 0 0
156  0257,  0338300,  0338454,  0000154, 0001, 0000 0 0 0 0 0 0 0 0 0 0
157  0264,  0338454,  0338465,  0000011, 0001, 0000 9 0 399 -175 0 0 0 0 0 0
158  0270,  0338465,  0339272,  0000807, 0001, 0000 9 0 0 0 0 0 0 0 0 0
159  0257,  0339272,  0339871,  0000599, 0001, 0000 0 0 0 0 0 0 0 0 0 0
160  0274,  0339871,  0339942,  0000071, 0001, 0000 3 8 0 0 0 0 0 0 0 0
161  0257,  0339942,  0340014,  0000072, 0001, 0000 0 0 0 0 0 0 0 0 0 0
162  0274,  0340014,  0340069,  0000055, 0001, 0000 3 9 0 0 0 0 0 0 0 0
163  0257,  0340069,  0340316,  0000247, 0001, 0000 0 0 0 0 0 0 0 0 0 0
164  0297,  0340316,  0340321,  0000005, 0001, 0000 0 0 0 0 0 0 0 0 0 0
165  0257,  0340321,  0340904,  0000583, 0001, 0000 0 0 0 0 0 0 0 0 0 0
166  0264,  0340904,  0340909,  0000005, 0001, 0000 10 0 557 -187 0 0 0 0 0 0
167  0270,  0340909,  0343172,  0002263, 0001, 0000 10 0 0 0 0 0 0 0 0 0
168  0257,  0343172,  0343364,  0000192, 0001, 0000 0 0 0 0 0 0 0 0 0 0
169  0274,  0343364,  0343458,  0000094, 0001, 0000 3 10 0 0 0 0 0 0 0 0
170  0257,  0343458,  0343760,  0000302, 0001, 0000 0 0 0 0 0 0 0 0 0 0
171  0297,  0343760,  0343765,  0000005, 0001, 0000 0 0 0 0 0 0 0 0 0 0
172  0257,  0343765,  0344106,  0000341, 0001, 0000 0 0 0 0 0 0 0 0 0 0
173  0281,  0344106,  0344111,  0000005, 0001, 0002 0 0 0 0 0 0 0 0 0 0
174  0257,  0344111,  0344331,  0000220, 0001, 0000 0 0 0 0 0 0 0 0 0 0
175  0283,  0344331,  0344413,  0000082, 0001, 0000 0 0 0 0 608 -258 140 -9 692 -238
176  0257,  0344413,  0345484,  0001071, 0001, 0000 0 0 0 0 0 0 0 0 0 0
177  0283,  0345484,  0345611,  0000127, 0001, 0000 0 0 140 -47 692 -276 -92 64 876 -311
178  0257,  0345611,  0346160,  0000549, 0001, 0000 0 0 0 0 0 0 0 0 0 0
179  0264,  0346160,  0346165,  0000005, 0001, 0000 11 0 717 -182 0 0 0 0 0 0
180  0270,  0346165,  0346759,  0000594, 0001, 0000 11 0 0 0 0 0 0 0 0 0
181  0257,  0346759,  0346940,  0000181, 0001, 0000 0 0 0 0 0 0 0 0 0 0
182  0274,  0346940,  0347028,  0000088, 0001, 0000 3 11 0 0 0 0 0 0 0 0
183  0257,  0347028,  0347187,  0000159, 0001, 0000 0 0 0 0 0 0 0 0 0 0
184  0264,  0347187,  0347198,  0000011, 0001, 0000 12 0 839 -177 0 0 0 0 0 0
185  0270,  0347198,  0348511,  0001313, 0001, 0000 12 0 0 0 0 0 0 0 0 0
186  0257,  0348511,  0348659,  0000148, 0001, 0000 0 0 0 0 0 0 0 0 0 0
187  0274,  0348659,  0348752,  0000093, 0001, 0000 3 12 0 0 0 0 0 0 0 0
188  0257,  0348752,  0349054,  0000302, 0001, 0000 0 0 0 0 0 0 0 0 0 0
189  0297,  0349054,  0349060,  0000006, 0001, 0000 0 0 0 0 0 0 0 0 0 0
190  0257,  0349060,  0349373,  0000313, 0001, 0000 0 0 0 0 0 0 0 0 0 0
191  0283,  0349373,  0349505,  0000132, 0001, 0000 0 0 -92 64 876 -311 5 128 1196 -311
192  0257,  0349505,  0349807,  0000302, 0001, 0000 0 0 0 0 0 0 0 0 0 0
193  0264,  0349807,  0349812,  0000005, 0001, 0000 13 0 926 -173 0 0 0 0 0 0
194  0270,  0349812,  0350505,  0000693, 0001, 0000 13 0 0 0 0 0 0 0 0 0
195  0257,  0350505,  0350653,  0000148, 0001, 0000 0 0 0 0 0 0 0 0 0 0
196  0264,  0350653,  0350664,  0000011, 0001, 0000 14 0 1063 -180 0 0 0 0 0 0
197  0270,  0350664,  0352345,  0001681, 0001, 0000 14 0 0 0 0 0 0 0 0 0
198  0257,  0352345,  0352542,  0000197, 0001, 0000 0 0 0 0 0 0 0 0 0 0
199  0274,  0352542,  0352652,  0000110, 0001, 0000 4 14 0 0 0 0 0 0 0 0
200  0257,  0352652,  0352751,  0000099, 0001, 0000 0 0 0 0 0 0 0 0 0 0
201  0274,  0352751,  0352828,  0000077, 0001, 0000 3 13 0 0 0 0 0 0 0 0
202  0257,  0352828,  0353135,  0000307, 0001, 0000 0 0 0 0 0 0 0 0 0 0
203  0297,  0353135,  0353141,  0000006, 0001, 0000 0 0 0 0 0 0 0 0 0 0
204  0257,  0353141,  0353481,  0000340, 0001, 0000 0 0 0 0 0 0 0 0 0 0
205  0264,  0353481,  0353492,  0000011, 0001, 0000 15 0 1016 -182 0 0 0 0 0 0
206  0270,  0353492,  0353849,  0000357, 0001, 0000 15 0 0 0 0 0 0 0 0 0
207  0257,  0353849,  0354020,  0000171, 0001, 0000 0 0 0 0 0 0 0 0 0 0
208  0264,  0354020,  0354031,  0000011, 0001, 0000 16 0 1122 -178 0 0 0 0 0 0
209  0270,  0354031,  0354388,  0000357, 0001, 0000 16 0 0 0 0 0 0 0 0 0
210  0257,  0354388,  0354536,  0000148, 0001, 0000 0 0 0 0 0 0 0 0 0 0
211  0274,  0354536,  0354602,  0000066, 0001, 0000 4 15 0 0 0 0 0 0 0 0
212  0257,  0354602,  0354690,  0000088, 0001, 0000 0 0 0 0 0 0 0 0 0 0
213  0274,  0354690,  0354789,  0000099, 0001, 0000 4 16 0 0 0 0 0 0 0 0
214  0257,  0354789,  0355008,  0000219, 0001, 0000 0 0 0 0 0 0 0 0 0 0
215  0297,  0355008,  0355014,  0000006, 0001, 0000 0 0 0 0 0 0 0 0 0 0
216  0257,  0355014,  0355865,  0000851, 0001, 0000 0 0 0 0 0 0 0 0 0 0
217  0264,  0355865,  0355882,  0000017, 0001, 0000 17 0 1223 -180 0 0 0 0 0 0
218  0270,  0355882,  0356244,  0000362, 0001, 0000 17 0 0 0 0 0 0 0 0 0
219  0257,  0356244,  0356442,  0000198, 0001, 0000 0 0 0 0 0 0 0 0 0 0
220  0264,  0356442,  0356453,  0000011, 0001, 0000 18 0 1364 -180 0 0 0 0 0 0
221  0270,  0356453,  0359293,  0002840, 0001, 0000 18 0 0 0 0 0 0 0 0 0
222  0257,  0359293,  0359600,  0000307, 0001, 0000 0 0 0 0 0 0 0 0 0 0
223  0283,  0359600,  0359694,  0000094, 0001, 0000 0 0 300 128 1491 -311 -39 46 1605 -302
224  0257,  0359694,  0360402,  0000708, 0001, 0000 0 0 0 0 0 0 0 0 0 0
225  0274,  0360402,  0360545,  0000143, 0001, 0000 1 18 0 0 0 0 0 0 0 0
226  0257,  0360545,  0360803,  0000258, 0001, 0000 0 0 0 0 0 0 0 0 0 0
```

```
227  0297,  0360803,  0360809,  0000006, 0001, 0000 0 0 0 0 0 0 0 0 0 0
228  0257,  0360809,  0361188,  0000379, 0001, 0000 0 0 0 0 0 0 0 0 0 0
229  0274,  0361188,  0361281,  0000093, 0001, 0000 4 17 0 0 0 0 0 0 0 0
230  0257,  0361281,  0361479,  0000198, 0001, 0000 0 0 0 0 0 0 0 0 0 0
231  0297,  0361479,  0361484,  0000005, 0001, 0000 0 0 0 0 0 0 0 0 0 0
232  0257,  0361484,  0362094,  0000610, 0001, 0000 0 0 0 0 0 0 0 0 0 0
233  0264,  0362094,  0362105,  0000011, 0001, 0000 19 0 1329 -187 0 0 0 0 0 0
234  0270,  0362105,  0363071,  0000966, 0001, 0000 19 0 0 0 0 0 0 0 0 0
235  0257,  0363071,  0363231,  0000160, 0001, 0000 0 0 0 0 0 0 0 0 0 0
236  0264,  0363231,  0363242,  0000011, 0001, 0000 20 0 1467 -183 0 0 0 0 0 0
237  0270,  0363242,  0363758,  0000516, 0001, 0000 20 0 0 0 0 0 0 0 0 0
238  0257,  0363758,  0364044,  0000286, 0001, 0000 0 0 0 0 0 0 0 0 0 0
239  0274,  0364044,  0364115,  0000071, 0001, 0000 18 19 0 0 0 0 0 0 0 0
240  0257,  0364115,  0364203,  0000088, 0001, 0000 0 0 0 0 0 0 0 0 0 0
241  0274,  0364203,  0364307,  0000104, 0001, 0000 18 20 0 0 0 0 0 0 0 0
242  0257,  0364307,  0364576,  0000269, 0001, 0000 0 0 0 0 0 0 0 0 0 0
243  0297,  0364576,  0364582,  0000006, 0001, 0000 0 0 0 0 0 0 0 0 0 0
244  0257,  0364582,  0366164,  0001582, 0001, 0000 0 0 0 0 0 0 0 0 0 0
245  0283,  0366164,  0366290,  0000126, 0001, 0000 0 0 79 46 1723 -302 597 -55 1677 -320
246  0257,  0366290,  0369306,  0003016, 0001, 0000 0 0 0 0 0 0 0 0 0 0
247  0264,  0369306,  0369316,  0000010, 0001, 0000 21 0 1422 -179 0 0 0 0 0 0
248  0270,  0369316,  0369586,  0000270, 0001, 0000 21 0 0 0 0 0 0 0 0 0
249  0257,  0369586,  0369723,  0000137, 0001, 0000 0 0 0 0 0 0 0 0 0 0
250  0274,  0369723,  0369789,  0000066, 0001, 0000 18 21 0 0 0 0 0 0 0 0
251  0257,  0369789,  0372151,  0002362, 0001, 0000 0 0 0 0 0 0 0 0 0 0
252  0264,  0372151,  0372162,  0000011, 0001, 0000 22 0 1033 -137 0 0 0 0 0 0
253  0270,  0372162,  0372628,  0000466, 0001, 0000 22 0 0 0 0 0 0 0 0 0
254  0257,  0372628,  0372766,  0000138, 0001, 0000 0 0 0 0 0 0 0 0 0 0
255  0274,  0372766,  0372837,  0000071, 0001, 0000 4 22 0 0 0 0 0 0 0 0
256  0257,  0372837,  0373117,  0000280, 0001, 0000 0 0 0 0 0 0 0 0 0 0
257  0297,  0373117,  0373123,  0000006, 0001, 0000 0 0 0 0 0 0 0 0 0 0
258  0257,  0373123,  0373469,  0000346, 0001, 0000 0 0 0 0 0 0 0 0 0 0
259  0264,  0373469,  0373480,  0000011, 0001, 0000 23 0 1077 -145 0 0 0 0 0 0
260  0270,  0373480,  0373881,  0000401, 0001, 0000 23 0 0 0 0 0 0 0 0 0
261  0257,  0373881,  0374018,  0000137, 0001, 0000 0 0 0 0 0 0 0 0 0 0
262  0274,  0374018,  0374095,  0000077, 0001, 0000 4 23 0 0 0 0 0 0 0 0
263  0257,  0374095,  0374397,  0000302, 0001, 0000 0 0 0 0 0 0 0 0 0 0
264  0297,  0374397,  0374403,  0000006, 0001, 0000 0 0 0 0 0 0 0 0 0 0
265  0257,  0374403,  0375276,  0000873, 0001, 0000 0 0 0 0 0 0 0 0 0 0
266  0283,  0375276,  0375331,  0000055, 0001, 0000 0 0 420 -55 1500 -320 773 128 1887 -247
267  0257,  0375331,  0375737,  0000406, 0001, 0000 0 0 0 0 0 0 0 0 0 0
268  0264,  0375737,  0375743,  0000006, 0001, 0000 24 0 1724 -97 0 0 0 0 0 0
269  0270,  0375743,  0376429,  0000686, 0001, 0000 24 0 0 0 0 0 0 0 0 0
270  0257,  0376429,  0376808,  0000379, 0001, 0000 0 0 0 0 0 0 0 0 0 0
271  0283,  0376808,  0376896,  0000088, 0001, 0000 0 0 773 128 1887 -247 -18 119 1967 -347
272  0257,  0376896,  0377182,  0000286, 0001, 0000 0 0 0 0 0 0 0 0 0 0
273  0274,  0377182,  0377319,  0000137, 0001, 0000 1 24 0 0 0 0 0 0 0 0
274  0257,  0377319,  0377698,  0000379, 0001, 0000 0 0 0 0 0 0 0 0 0 0
275  0297,  0377698,  0377704,  0000006, 0001, 0000 0 0 0 0 0 0 0 0 0 0
276  0257,  0377704,  0378159,  0000455, 0001, 0000 0 0 0 0 0 0 0 0 0 0
277  0264,  0378159,  0378165,  0000006, 0001, 0000 25 0 1657 -179 0 0 0 0 0 0
278  0270,  0378165,  0378670,  0000505, 0001, 0000 25 0 0 0 0 0 0 0 0 0
279  0257,  0378670,  0378797,  0000127, 0001, 0000 0 0 0 0 0 0 0 0 0 0
280  0274,  0378797,  0378868,  0000071, 0001, 0000 24 25 0 0 0 0 0 0 0 0
281  0257,  0378868,  0378934,  0000066, 0001, 0000 0 0 0 0 0 0 0 0 0 0
282  0264,  0378934,  0378945,  0000011, 0001, 0000 26 0 1817 -181 0 0 0 0 0 0
283  0270,  0378945,  0379653,  0000708, 0001, 0000 26 0 0 0 0 0 0 0 0 0
284  0257,  0379653,  0379846,  0000193, 0001, 0000 0 0 0 0 0 0 0 0 0 0
285  0264,  0379846,  0379862,  0000016, 0001, 0000 27 0 1885 -118 0 0 0 0 0 0
286  0270,  0379862,  0380631,  0000769, 0001, 0000 27 0 0 0 0 0 0 0 0 0
287  0257,  0380631,  0380763,  0000132, 0001, 0000 0 0 0 0 0 0 0 0 0 0
288  0274,  0380763,  0380856,  0000093, 0001, 0000 24 26 0 0 0 0 0 0 0 0
289  0257,  0380856,  0380955,  0000099, 0001, 0000 0 0 0 0 0 0 0 0 0 0
290  0274,  0380955,  0381054,  0000099, 0001, 0000 24 27 0 0 0 0 0 0 0 0
291  0257,  0381054,  0381395,  0000341, 0001, 0000 0 0 0 0 0 0 0 0 0 0
292  0297,  0381395,  0381400,  0000005, 0001, 0000 0 0 0 0 0 0 0 0 0 0
293  0257,  0381400,  0382098,  0000698, 0001, 0000 0 0 0 0 0 0 0 0 0 0
294  0283,  0382098,  0382197,  0000099, 0001, 0000 0 0 -18 119 1967 -347 -1423 -46 1932 -265
295  0257,  0382197,  0386794,  0004597, 0001, 0000 0 0 0 0 0 0 0 0 0 0
296  0283,  0386794,  0386898,  0000104, 0001, 0000 0 0 -1423 -46 1932 -265 -1314 -46 244 -311
297  0257,  0386898,  0387343,  0000445, 0001, 0000 0 0 0 0 0 0 0 0 0 0
298  0283,  0387343,  0387480,  0000137, 0001, 0000 0 0 -1314 -46 244 -311 -1379 275 201 -311
299  0257,  0387480,  0388573,  0001093, 0001, 0000 0 0 0 0 0 0 0 0 0 0
300  0283,  0388573,  0388656,  0000083, 0001, 0000 0 0 -1379 275 201 -311 -1011 -27 309 -256
301  0257,  0388656,  0390463,  0001807, 0001, 0000 0 0 0 0 0 0 0 0 0 0
```

```
302  0258,  0390463,  0390468,  0000005, 0001, 0000 3 10 0 0 0 0 0 0 0 0
303  0257,  0390468,  0390611,  0000143, 0001, 0000 0 0 0 0 0 0 0 0 0 0
304  0258,  0390611,  0390677,  0000066, 0001, 0000 3 11 0 0 0 0 0 0 0 0
305  0257,  0390677,  0391182,  0000505, 0001, 0000 0 0 0 0 0 0 0 0 0 0
306  0289,  0391182,  0391193,  0000011, 0001, 0008 0 0 0 0 0 0 0 0 0 0
307  0257,  0391193,  0391270,  0000077, 0001, 0000 0 0 0 0 0 0 0 0 0 0
308  0265,  0391270,  0391276,  0000006, 0001, 0000 11 0 0 0 0 0 0 0 0 0
309  0257,  0391276,  0391847,  0000571, 0001, 0000 0 0 0 0 0 0 0 0 0 0
310  0289,  0391847,  0391852,  0000005, 0001, 0000 0 0 0 0 0 0 0 0 0 0
311  0257,  0391852,  0392165,  0000313, 0001, 0000 0 0 0 0 0 0 0 0 0 0
312  0283,  0392165,  0392281,  0000116, 0001, 0000 0 0 -1011 -27 309 -256 -860 0 1131 -293
313  0257,  0392281,  0392665,  0000384, 0001, 0000 0 0 0 0 0 0 0 0 0 0
314  0274,  0392665,  0392726,  0000061, 0001, 0000 4 10 0 0 0 0 0 0 0 0
315  0257,  0392726,  0393006,  0000280, 0001, 0000 0 0 0 0 0 0 0 0 0 0
316  0297,  0393006,  0393011,  0000005, 0001, 0000 0 0 0 0 0 0 0 0 0 0
317  0257,  0393011,  0396466,  0003455, 0001, 0000 0 0 0 0 0 0 0 0 0 0
318  0287,  0396466,  0397323,  0000857, 0001, 0000 0 0 0 0 0 0 0 0 0 0
319  0257,  0397323,  0398443,  0001120, 0001, 0000 0 0 0 0 0 0 0 0 0 0
320  0283,  0398443,  0398713,  0000270, 0001, 0000 0 0 -860 0 1131 -293 -1318 211 -87 -366
321  0257,  0398713,  0399921,  0001158, 0001, 0000 0 0 0 0 0 0 0 0 0 0
322  0264,  0399921,  0399932,  0000011, 0001, 0000 28 0 -1063 -321 0 0 0 0 0 0
323  0257,  0399932,  0400432,  0000500, 0001, 0000 0 0 0 0 0 0 0 0 0 0
324  0265,  0400432,  0400437,  0000005, 0001, 0000 28 0 0 0 0 0 0 0 0 0
325  0257,  0400437,  0401684,  0001247, 0001, 0000 0 0 0 0 0 0 0 0 0 0
326  0278,  0401684,  0401926,  0000242, 0001, 0000 0 0 -1318 211 -87 -366 -1146 -10 -779 -259
327  0257,  0401926,  0402189,  0000263, 0001, 0000 0 0 0 0 0 0 0 0 0 0
328  0257,  0402189,  0402645,  0000456, 0001, 0000 0 0 0 0 0 0 0 0 0 0
329  0257,  0402645,  0403156,  0000511, 0001, 0000 0 0 0 0 0 0 0 0 0 0
330  0257,  0403156,  0403162,  0000006, 0001, 0000 0 0 0 0 0 0 0 0 0 0
331  0275,  0403162,  0409912,  0006750, 0001, 0000 0 0 0 0 0 0 0 0 0 0
332  0271,  0409912,  0409983,  0000071, 0001, 0000 5 0 0 0 0 0 0 0 0 0
333  0257,  0409983,  0410456,  0000473, 0001, 0000 0 0 0 0 0 0 0 0 0 0
334  0257,  0410456,  0410818,  0000362, 0001, 0000 0 0 0 0 0 0 0 0 0 0
335  0257,  0410818,  0411296,  0000478, 0001, 0000 0 0 0 0 0 0 0 0 0 0
336  0275,  0411296,  0423292,  0011996, 0001, 0000 0 0 0 0 0 0 0 0 0 0
337  0271,  0423292,  0423385,  0000093, 0001, 0000 6 0 0 0 0 0 0 0 0 0
338  0257,  0423385,  0423792,  0000407, 0001, 0000 0 0 0 0 0 0 0 0 0 0
339  0257,  0423792,  0424061,  0000269, 0001, 0000 0 0 0 0 0 0 0 0 0 0
340  0257,  0424061,  0424528,  0000467, 0001, 0000 0 0 0 0 0 0 0 0 0 0
341  0257,  0424528,  0424533,  0000005, 0001, 0000 0 0 0 0 0 0 0 0 0 0
342  0275,  0424533,  0430701,  0006168, 0001, 0000 0 0 0 0 0 0 0 0 0 0
343  0271,  0430701,  0430789,  0000088, 0001, 0000 5 0 0 0 0 0 0 0 0 0
344  0257,  0430789,  0432300,  0001511, 0001, 0000 0 0 0 0 0 0 0 0 0 0
345  0257,  0432300,  0432744,  0000444, 0001, 0000 0 0 0 0 0 0 0 0 0 0
346  0257,  0432744,  0433228,  0000484, 0001, 0000 0 0 0 0 0 0 0 0 0 0
347  0257,  0433228,  0433233,  0000005, 0001, 0000 0 0 0 0 0 0 0 0 0 0
348  0275,  0433233,  0436029,  0002796, 0001, 0000 0 0 0 0 0 0 0 0 0 0
349  0271,  0436029,  0436100,  0000071, 0001, 0000 8 0 0 0 0 0 0 0 0 0
350  0257,  0436100,  0436551,  0000451, 0001, 0000 0 0 0 0 0 0 0 0 0 0
351  0257,  0436551,  0436957,  0000406, 0001, 0000 0 0 0 0 0 0 0 0 0 0
352  0257,  0436957,  0437441,  0000484, 0001, 0000 0 0 0 0 0 0 0 0 0 0
353  0275,  0437441,  0442790,  0005349, 0001, 0000 0 0 0 0 0 0 0 0 0 0
354  0271,  0442790,  0442873,  0000083, 0001, 0000 9 0 0 0 0 0 0 0 0 0
355  0257,  0442873,  0443422,  0000549, 0001, 0000 0 0 0 0 0 0 0 0 0 0
356  0257,  0443422,  0443757,  0000335, 0001, 0000 0 0 0 0 0 0 0 0 0 0
357  0257,  0443757,  0444213,  0000456, 0001, 0000 0 0 0 0 0 0 0 0 0 0
358  0275,  0444213,  0452616,  0008403, 0001, 0000 0 0 0 0 0 0 0 0 0 0
359  0271,  0452616,  0452688,  0000072, 0001, 0000 12 0 0 0 0 0 0 0 0 0
360  0257,  0452688,  0453512,  0000824, 0001, 0000 0 0 0 0 0 0 0 0 0 0
361  0257,  0453512,  0454012,  0000500, 0001, 0000 0 0 0 0 0 0 0 0 0 0
362  0257,  0454012,  0454533,  0000521, 0001, 0000 0 0 0 0 0 0 0 0 0 0
363  0275,  0454533,  0457900,  0003367, 0001, 0000 0 0 0 0 0 0 0 0 0 0
364  0271,  0457900,  0457972,  0000072, 0001, 0000 13 0 0 0 0 0 0 0 0 0
365  0257,  0457972,  0458422,  0000450, 0001, 0000 0 0 0 0 0 0 0 0 0 0
366  0257,  0458422,  0458796,  0000374, 0001, 0000 0 0 0 0 0 0 0 0 0 0
367  0257,  0458796,  0459323,  0000527, 0001, 0000 0 0 0 0 0 0 0 0 0 0
368  0275,  0459323,  0464931,  0005608, 0001, 0000 0 0 0 0 0 0 0 0 0 0
369  0271,  0464931,  0465002,  0000071, 0001, 0000 10 0 0 0 0 0 0 0 0 0
370  0257,  0465002,  0465464,  0000462, 0001, 0000 0 0 0 0 0 0 0 0 0 0
371  0257,  0465464,  0465947,  0000483, 0001, 0000 0 0 0 0 0 0 0 0 0 0
372  0257,  0465947,  0466447,  0000500, 0001, 0000 0 0 0 0 0 0 0 0 0 0
373  0275,  0466447,  0469259,  0002812, 0001, 0000 0 0 0 0 0 0 0 0 0 0
374  0271,  0469259,  0469363,  0000104, 0001, 0000 14 0 0 0 0 0 0 0 0 0
375  0257,  0469363,  0470242,  0000879, 0001, 0000 0 0 0 0 0 0 0 0 0 0
376  0257,  0470242,  0470890,  0000648, 0001, 0000 0 0 0 0 0 0 0 0 0 0
```

B.3 Output from Pass 1

Pass 1 reduces the Pass 0 input stream from 449 lines to 425 lines.

```
 8  0257,  0316604,  0316885,  0000281, 0001, 0000 0 0 0 0 0 0 0 0 0 0
 9  0275,  0316885,  0316962,  0000077, 0001, 0000 0 0 0 0 0 0 0 0 0 0
10  0257,  0316962,  0317005,  0000043, 0001, 0000 0 0 0 0 0 0 0 0 0 0
11  0257,  0317005,  0317258,  0000253, 0001, 0000 0 0 0 0 0 0 0 0 0 0
12  0236,  0317258,  0318571,  0001313, 0001, 0000 0 0 0 0 0 0 0 0 0 0
13  0264, -0000001, -0000001, -0000001, 0001, 0000 1 0 0 0 0 0 0 0 0 0
14  0264, -0000001, -0000001, -0000001, 0001, 0000 2 0 0 0 0 0 0 0 0 0
15  0274, -0000001, -0000001, -0000001, 0001, 0000 1 2 0 0 0 0 0 0 0 0
16  0264, -0000001, -0000001, -0000001, 0001, 0000 3 0 0 0 0 0 0 0 0 0
17  0274, -0000001, -0000001, -0000001, 0001, 0000 2 3 0 0 0 0 0 0 0 0
18  0264, -0000001, -0000001, -0000001, 0001, 0000 4 0 0 0 0 0 0 0 0 0
19  0274, -0000001, -0000001, -0000001, 0001, 0000 2 4 0 0 0 0 0 0 0 0
20  0264, -0000001, -0000001, -0000001, 0001, 0000 5 0 0 0 0 0 0 0 0 0
21  0274, -0000001, -0000001, -0000001, 0001, 0000 2 5 0 0 0 0 0 0 0 0
22  0264, -0000001, -0000001, -0000001, 0001, 0000 6 0 0 0 0 0 0 0 0 0
23  0274, -0000001, -0000001, -0000001, 0001, 0000 2 6 0 0 0 0 0 0 0 0
24  0264, -0000001, -0000001, -0000001, 0001, 0000 7 0 0 0 0 0 0 0 0 0
25  0274, -0000001, -0000001, -0000001, 0001, 0000 2 7 0 0 0 0 0 0 0 0
26  0264, -0000001, -0000001, -0000001, 0001, 0000 8 0 0 0 0 0 0 0 0 0
27  0274, -0000001, -0000001, -0000001, 0001, 0000 2 8 0 0 0 0 0 0 0 0
28  0264, -0000001, -0000001, -0000001, 0001, 0000 9 0 0 0 0 0 0 0 0 0
29  0274, -0000001, -0000001, -0000001, 0001, 0000 2 9 0 0 0 0 0 0 0 0
30  0264, -0000001, -0000001, -0000001, 0001, 0000 10 0 0 0 0 0 0 0 0 0
31  0274, -0000001, -0000001, -0000001, 0001, 0000 2 10 0 0 0 0 0 0 0 0
32  0264, -0000001, -0000001, -0000001, 0001, 0000 11 0 0 0 0 0 0 0 0 0
33  0274, -0000001, -0000001, -0000001, 0001, 0000 2 11 0 0 0 0 0 0 0 0
34  0264, -0000001, -0000001, -0000001, 0001, 0000 12 0 0 0 0 0 0 0 0 0
35  0274, -0000001, -0000001, -0000001, 0001, 0000 2 12 0 0 0 0 0 0 0 0
36  0264, -0000001, -0000001, -0000001, 0001, 0000 13 0 0 0 0 0 0 0 0 0
37  0274, -0000001, -0000001, -0000001, 0001, 0000 2 13 0 0 0 0 0 0 0 0
38  0264, -0000001, -0000001, -0000001, 0001, 0000 14 0 0 0 0 0 0 0 0 0
39  0274, -0000001, -0000001, -0000001, 0001, 0000 2 14 0 0 0 0 0 0 0 0
40  0264, -0000001, -0000001, -0000001, 0001, 0000 15 0 0 0 0 0 0 0 0 0
41  0274, -0000001, -0000001, -0000001, 0001, 0000 2 15 0 0 0 0 0 0 0 0
42  0264, -0000001, -0000001, -0000001, 0001, 0000 16 0 0 0 0 0 0 0 0 0
43  0274, -0000001, -0000001, -0000001, 0001, 0000 2 16 0 0 0 0 0 0 0 0
44  0264, -0000001, -0000001, -0000001, 0001, 0000 17 0 0 0 0 0 0 0 0 0
45  0274, -0000001, -0000001, -0000001, 0001, 0000 1 17 0 0 0 0 0 0 0 0
46  0264, -0000001, -0000001, -0000001, 0001, 0000 18 0 0 0 0 0 0 0 0 0
47  0274, -0000001, -0000001, -0000001, 0001, 0000 1 18 0 0 0 0 0 0 0 0
48  0264, -0000001, -0000001, -0000001, 0001, 0000 19 0 0 0 0 0 0 0 0 0
49  0274, -0000001, -0000001, -0000001, 0001, 0000 1 19 0 0 0 0 0 0 0 0
50  0264, -0000001, -0000001, -0000001, 0001, 0000 20 0 0 0 0 0 0 0 0 0
51  0274, -0000001, -0000001, -0000001, 0001, 0000 19 24 0 0 0 0 0 0 0 0
52  0264, -0000001, -0000001, -0000001, 0001, 0000 25 0 0 0 0 0 0 0 0 0
53  0274, -0000001, -0000001, -0000001, 0001, 0000 19 20 0 0 0 0 0 0 0 0
54  0264, -0000001, -0000001, -0000001, 0001, 0000 26 0 0 0 0 0 0 0 0 0
55  0274, -0000001, -0000001, -0000001, 0001, 0000 19 21 0 0 0 0 0 0 0 0
56  0264, -0000001, -0000001, -0000001, 0001, 0000 27 0 0 0 0 0 0 0 0 0
57  0274, -0000001, -0000001, -0000001, 0001, 0000 19 22 0 0 0 0 0 0 0 0
58  0264, -0000001, -0000001, -0000001, 0001, 0000 28 0 0 0 0 0 0 0 0 0
59  0274, -0000001, -0000001, -0000001, 0001, 0000 19 23 0 0 0 0 0 0 0 0
60  0257,  0318571,  0319241,  0000670, 0001, 0000 0 0 0 0 0 0 0 0 0 0
61  0257,  0319241,  0320070,  0000829, 0001, 0000 0 0 0 0 0 0 0 0 0 0
62  0279,  0320070,  0320076,  0000006, 0001, 0000 0 0 0 0 550 -216 10 36 210 136
63  0257,  0320076,  0320109,  0000033, 0001, 0000 0 0 0 0 0 0 0 0 0 0
64  0281,  0320109,  0320114,  0000005, 0001, 0002 0 0 0 0 0 0 0 0 0 0
65  0257,  0320114,  0320405,  0000291, 0001, 0000 0 0 0 0 0 0 0 0 0 0
66  0283,  0320405,  0320570,  0000165, 0001, 0000 0 0 0 0 550 -216 -40 494 768 -480
67  0257,  0320570,  0322707,  0002137, 0001, 0000 0 0 0 0 0 0 0 0 0 0
68  0285,  0322707,  0322712,  0000005, 0001, 0000 0 0 0 0 0 0 0 0 0 0
69  0265, -0000001, -0000001, -0000001, 0001, 0000 1 0 0 0 0 0 0 0 0 0
70  0265, -0000001, -0000001, -0000001, 0001, 0000 2 0 0 0 0 0 0 0 0 0
71  0265, -0000001, -0000001, -0000001, 0001, 0000 3 0 0 0 0 0 0 0 0 0
72  0265, -0000001, -0000001, -0000001, 0001, 0000 4 0 0 0 0 0 0 0 0 0
73  0265, -0000001, -0000001, -0000001, 0001, 0000 5 0 0 0 0 0 0 0 0 0
74  0265, -0000001, -0000001, -0000001, 0001, 0000 6 0 0 0 0 0 0 0 0 0
75  0265, -0000001, -0000001, -0000001, 0001, 0000 7 0 0 0 0 0 0 0 0 0
76  0265, -0000001, -0000001, -0000001, 0001, 0000 8 0 0 0 0 0 0 0 0 0
```

```
 77  0265, -0000001, -0000001, -0000001, 0001, 0000 9 0 0 0 0 0 0 0 0 0
 78  0265, -0000001, -0000001, -0000001, 0001, 0000 10 0 0 0 0 0 0 0 0 0
 79  0265, -0000001, -0000001, -0000001, 0001, 0000 11 0 0 0 0 0 0 0 0 0
 80  0265, -0000001, -0000001, -0000001, 0001, 0000 12 0 0 0 0 0 0 0 0 0
 81  0265, -0000001, -0000001, -0000001, 0001, 0000 13 0 0 0 0 0 0 0 0 0
 82  0265, -0000001, -0000001, -0000001, 0001, 0000 14 0 0 0 0 0 0 0 0 0
 83  0265, -0000001, -0000001, -0000001, 0001, 0000 15 0 0 0 0 0 0 0 0 0
 84  0265, -0000001, -0000001, -0000001, 0001, 0000 16 0 0 0 0 0 0 0 0 0
 85  0258, -0000001, -0000001, -0000001, 0001, 0000 2 16 0 0 0 0 0 0 0 0
 86  0258, -0000001, -0000001, -0000001, 0001, 0000 2 15 0 0 0 0 0 0 0 0
 87  0258, -0000001, -0000001, -0000001, 0001, 0000 2 14 0 0 0 0 0 0 0 0
 88  0258, -0000001, -0000001, -0000001, 0001, 0000 2 13 0 0 0 0 0 0 0 0
 89  0258, -0000001, -0000001, -0000001, 0001, 0000 2 12 0 0 0 0 0 0 0 0
 90  0258, -0000001, -0000001, -0000001, 0001, 0000 2 11 0 0 0 0 0 0 0 0
 91  0258, -0000001, -0000001, -0000001, 0001, 0000 2 10 0 0 0 0 0 0 0 0
 92  0258, -0000001, -0000001, -0000001, 0001, 0000 2 9 0 0 0 0 0 0 0 0
 93  0258, -0000001, -0000001, -0000001, 0001, 0000 2 8 0 0 0 0 0 0 0 0
 94  0258, -0000001, -0000001, -0000001, 0001, 0000 2 7 0 0 0 0 0 0 0 0
 95  0258, -0000001, -0000001, -0000001, 0001, 0000 2 6 0 0 0 0 0 0 0 0
 96  0258, -0000001, -0000001, -0000001, 0001, 0000 2 5 0 0 0 0 0 0 0 0
 97  0258, -0000001, -0000001, -0000001, 0001, 0000 2 4 0 0 0 0 0 0 0 0
 98  0258, -0000001, -0000001, -0000001, 0001, 0000 2 3 0 0 0 0 0 0 0 0
 99  0265, -0000001, -0000001, -0000001, 0001, 0000 17 0 0 0 0 0 0 0 0 0
100  0265, -0000001, -0000001, -0000001, 0001, 0000 18 0 0 0 0 0 0 0 0 0
101  0265, -0000001, -0000001, -0000001, 0001, 0000 19 0 0 0 0 0 0 0 0 0
102  0265, -0000001, -0000001, -0000001, 0001, 0000 24 0 0 0 0 0 0 0 0 0
103  0265, -0000001, -0000001, -0000001, 0001, 0000 20 0 0 0 0 0 0 0 0 0
104  0265, -0000001, -0000001, -0000001, 0001, 0000 21 0 0 0 0 0 0 0 0 0
105  0265, -0000001, -0000001, -0000001, 0001, 0000 22 0 0 0 0 0 0 0 0 0
106  0265, -0000001, -0000001, -0000001, 0001, 0000 23 0 0 0 0 0 0 0 0 0
107  0258, -0000001, -0000001, -0000001, 0001, 0000 19 23 0 0 0 0 0 0 0 0
108  0258, -0000001, -0000001, -0000001, 0001, 0000 19 22 0 0 0 0 0 0 0 0
109  0258, -0000001, -0000001, -0000001, 0001, 0000 19 21 0 0 0 0 0 0 0 0
110  0258, -0000001, -0000001, -0000001, 0001, 0000 19 20 0 0 0 0 0 0 0 0
111  0258, -0000001, -0000001, -0000001, 0001, 0000 19 24 0 0 0 0 0 0 0 0
112  0258, -0000001, -0000001, -0000001, 0001, 0000 1 19 0 0 0 0 0 0 0 0
113  0258, -0000001, -0000001, -0000001, 0001, 0000 1 18 0 0 0 0 0 0 0 0
114  0258, -0000001, -0000001, -0000001, 0001, 0000 1 17 0 0 0 0 0 0 0 0
115  0258, -0000001, -0000001, -0000001, 0001, 0000 1 2 0 0 0 0 0 0 0 0
116  0276, -0000001, -0000001, -0000001, 0001, 0000 0 0 0 0 0 0 0 0 0 0
117  0257,  0322712,  0324733,  0002021, 0001, 0000 0 0 0 0 0 0 0 0 0 0
118  0264,  0324733,  0324744,  0000011, 0001, 0000 1 0 293 -29 0 0 0 0 0 0
119  0270,  0324744,  0325343,  0000599, 0001, 0000 1 0 0 0 0 0 0 0 0 0
120  0257,  0325343,  0325920,  0000577, 0001, 0000 0 0 0 0 0 0 0 0 0 0
121  0264,  0325920,  0325936,  0000016, 0001, 0000 2 0 99 -84 0 0 0 0 0 0
122  0270,  0325936,  0326678,  0000742, 0001, 0000 2 0 0 0 0 0 0 0 0 0
123  0257,  0326678,  0326925,  0000247, 0001, 0000 0 0 0 0 0 0 0 0 0 0
124  0264,  0326925,  0326936,  0000011, 0001, 0000 3 0 257 -80 0 0 0 0 0 0
125  0270,  0326936,  0328683,  0001747, 0001, 0000 3 0 0 0 0 0 0 0 0 0
126  0257,  0328683,  0328902,  0000219, 0001, 0000 0 0 0 0 0 0 0 0 0 0
127  0264,  0328902,  0328913,  0000011, 0001, 0000 4 0 396 -81 0 0 0 0 0 0
128  0270,  0328913,  0329902,  0000989, 0001, 0000 4 0 0 0 0 0 0 0 0 0
129  0257,  0329902,  0331802,  0001900, 0001, 0000 0 0 0 0 0 0 0 0 0 0
130  0274,  0331802,  0331879,  0000077, 0001, 0000 1 2 0 0 0 0 0 0 0 0
131  0257,  0331879,  0331967,  0000088, 0001, 0000 0 0 0 0 0 0 0 0 0 0
132  0274,  0331967,  0332022,  0000055, 0001, 0000 1 3 0 0 0 0 0 0 0 0
133  0257,  0332022,  0332093,  0000071, 0001, 0000 0 0 0 0 0 0 0 0 0 0
134  0274,  0332093,  0332165,  0000072, 0001, 0000 1 4 0 0 0 0 0 0 0 0
135  0257,  0332165,  0332429,  0000264, 0001, 0000 0 0 0 0 0 0 0 0 0 0
136  0297,  0332429,  0332434,  0000005, 0001, 0000 0 0 0 0 0 0 0 0 0 0
137  0257,  0332434,  0332840,  0000406, 0001, 0000 0 0 0 0 0 0 0 0 0 0
138  0264,  0332840,  0332857,  0000017, 0001, 0000 5 0 24 -179 0 0 0 0 0 0
139  0270,  0332857,  0333406,  0000549, 0001, 0000 5 0 0 0 0 0 0 0 0 0
140  0257,  0333406,  0333659,  0000253, 0001, 0000 0 0 0 0 0 0 0 0 0 0
141  0257,  0333659,  0334247,  0000588, 0001, 0000 0 0 0 0 0 0 0 0 0 0
142  0264,  0334247,  0334263,  0000016, 0001, 0000 6 0 152 -176 0 0 0 0 0 0
143  0270,  0334263,  0335186,  0000923, 0001, 0000 6 0 0 0 0 0 0 0 0 0
144  0257,  0335186,  0335318,  0000132, 0001, 0000 0 0 0 0 0 0 0 0 0 0
145  0274,  0335318,  0335378,  0000060, 0001, 0000 2 5 0 0 0 0 0 0 0 0
146  0257,  0335378,  0335460,  0000082, 0001, 0000 0 0 0 0 0 0 0 0 0 0
147  0274,  0335460,  0335537,  0000077, 0001, 0000 2 6 0 0 0 0 0 0 0 0
148  0257,  0335537,  0335675,  0000138, 0001, 0000 0 0 0 0 0 0 0 0 0 0
149  0264,  0335675,  0335686,  0000011, 0001, 0000 7 0 279 -181 0 0 0 0 0 0
150  0270,  0335686,  0336630,  0000944, 0001, 0256 7 0 0 0 0 0 0 0 0 0
151  0257,  0336630,  0337015,  0000385, 0001, 0000 0 0 0 0 0 0 0 0 0 0
```

```
152  0265,  0337015,  0337020,  0000005, 0001, 0000 7 0 0 0 0 0 0 0 0 0
153  0257,  0337020,  0337229,  0000209, 0001, 0000 0 0 0 0 0 0 0 0 0 0
154  0264,  0337229,  0337240,  0000011, 0001, 0000 8 0 279 -180 0 0 0 0 0 0
155  0270,  0337240,  0338300,  0001060, 0001, 0000 8 0 0 0 0 0 0 0 0 0
156  0257,  0338300,  0338454,  0000154, 0001, 0000 0 0 0 0 0 0 0 0 0 0
157  0264,  0338454,  0338465,  0000011, 0001, 0000 9 0 399 -175 0 0 0 0 0 0
158  0270,  0338465,  0339272,  0000807, 0001, 0000 9 0 0 0 0 0 0 0 0 0
159  0257,  0339272,  0339871,  0000599, 0001, 0000 0 0 0 0 0 0 0 0 0 0
160  0274,  0339871,  0339942,  0000071, 0001, 0000 3 8 0 0 0 0 0 0 0 0
161  0257,  0339942,  0340014,  0000072, 0001, 0000 0 0 0 0 0 0 0 0 0 0
162  0274,  0340014,  0340069,  0000055, 0001, 0000 3 9 0 0 0 0 0 0 0 0
163  0257,  0340069,  0340316,  0000247, 0001, 0000 0 0 0 0 0 0 0 0 0 0
164  0297,  0340316,  0340321,  0000005, 0001, 0000 0 0 0 0 0 0 0 0 0 0
165  0257,  0340321,  0340904,  0000583, 0001, 0000 0 0 0 0 0 0 0 0 0 0
166  0264,  0340904,  0340909,  0000005, 0001, 0000 10 0 557 -187 0 0 0 0 0 0
167  0270,  0340909,  0343172,  0002263, 0001, 0000 10 0 0 0 0 0 0 0 0 0
168  0257,  0343172,  0343364,  0000192, 0001, 0000 0 0 0 0 0 0 0 0 0 0
169  0274,  0343364,  0343458,  0000094, 0001, 0000 3 10 0 0 0 0 0 0 0 0
170  0257,  0343458,  0343760,  0000302, 0001, 0000 0 0 0 0 0 0 0 0 0 0
171  0297,  0343760,  0343765,  0000005, 0001, 0000 0 0 0 0 0 0 0 0 0 0
172  0257,  0343765,  0344106,  0000341, 0001, 0000 0 0 0 0 0 0 0 0 0 0
173  0281,  0344106,  0344111,  0000005, 0001, 0002 0 0 0 0 0 0 0 0 0 0
174  0257,  0344111,  0344331,  0000220, 0001, 0000 0 0 0 0 0 0 0 0 0 0
175  0283,  0344331,  0344413,  0000082, 0001, 0000 0 0 0 0 608 -258 140 -9 692 -238
176  0257,  0344413,  0345484,  0001071, 0001, 0000 0 0 0 0 0 0 0 0 0 0
177  0283,  0345484,  0345611,  0000127, 0001, 0000 0 0 140 -47 692 -276 -92 64 876 -311
178  0257,  0345611,  0346160,  0000549, 0001, 0000 0 0 0 0 0 0 0 0 0 0
179  0264,  0346160,  0346165,  0000005, 0001, 0000 11 0 717 -182 0 0 0 0 0 0
180  0270,  0346165,  0346759,  0000594, 0001, 0000 11 0 0 0 0 0 0 0 0 0
181  0257,  0346759,  0346940,  0000181, 0001, 0000 0 0 0 0 0 0 0 0 0 0
182  0274,  0346940,  0347028,  0000088, 0001, 0000 3 11 0 0 0 0 0 0 0 0
183  0257,  0347028,  0347187,  0000159, 0001, 0000 0 0 0 0 0 0 0 0 0 0
184  0264,  0347187,  0347198,  0000011, 0001, 0000 12 0 839 -177 0 0 0 0 0 0
185  0270,  0347198,  0348511,  0001313, 0001, 0000 12 0 0 0 0 0 0 0 0 0
186  0257,  0348511,  0348659,  0000148, 0001, 0000 0 0 0 0 0 0 0 0 0 0
187  0274,  0348659,  0348752,  0000093, 0001, 0000 3 12 0 0 0 0 0 0 0 0
188  0257,  0348752,  0349054,  0000302, 0001, 0000 0 0 0 0 0 0 0 0 0 0
189  0297,  0349054,  0349060,  0000006, 0001, 0000 0 0 0 0 0 0 0 0 0 0
190  0257,  0349060,  0349373,  0000313, 0001, 0000 0 0 0 0 0 0 0 0 0 0
191  0283,  0349373,  0349505,  0000132, 0001, 0000 0 0 -92 64 876 -311 5 128 1196 -311
192  0257,  0349505,  0349807,  0000302, 0001, 0000 0 0 0 0 0 0 0 0 0 0
193  0264,  0349807,  0349812,  0000005, 0001, 0000 13 0 926 -173 0 0 0 0 0 0
194  0270,  0349812,  0350505,  0000693, 0001, 0000 13 0 0 0 0 0 0 0 0 0
195  0257,  0350505,  0350653,  0000148, 0001, 0000 0 0 0 0 0 0 0 0 0 0
196  0264,  0350653,  0350664,  0000011, 0001, 0000 14 0 1063 -180 0 0 0 0 0 0
197  0270,  0350664,  0352345,  0001681, 0001, 0000 14 0 0 0 0 0 0 0 0 0
198  0257,  0352345,  0352542,  0000197, 0001, 0000 0 0 0 0 0 0 0 0 0 0
199  0274,  0352542,  0352652,  0000110, 0001, 0000 4 14 0 0 0 0 0 0 0 0
200  0257,  0352652,  0352751,  0000099, 0001, 0000 0 0 0 0 0 0 0 0 0 0
201  0274,  0352751,  0352828,  0000077, 0001, 0000 3 13 0 0 0 0 0 0 0 0
202  0257,  0352828,  0353135,  0000307, 0001, 0000 0 0 0 0 0 0 0 0 0 0
203  0297,  0353135,  0353141,  0000006, 0001, 0000 0 0 0 0 0 0 0 0 0 0
204  0257,  0353141,  0353481,  0000340, 0001, 0000 0 0 0 0 0 0 0 0 0 0
205  0264,  0353481,  0353492,  0000011, 0001, 0000 15 0 1016 -182 0 0 0 0 0 0
206  0270,  0353492,  0353849,  0000357, 0001, 0000 15 0 0 0 0 0 0 0 0 0
207  0257,  0353849,  0354020,  0000171, 0001, 0000 0 0 0 0 0 0 0 0 0 0
208  0264,  0354020,  0354031,  0000011, 0001, 0000 16 0 1122 -178 0 0 0 0 0 0
209  0270,  0354031,  0354388,  0000357, 0001, 0000 16 0 0 0 0 0 0 0 0 0
210  0257,  0354388,  0354536,  0000148, 0001, 0000 0 0 0 0 0 0 0 0 0 0
211  0274,  0354536,  0354602,  0000066, 0001, 0000 4 15 0 0 0 0 0 0 0 0
212  0257,  0354602,  0354690,  0000088, 0001, 0000 0 0 0 0 0 0 0 0 0 0
213  0274,  0354690,  0354789,  0000099, 0001, 0000 4 16 0 0 0 0 0 0 0 0
214  0257,  0354789,  0355008,  0000219, 0001, 0000 0 0 0 0 0 0 0 0 0 0
215  0297,  0355008,  0355014,  0000006, 0001, 0000 0 0 0 0 0 0 0 0 0 0
216  0257,  0355014,  0355865,  0000851, 0001, 0000 0 0 0 0 0 0 0 0 0 0
217  0264,  0355865,  0355882,  0000017, 0001, 0000 17 0 1223 -180 0 0 0 0 0 0
218  0270,  0355882,  0356244,  0000362, 0001, 0000 17 0 0 0 0 0 0 0 0 0
219  0257,  0356244,  0356442,  0000198, 0001, 0000 0 0 0 0 0 0 0 0 0 0
220  0264,  0356442,  0356453,  0000011, 0001, 0000 18 0 1364 -180 0 0 0 0 0 0
221  0270,  0356453,  0359293,  0002840, 0001, 0000 18 0 0 0 0 0 0 0 0 0
222  0257,  0359293,  0359600,  0000307, 0001, 0000 0 0 0 0 0 0 0 0 0 0
223  0283,  0359600,  0359694,  0000094, 0001, 0000 0 0 300 128 1491 -311 -39 46 1605 -302
224  0257,  0359694,  0360402,  0000708, 0001, 0000 0 0 0 0 0 0 0 0 0 0
225  0274,  0360402,  0360545,  0000143, 0001, 0000 1 18 0 0 0 0 0 0 0 0
226  0257,  0360545,  0360803,  0000258, 0001, 0000 0 0 0 0 0 0 0 0 0 0
```

```
227  0297,  0360803,  0360809,  0000006, 0001, 0000 0 0 0 0 0 0 0 0 0 0
228  0257,  0360809,  0361188,  0000379, 0001, 0000 0 0 0 0 0 0 0 0 0 0
229  0274,  0361188,  0361281,  0000093, 0001, 0000 4 17 0 0 0 0 0 0 0 0
230  0257,  0361281,  0361479,  0000198, 0001, 0000 0 0 0 0 0 0 0 0 0 0
231  0297,  0361479,  0361484,  0000005, 0001, 0000 0 0 0 0 0 0 0 0 0 0
232  0257,  0361484,  0362094,  0000610, 0001, 0000 0 0 0 0 0 0 0 0 0 0
233  0264,  0362094,  0362105,  0000011, 0001, 0000 19 0 1329 -187 0 0 0 0 0 0
234  0270,  0362105,  0363071,  0000966, 0001, 0000 19 0 0 0 0 0 0 0 0 0
235  0257,  0363071,  0363231,  0000160, 0001, 0000 0 0 0 0 0 0 0 0 0 0
236  0264,  0363231,  0363242,  0000011, 0001, 0000 20 0 1467 -183 0 0 0 0 0 0
237  0270,  0363242,  0363758,  0000516, 0001, 0000 20 0 0 0 0 0 0 0 0 0
238  0257,  0363758,  0364044,  0000286, 0001, 0000 0 0 0 0 0 0 0 0 0 0
239  0274,  0364044,  0364115,  0000071, 0001, 0000 18 19 0 0 0 0 0 0 0 0
240  0257,  0364115,  0364203,  0000088, 0001, 0000 0 0 0 0 0 0 0 0 0 0
241  0274,  0364203,  0364307,  0000104, 0001, 0000 18 20 0 0 0 0 0 0 0 0
242  0257,  0364307,  0364576,  0000269, 0001, 0000 0 0 0 0 0 0 0 0 0 0
243  0297,  0364576,  0364582,  0000006, 0001, 0000 0 0 0 0 0 0 0 0 0 0
244  0257,  0364582,  0366164,  0001582, 0001, 0000 0 0 0 0 0 0 0 0 0 0
245  0283,  0366164,  0366290,  0000126, 0001, 0000 0 0 79 46 1723 -302 597 -55 1677 -320
246  0257,  0366290,  0369306,  0003016, 0001, 0000 0 0 0 0 0 0 0 0 0 0
247  0264,  0369306,  0369316,  0000010, 0001, 0000 21 0 1422 -179 0 0 0 0 0 0
248  0270,  0369316,  0369586,  0000270, 0001, 0000 21 0 0 0 0 0 0 0 0 0
249  0257,  0369586,  0369723,  0000137, 0001, 0000 0 0 0 0 0 0 0 0 0 0
250  0274,  0369723,  0369789,  0000066, 0001, 0000 18 21 0 0 0 0 0 0 0 0
251  0257,  0369789,  0372151,  0002362, 0001, 0000 0 0 0 0 0 0 0 0 0 0
252  0264,  0372151,  0372162,  0000011, 0001, 0000 22 0 1033 -137 0 0 0 0 0 0
253  0270,  0372162,  0372628,  0000466, 0001, 0000 22 0 0 0 0 0 0 0 0 0
254  0257,  0372628,  0372766,  0000138, 0001, 0000 0 0 0 0 0 0 0 0 0 0
255  0274,  0372766,  0372837,  0000071, 0001, 0000 4 22 0 0 0 0 0 0 0 0
256  0257,  0372837,  0373117,  0000280, 0001, 0000 0 0 0 0 0 0 0 0 0 0
257  0297,  0373117,  0373123,  0000006, 0001, 0000 0 0 0 0 0 0 0 0 0 0
258  0257,  0373123,  0373469,  0000346, 0001, 0000 0 0 0 0 0 0 0 0 0 0
259  0264,  0373469,  0373480,  0000011, 0001, 0000 23 0 1077 -145 0 0 0 0 0 0
260  0270,  0373480,  0373881,  0000401, 0001, 0000 23 0 0 0 0 0 0 0 0 0
261  0257,  0373881,  0374018,  0000137, 0001, 0000 0 0 0 0 0 0 0 0 0 0
262  0274,  0374018,  0374095,  0000077, 0001, 0000 4 23 0 0 0 0 0 0 0 0
263  0257,  0374095,  0374397,  0000302, 0001, 0000 0 0 0 0 0 0 0 0 0 0
264  0297,  0374397,  0374403,  0000006, 0001, 0000 0 0 0 0 0 0 0 0 0 0
265  0257,  0374403,  0375276,  0000873, 0001, 0000 0 0 0 0 0 0 0 0 0 0
266  0283,  0375276,  0375331,  0000055, 0001, 0000 0 0 420 -55 1500 -320 773 128 1887 -247
267  0257,  0375331,  0375737,  0000406, 0001, 0000 0 0 0 0 0 0 0 0 0 0
268  0264,  0375737,  0375743,  0000006, 0001, 0000 24 0 1724 -97 0 0 0 0 0 0
269  0270,  0375743,  0376429,  0000686, 0001, 0000 24 0 0 0 0 0 0 0 0 0
270  0257,  0376429,  0376808,  0000379, 0001, 0000 0 0 0 0 0 0 0 0 0 0
271  0283,  0376808,  0376896,  0000088, 0001, 0000 0 0 773 128 1887 -247 -18 119 1967 -347
272  0257,  0376896,  0377182,  0000286, 0001, 0000 0 0 0 0 0 0 0 0 0 0
273  0274,  0377182,  0377319,  0000137, 0001, 0000 1 24 0 0 0 0 0 0 0 0
274  0257,  0377319,  0377698,  0000379, 0001, 0000 0 0 0 0 0 0 0 0 0 0
275  0297,  0377698,  0377704,  0000006, 0001, 0000 0 0 0 0 0 0 0 0 0 0
276  0257,  0377704,  0378159,  0000455, 0001, 0000 0 0 0 0 0 0 0 0 0 0
277  0264,  0378159,  0378165,  0000006, 0001, 0000 25 0 1657 -179 0 0 0 0 0 0
278  0270,  0378165,  0378670,  0000505, 0001, 0000 25 0 0 0 0 0 0 0 0 0
279  0257,  0378670,  0378797,  0000127, 0001, 0000 0 0 0 0 0 0 0 0 0 0
280  0274,  0378797,  0378868,  0000071, 0001, 0000 24 25 0 0 0 0 0 0 0 0
281  0257,  0378868,  0378934,  0000066, 0001, 0000 0 0 0 0 0 0 0 0 0 0
282  0264,  0378934,  0378945,  0000011, 0001, 0000 26 0 1817 -181 0 0 0 0 0 0
283  0270,  0378945,  0379653,  0000708, 0001, 0000 26 0 0 0 0 0 0 0 0 0
284  0257,  0379653,  0379846,  0000193, 0001, 0000 0 0 0 0 0 0 0 0 0 0
285  0264,  0379846,  0379862,  0000016, 0001, 0000 27 0 1885 -118 0 0 0 0 0 0
286  0270,  0379862,  0380631,  0000769, 0001, 0000 27 0 0 0 0 0 0 0 0 0
287  0257,  0380631,  0380763,  0000132, 0001, 0000 0 0 0 0 0 0 0 0 0 0
288  0274,  0380763,  0380856,  0000093, 0001, 0000 24 26 0 0 0 0 0 0 0 0
289  0257,  0380856,  0380955,  0000099, 0001, 0000 0 0 0 0 0 0 0 0 0 0
290  0274,  0380955,  0381054,  0000099, 0001, 0000 24 27 0 0 0 0 0 0 0 0
291  0257,  0381054,  0381395,  0000341, 0001, 0000 0 0 0 0 0 0 0 0 0 0
292  0297,  0381395,  0381400,  0000005, 0001, 0000 0 0 0 0 0 0 0 0 0 0
293  0257,  0381400,  0382098,  0000698, 0001, 0000 0 0 0 0 0 0 0 0 0 0
294  0283,  0382098,  0382197,  0000099, 0001, 0000 0 0 -18 119 1967 -347 -1423 -46 1932 -265
295  0257,  0382197,  0386794,  0004597, 0001, 0000 0 0 0 0 0 0 0 0 0 0
296  0283,  0386794,  0386898,  0000104, 0001, 0000 0 0 -1423 -46 1932 -265 -1314 -46 244 -311
297  0257,  0386898,  0387343,  0000445, 0001, 0000 0 0 0 0 0 0 0 0 0 0
298  0283,  0387343,  0387480,  0000137, 0001, 0000 0 0 -1314 -46 244 -311 -1379 275 201 -311
299  0257,  0387480,  0388573,  0001093, 0001, 0000 0 0 0 0 0 0 0 0 0 0
300  0283,  0388573,  0388656,  0000083, 0001, 0000 0 0 -1379 275 201 -311 -1011 -27 309 -256
301  0257,  0388656,  0390463,  0001807, 0001, 0000 0 0 0 0 0 0 0 0 0 0
```

```
302  0258,  0390463,  0390468,  0000005, 0001, 0000 3 10 0 0 0 0 0 0 0 0
303  0257,  0390468,  0390611,  0000143, 0001, 0000 0 0 0 0 0 0 0 0 0 0
304  0258,  0390611,  0390677,  0000066, 0001, 0000 3 11 0 0 0 0 0 0 0 0
305  0257,  0390677,  0391182,  0000505, 0001, 0000 0 0 0 0 0 0 0 0 0 0
306  0265,  0391182,  0391852,  0000670, 0005, 0003 11 0 0 0 0 0 0 0 0 0
311  0257,  0391852,  0392165,  0000313, 0001, 0000 0 0 0 0 0 0 0 0 0 0
312  0283,  0392165,  0392281,  0000116, 0001, 0000 0 0 -1011 -27 309 -256 -860 0 1131 -293
313  0257,  0392281,  0392665,  0000384, 0001, 0000 0 0 0 0 0 0 0 0 0 0
314  0274,  0392665,  0392726,  0000061, 0001, 0000 4 10 0 0 0 0 0 0 0 0
315  0257,  0392726,  0393006,  0000280, 0001, 0000 0 0 0 0 0 0 0 0 0 0
316  0297,  0393006,  0393011,  0000005, 0001, 0000 0 0 0 0 0 0 0 0 0 0
317  0257,  0393011,  0396466,  0003455, 0001, 0000 0 0 0 0 0 0 0 0 0 0
318  0287,  0396466,  0397323,  0000857, 0001, 0000 0 0 0 0 0 0 0 0 0 0
319  0257,  0397323,  0398443,  0001120, 0001, 0000 0 0 0 0 0 0 0 0 0 0
320  0283,  0398443,  0398713,  0000270, 0001, 0000 0 0 -860 0 1131 -293 -1318 211 -87 -366
321  0257,  0398713,  0399921,  0001158, 0001, 0000 0 0 0 0 0 0 0 0 0 0
322  0264,  0399921,  0399932,  0000011, 0001, 0000 28 0 -1063 -321 0 0 0 0 0 0
323  0257,  0399932,  0400432,  0000500, 0001, 0000 0 0 0 0 0 0 0 0 0 0
324  0265,  0400432,  0400437,  0000005, 0001, 0000 28 0 0 0 0 0 0 0 0 0
325  0257,  0400437,  0401684,  0001247, 0001, 0000 0 0 0 0 0 0 0 0 0 0
326  0278,  0401684,  0401926,  0000242, 0001, 0000 0 0 -1318 211 -87 -366 -1146 -10 -779 -259
327  0257,  0401926,  0402189,  0000263, 0001, 0000 0 0 0 0 0 0 0 0 0 0
328  0257,  0402189,  0402645,  0000456, 0001, 0000 0 0 0 0 0 0 0 0 0 0
329  0257,  0402645,  0403156,  0000511, 0001, 0000 0 0 0 0 0 0 0 0 0 0
330  0257,  0403156,  0403162,  0000006, 0001, 0000 0 0 0 0 0 0 0 0 0 0
331  0271,  0403162,  0409983,  0006821, 0002, 0000 5 0 0 0 0 0 0 0 0 0
333  0257,  0409983,  0410456,  0000473, 0001, 0000 0 0 0 0 0 0 0 0 0 0
334  0257,  0410456,  0410818,  0000362, 0001, 0000 0 0 0 0 0 0 0 0 0 0
335  0257,  0410818,  0411296,  0000478, 0001, 0000 0 0 0 0 0 0 0 0 0 0
336  0271,  0411296,  0423385,  0012089, 0002, 0000 6 0 0 0 0 0 0 0 0 0
338  0257,  0423385,  0423792,  0000407, 0001, 0000 0 0 0 0 0 0 0 0 0 0
339  0257,  0423792,  0424061,  0000269, 0001, 0000 0 0 0 0 0 0 0 0 0 0
340  0257,  0424061,  0424528,  0000467, 0001, 0000 0 0 0 0 0 0 0 0 0 0
341  0257,  0424528,  0424533,  0000005, 0001, 0000 0 0 0 0 0 0 0 0 0 0
342  0271,  0424533,  0430789,  0006256, 0002, 0000 5 0 0 0 0 0 0 0 0 0
344  0257,  0430789,  0432300,  0001511, 0001, 0000 0 0 0 0 0 0 0 0 0 0
345  0257,  0432300,  0432744,  0000444, 0001, 0000 0 0 0 0 0 0 0 0 0 0
346  0257,  0432744,  0433228,  0000484, 0001, 0000 0 0 0 0 0 0 0 0 0 0
347  0257,  0433228,  0433233,  0000005, 0001, 0000 0 0 0 0 0 0 0 0 0 0
348  0271,  0433233,  0436100,  0002867, 0002, 0000 8 0 0 0 0 0 0 0 0 0
350  0257,  0436100,  0436551,  0000451, 0001, 0000 0 0 0 0 0 0 0 0 0 0
351  0257,  0436551,  0436957,  0000406, 0001, 0000 0 0 0 0 0 0 0 0 0 0
352  0257,  0436957,  0437441,  0000484, 0001, 0000 0 0 0 0 0 0 0 0 0 0
353  0271,  0437441,  0442873,  0005432, 0002, 0000 9 0 0 0 0 0 0 0 0 0
355  0257,  0442873,  0443422,  0000549, 0001, 0000 0 0 0 0 0 0 0 0 0 0
356  0257,  0443422,  0443757,  0000335, 0001, 0000 0 0 0 0 0 0 0 0 0 0
357  0257,  0443757,  0444213,  0000456, 0001, 0000 0 0 0 0 0 0 0 0 0 0
358  0271,  0444213,  0452688,  0008475, 0002, 0000 12 0 0 0 0 0 0 0 0 0
360  0257,  0452688,  0453512,  0000824, 0001, 0000 0 0 0 0 0 0 0 0 0 0
361  0257,  0453512,  0454012,  0000500, 0001, 0000 0 0 0 0 0 0 0 0 0 0
362  0257,  0454012,  0454533,  0000521, 0001, 0000 0 0 0 0 0 0 0 0 0 0
363  0271,  0454533,  0457972,  0003439, 0002, 0000 13 0 0 0 0 0 0 0 0 0
365  0257,  0457972,  0458422,  0000450, 0001, 0000 0 0 0 0 0 0 0 0 0 0
366  0257,  0458422,  0458796,  0000374, 0001, 0000 0 0 0 0 0 0 0 0 0 0
367  0257,  0458796,  0459323,  0000527, 0001, 0000 0 0 0 0 0 0 0 0 0 0
368  0271,  0459323,  0465002,  0005679, 0002, 0000 10 0 0 0 0 0 0 0 0 0
370  0257,  0465002,  0465464,  0000462, 0001, 0000 0 0 0 0 0 0 0 0 0 0
371  0257,  0465464,  0465947,  0000483, 0001, 0000 0 0 0 0 0 0 0 0 0 0
372  0257,  0465947,  0466447,  0000500, 0001, 0000 0 0 0 0 0 0 0 0 0 0
373  0271,  0466447,  0469363,  0002916, 0002, 0000 14 0 0 0 0 0 0 0 0 0
375  0257,  0469363,  0470242,  0000879, 0001, 0000 0 0 0 0 0 0 0 0 0 0
376  0257,  0470242,  0470890,  0000648, 0001, 0000 0 0 0 0 0 0 0 0 0 0
377  0257,  0470890,  0471423,  0000533, 0001, 0000 0 0 0 0 0 0 0 0 0 0
378  0271,  0471423,  0476992,  0005569, 0002, 0000 15 0 0 0 0 0 0 0 0 0
380  0257,  0476992,  0477613,  0000621, 0001, 0000 0 0 0 0 0 0 0 0 0 0
381  0287,  0477613,  0477992,  0000379, 0001, 0000 0 0 0 0 0 0 0 0 0 0
382  0257,  0477992,  0478234,  0000242, 0001, 0000 0 0 0 0 0 0 0 0 0 0
383  0257,  0478234,  0478322,  0000088, 0001, 0000 0 0 0 0 0 0 0 0 0 0
384  0257,  0478322,  0478673,  0000351, 0001, 0000 0 0 0 0 0 0 0 0 0 0
385  0257,  0478673,  0479178,  0000505, 0001, 0000 0 0 0 0 0 0 0 0 0 0
386  0271,  0479178,  0481617,  0002439, 0002, 0000 16 0 0 0 0 0 0 0 0 0
388  0257,  0481617,  0482309,  0000692, 0001, 0000 0 0 0 0 0 0 0 0 0 0
389  0257,  0482309,  0482611,  0000302, 0001, 0000 0 0 0 0 0 0 0 0 0 0
390  0257,  0482611,  0483122,  0000511, 0001, 0000 0 0 0 0 0 0 0 0 0 0
391  0271,  0483122,  0488302,  0005180, 0002, 0000 17 0 0 0 0 0 0 0 0 0
```

```
393  0257,  0488302,  0488774,  0000472, 0001, 0000 0 0 0 0 0 0 0 0 0 0
394  0257,  0488774,  0489049,  0000275, 0001, 0000 0 0 0 0 0 0 0 0 0 0
395  0257,  0489049,  0489581,  0000532, 0001, 0000 0 0 0 0 0 0 0 0 0 0
396  0271,  0489581,  0493701,  0004120, 0002, 0000 22 0 0 0 0 0 0 0 0 0
398  0257,  0493701,  0494206,  0000505, 0001, 0000 0 0 0 0 0 0 0 0 0 0
399  0257,  0494206,  0494552,  0000346, 0001, 0000 0 0 0 0 0 0 0 0 0 0
400  0257,  0494552,  0495085,  0000533, 0001, 0000 0 0 0 0 0 0 0 0 0 0
401  0271,  0495085,  0501352,  0006267, 0002, 0000 23 0 0 0 0 0 0 0 0 0
403  0257,  0501352,  0501835,  0000483, 0001, 0000 0 0 0 0 0 0 0 0 0 0
404  0257,  0501835,  0503022,  0001187, 0001, 0000 0 0 0 0 0 0 0 0 0 0
405  0257,  0503022,  0503543,  0000521, 0001, 0000 0 0 0 0 0 0 0 0 0 0
406  0271,  0503543,  0508262,  0004719, 0002, 0000 19 0 0 0 0 0 0 0 0 0
408  0257,  0508262,  0508662,  0000400, 0001, 0000 0 0 0 0 0 0 0 0 0 0
409  0257,  0508662,  0508965,  0000303, 0001, 0000 0 0 0 0 0 0 0 0 0 0
410  0257,  0508965,  0509464,  0000499, 0001, 0000 0 0 0 0 0 0 0 0 0 0
411  0257,  0509464,  0509470,  0000006, 0001, 0000 0 0 0 0 0 0 0 0 0 0
412  0271,  0509470,  0516632,  0007162, 0002, 0000 20 0 0 0 0 0 0 0 0 0
414  0257,  0516632,  0517280,  0000648, 0001, 0000 0 0 0 0 0 0 0 0 0 0
415  0257,  0517280,  0517610,  0000330, 0001, 0000 0 0 0 0 0 0 0 0 0 0
416  0257,  0517610,  0518159,  0000549, 0001, 0000 0 0 0 0 0 0 0 0 0 0
417  0271,  0518159,  0526607,  0008448, 0002, 0000 21 0 0 0 0 0 0 0 0 0
419  0257,  0526607,  0527079,  0000472, 0001, 0000 0 0 0 0 0 0 0 0 0 0
420  0257,  0527079,  0527343,  0000264, 0001, 0000 0 0 0 0 0 0 0 0 0 0
421  0257,  0527343,  0527864,  0000521, 0001, 0000 0 0 0 0 0 0 0 0 0 0
422  0271,  0527864,  0530775,  0002911, 0002, 0000 25 0 0 0 0 0 0 0 0 0
424  0257,  0530775,  0531220,  0000445, 0001, 0000 0 0 0 0 0 0 0 0 0 0
425  0257,  0531220,  0531566,  0000346, 0001, 0000 0 0 0 0 0 0 0 0 0 0
426  0257,  0531566,  0532077,  0000511, 0001, 0000 0 0 0 0 0 0 0 0 0 0
427  0271,  0532077,  0536026,  0003949, 0002, 0000 26 0 0 0 0 0 0 0 0 0
429  0257,  0536026,  0536400,  0000374, 0001, 0000 0 0 0 0 0 0 0 0 0 0
430  0257,  0536400,  0536674,  0000274, 0001, 0000 0 0 0 0 0 0 0 0 0 0
431  0257,  0536674,  0537202,  0000528, 0001, 0000 0 0 0 0 0 0 0 0 0 0
432  0257,  0537202,  0537207,  0000005, 0001, 0000 0 0 0 0 0 0 0 0 0 0
433  0257,  0537207,  0537213,  0000006, 0001, 0000 0 0 0 0 0 0 0 0 0 0
434  0271,  0537213,  0542799,  0005586, 0002, 0000 27 0 0 0 0 0 0 0 0 0
436  0257,  0542799,  0543342,  0000543, 0001, 0000 0 0 0 0 0 0 0 0 0 0
437  0279,  0543342,  0543348,  0000006, 0001, 0000 0 0 -1318 211 -87 -366 418 40 618 140
438  0257,  0543348,  0543381,  0000033, 0001, 0000 0 0 0 0 0 0 0 0 0 0
439  0281,  0543381,  0543386,  0000005, 0001, 0002 0 0 0 0 0 0 0 0 0 0
440  0257,  0543386,  0543612,  0000226, 0001, 0000 0 0 0 0 0 0 0 0 0 0
441  0283,  0543612,  0543826,  0000214, 0001, 0000 0 0 1482 -10 1849 -259 -1276 183 1832 -329
442  0257,  0543826,  0544408,  0000582, 0001, 0000 0 0 0 0 0 0 0 0 0 0
443  0283,  0544408,  0544534,  0000126, 0001, 0000 0 0 -1276 183 1832 -329 -1276 -110 1936 -256
444  0257,  0544534,  0544930,  0000396, 0001, 0000 0 0 0 0 0 0 0 0 0 0
445  0283,  0544930,  0545012,  0000082, 0001, 0000 0 0 -1276 -110 1936 -256 -650 19 -171 -293
446  0257,  0545012,  0545798,  0000786, 0001, 0000 0 0 0 0 0 0 0 0 0 0
447  0287,  0545798,  0546210,  0000412, 0001, 0000 0 0 0 0 0 0 0 0 0 0
448  0257,  0546210,  0546874,  0000664, 0001, 0000 0 0 0 0 0 0 0 0 0 0
449  0266,  0546874,  0546880,  0000006, 0001, 0000 0 0 -1276 -110 1936 -256 10 36 245 236
450  0257,  0546880,  0546924,  0000044, 0001, 0000 0 0 0 0 0 0 0 0 0 0
451  0268,  0546924,  0546929,  0000005, 0001, 0004 0 0 0 0 0 0 0 0 0 0
452  0257,  0546929,  0548088,  0001209, 0001, 0000 0 0 0 0 0 0 0 0 0 0
453  0287,  0548088,  0549780,  0001692, 0001, 0000 0 0 0 0 0 0 0 0 0 0
454  0257,  0549780,  0550192,  0000412, 0001, 0000 0 0 0 0 0 0 0 0 0 0
455  0287,  0550192,  0551911,  0001719, 0001, 0000 0 0 0 0 0 0 0 0 0 0
456  0257,  0551911,  0552279,  0000368, 0001, 0000 0 0 0 0 0 0 0 0 0 0
```

B.4 Output from Pass 2

Pass 2 reduces the Pass 1 input stream from 425 lines to 284 lines.

```
0257,  0316604,  0317258,  0000654, 0004, 0000 0 0 0 0 0 0 0 0 0 0
0286,  0317258,  0318571,  0001313, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 1 0 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 2 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 1 2 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 3 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 3 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 4 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 4 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 5 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 5 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 6 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 6 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 7 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 7 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 8 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 8 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 9 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 9 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 10 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 10 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 11 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 11 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 12 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 12 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 13 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 13 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 14 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 14 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 15 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 15 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 16 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 2 16 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 17 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 1 17 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 18 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 1 18 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 19 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 1 19 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 20 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 19 24 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 25 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 19 20 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 26 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 19 21 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 27 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 19 22 0 0 0 0 0 0 0 0
0264, -0000001, -0000001, -0000001, 0001, 0000 28 0 0 0 0 0 0 0 0 0
0274, -0000001, -0000001, -0000001, 0001, 0000 19 23 0 0 0 0 0 0 0 0
2005,  0318571,  0322707,  0004136, 0008, 0002 0 0 0 0 0 0 0 0 0 0
0285,  0322707,  0322712,  0000005, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 1 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 2 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 3 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 4 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 5 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 6 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 7 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 8 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 9 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 10 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 11 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 12 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 13 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 14 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 15 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 16 0 0 0 0 0 0 0 0 0
0258, -0000001, -0000001, -0000001, 0001, 0000 2 16 0 0 0 0 0 0 0 0
0258, -0000001, -0000001, -0000001, 0001, 0000 2 15 0 0 0 0 0 0 0 0
```

```
0258, -0000001, -0000001, -0000001, 0001, 0000 2 14 0 0 0 0 0 0 0 0
0258, -0000001, -0000001, -0000001, 0001, 0000 2 13 0 0 0 0 0 0 0 0
0258, -0000001, -0000001, -0000001, 0001, 0000 2 12 0 0 0 0 0 0 0 0
0258, -0000001, -0000001, -0000001, 0001, 0000 2 11 0 0 0 0 0 0 0 0
0258, -0000001, -0000001, -0000001, 0001, 0000 2 10 0 0 0 0 0 0 0 0
0258, -0000001, -0000001, -0000001, 0001, 0000 2 9 0 0 0 0 0 0 0 0
0258, -0000001, -0000001, -0000001, 0001, 0000 2 8 0 0 0 0 0 0 0 0
0258, -0000001, -0000001, -0000001, 0001, 0000 2 7 0 0 0 0 0 0 0 0
0258, -0000001, -0000001, -0000001, 0001, 0000 2 6 0 0 0 0 0 0 0 0
0258, -0000001, -0000001, -0000001, 0001, 0000 2 5 0 0 0 0 0 0 0 0
0258, -0000001, -0000001, -0000001, 0001, 0000 2 4 0 0 0 0 0 0 0 0
0258, -0000001, -0000001, -0000001, 0001, 0000 2 3 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 17 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 18 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 19 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 24 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 20 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 21 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 22 0 0 0 0 0 0 0 0 0
0265, -0000001, -0000001, -0000001, 0001, 0000 23 0 0 0 0 0 0 0 0 0
0258, -0000001, -0000001, -0000001, 0001, 0000 19 23 0 0 0 0 0 0 0 0
0258, -0000001, -0000001, -0000001, 0001, 0000 19 22 0 0 0 0 0 0 0 0
0258, -0000001, -0000001, -0000001, 0001, 0000 19 21 0 0 0 0 0 0 0 0
0258, -0000001, -0000001, -0000001, 0001, 0000 19 20 0 0 0 0 0 0 0 0
0258, -0000001, -0000001, -0000001, 0001, 0000 19 24 0 0 0 0 0 0 0 0
0258, -0000001, -0000001, -0000001, 0001, 0000 1 19 0 0 0 0 0 0 0 0
0258, -0000001, -0000001, -0000001, 0001, 0000 1 18 0 0 0 0 0 0 0 0
0258, -0000001, -0000001, -0000001, 0001, 0000 1 17 0 0 0 0 0 0 0 0
0258, -0000001, -0000001, -0000001, 0001, 0000 1 2 0 0 0 0 0 0 0 0
2008,  0322712,  0324733,  0002021, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0264,  0324733,  0324744,  0000011, 0001, 0000 1 0 293 -29 0 0 0 0 0 0
0270,  0324744,  0325343,  0000599, 0001, 0000 1 0 0 0 0 0 0 0 0 0
0257,  0325343,  0325920,  0000577, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0264,  0325920,  0325936,  0000016, 0001, 0000 2 0 99 -84 0 0 0 0 0 0
0270,  0325936,  0326678,  0000742, 0001, 0000 2 0 0 0 0 0 0 0 0 0
0257,  0326678,  0326925,  0000247, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0264,  0326925,  0326936,  0000011, 0001, 0000 3 0 257 -80 0 0 0 0 0 0
0270,  0326936,  0328683,  0001747, 0001, 0000 3 0 0 0 0 0 0 0 0 0
0257,  0328683,  0328902,  0000219, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0264,  0328902,  0328913,  0000011, 0001, 0000 4 0 396 -81 0 0 0 0 0 0
0270,  0328913,  0329902,  0000989, 0001, 0000 4 0 0 0 0 0 0 0 0 0
2008,  0329902,  0331802,  0001900, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0331802,  0331879,  0000077, 0001, 0000 1 2 0 0 0 0 0 0 0 0
0257,  0331879,  0331967,  0000088, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0331967,  0332022,  0000055, 0001, 0000 1 3 0 0 0 0 0 0 0 0
0257,  0332022,  0332093,  0000071, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0332093,  0332165,  0000072, 0001, 0000 1 4 0 0 0 0 0 0 0 0
2002,  0332165,  0332840,  0000675, 0003, 0032 0 0 0 0 0 0 0 0 0 0
0264,  0332840,  0332857,  0000017, 0001, 0000 5 0 24 -179 0 0 0 0 0 0
0270,  0332857,  0333406,  0000549, 0001, 0000 5 0 0 0 0 0 0 0 0 0
0257,  0333406,  0334247,  0000841, 0002, 0000 0 0 0 0 0 0 0 0 0 0
0264,  0334247,  0334263,  0000016, 0001, 0000 6 0 152 -176 0 0 0 0 0 0
0270,  0334263,  0335186,  0000923, 0001, 0000 6 0 0 0 0 0 0 0 0 0
0257,  0335186,  0335318,  0000132, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0335318,  0335378,  0000060, 0001, 0000 2 5 0 0 0 0 0 0 0 0
0257,  0335378,  0335460,  0000082, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0335460,  0335537,  0000077, 0001, 0000 2 6 0 0 0 0 0 0 0 0
0257,  0335537,  0335675,  0000138, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0264,  0335675,  0335686,  0000011, 0001, 0000 7 0 279 -181 0 0 0 0 0 0
0270,  0335686,  0336630,  0000944, 0001, 0256 7 0 0 0 0 0 0 0 0 0
0257,  0336630,  0337015,  0000385, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0265,  0337015,  0337020,  0000005, 0001, 0000 7 0 0 0 0 0 0 0 0 0
0257,  0337020,  0337229,  0000209, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0264,  0337229,  0337240,  0000011, 0001, 0000 8 0 279 -180 0 0 0 0 0 0
0270,  0337240,  0338300,  0001060, 0001, 0000 8 0 0 0 0 0 0 0 0 0
0257,  0338300,  0338454,  0000154, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0264,  0338454,  0338465,  0000011, 0001, 0000 9 0 399 -175 0 0 0 0 0 0
0270,  0338465,  0339272,  0000807, 0001, 0000 9 0 0 0 0 0 0 0 0 0
0257,  0339272,  0339871,  0000599, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0339871,  0339942,  0000071, 0001, 0000 3 8 0 0 0 0 0 0 0 0
0257,  0339942,  0340014,  0000072, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0340014,  0340069,  0000055, 0001, 0000 3 9 0 0 0 0 0 0 0 0
2002,  0340069,  0340904,  0000835, 0003, 0032 0 0 0 0 0 0 0 0 0 0
0264,  0340904,  0340909,  0000005, 0001, 0000 10 0 557 -187 0 0 0 0 0 0
0270,  0340909,  0343172,  0002263, 0001, 0000 10 0 0 0 0 0 0 0 0 0
```

```
0257,  0343172,  0343364,  0000192, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0343364,  0343458,  0000094, 0001, 0000 3 10 0 0 0 0 0 0 0 0
2005,  0343458,  0346160,  0002702, 0009, 0034 0 0 0 0 0 0 0 0 0 0
0264,  0346160,  0346165,  0000005, 0001, 0000 11 0 717 -182 0 0 0 0 0 0
0270,  0346165,  0346759,  0000594, 0001, 0000 11 0 0 0 0 0 0 0 0 0
0257,  0346759,  0346940,  0000181, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0346940,  0347028,  0000088, 0001, 0000 3 11 0 0 0 0 0 0 0 0
0257,  0347028,  0347187,  0000159, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0264,  0347187,  0347198,  0000011, 0001, 0000 12 0 839 -177 0 0 0 0 0 0
0270,  0347198,  0348511,  0001313, 0001, 0000 12 0 0 0 0 0 0 0 0 0
0257,  0348511,  0348659,  0000148, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0348659,  0348752,  0000093, 0001, 0000 3 12 0 0 0 0 0 0 0 0
2005,  0348752,  0349807,  0001055, 0005, 0032 0 0 0 0 0 0 0 0 0 0
0264,  0349807,  0349812,  0000005, 0001, 0000 13 0 926 -173 0 0 0 0 0 0
0270,  0349812,  0350505,  0000693, 0001, 0000 13 0 0 0 0 0 0 0 0 0
0257,  0350505,  0350653,  0000148, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0264,  0350653,  0350664,  0000011, 0001, 0000 14 0 1063 -180 0 0 0 0 0 0
0270,  0350664,  0352345,  0001681, 0001, 0000 14 0 0 0 0 0 0 0 0 0
0257,  0352345,  0352542,  0000197, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0352542,  0352652,  0000110, 0001, 0000 4 14 0 0 0 0 0 0 0 0
0257,  0352652,  0352751,  0000099, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0352751,  0352828,  0000077, 0001, 0000 3 13 0 0 0 0 0 0 0 0
2002,  0352828,  0353481,  0000653, 0003, 0032 0 0 0 0 0 0 0 0 0 0
0264,  0353481,  0353492,  0000011, 0001, 0000 15 0 1016 -182 0 0 0 0 0 0
0270,  0353492,  0353849,  0000357, 0001, 0000 15 0 0 0 0 0 0 0 0 0
0257,  0353849,  0354020,  0000171, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0264,  0354020,  0354031,  0000011, 0001, 0000 16 0 1122 -178 0 0 0 0 0 0
0270,  0354031,  0354388,  0000357, 0001, 0000 16 0 0 0 0 0 0 0 0 0
0257,  0354388,  0354536,  0000148, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0354536,  0354602,  0000066, 0001, 0000 4 15 0 0 0 0 0 0 0 0
0257,  0354602,  0354690,  0000088, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0354690,  0354789,  0000099, 0001, 0000 4 16 0 0 0 0 0 0 0 0
2003,  0354789,  0355865,  0001076, 0003, 0032 0 0 0 0 0 0 0 0 0 0
0264,  0355865,  0355882,  0000017, 0001, 0000 17 0 1223 -180 0 0 0 0 0 0
0270,  0355882,  0356244,  0000362, 0001, 0000 17 0 0 0 0 0 0 0 0 0
0257,  0356244,  0356442,  0000198, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0264,  0356442,  0356453,  0000011, 0001, 0000 18 0 1364 -180 0 0 0 0 0 0
0270,  0356453,  0359293,  0002840, 0001, 0000 18 0 0 0 0 0 0 0 0 0
2005,  0359293,  0360402,  0001109, 0003, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0360402,  0360545,  0000143, 0001, 0000 1 18 0 0 0 0 0 0 0 0
2002,  0360545,  0361188,  0000643, 0003, 0032 0 0 0 0 0 0 0 0 0 0
0274,  0361188,  0361281,  0000093, 0001, 0000 4 17 0 0 0 0 0 0 0 0
2002,  0361281,  0362094,  0000813, 0003, 0032 0 0 0 0 0 0 0 0 0 0
0264,  0362094,  0362105,  0000011, 0001, 0000 19 0 1329 -187 0 0 0 0 0 0
0270,  0362105,  0363071,  0000966, 0001, 0000 19 0 0 0 0 0 0 0 0 0
0257,  0363071,  0363231,  0000160, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0264,  0363231,  0363242,  0000011, 0001, 0000 20 0 1467 -183 0 0 0 0 0 0
0270,  0363242,  0363758,  0000516, 0001, 0000 20 0 0 0 0 0 0 0 0 0
0257,  0363758,  0364044,  0000286, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0364044,  0364115,  0000071, 0001, 0000 18 19 0 0 0 0 0 0 0 0
0257,  0364115,  0364203,  0000088, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0364203,  0364307,  0000104, 0001, 0000 18 20 0 0 0 0 0 0 0 0
2005,  0364307,  0369306,  0004999, 0005, 0032 0 0 0 0 0 0 0 0 0 0
0264,  0369306,  0369316,  0000010, 0001, 0000 21 0 1422 -179 0 0 0 0 0 0
0270,  0369316,  0369586,  0000270, 0001, 0000 21 0 0 0 0 0 0 0 0 0
0257,  0369586,  0369723,  0000137, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0369723,  0369789,  0000066, 0001, 0000 18 21 0 0 0 0 0 0 0 0
2008,  0369789,  0372151,  0002362, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0264,  0372151,  0372162,  0000011, 0001, 0000 22 0 1033 -137 0 0 0 0 0 0
0270,  0372162,  0372628,  0000466, 0001, 0000 22 0 0 0 0 0 0 0 0 0
0257,  0372628,  0372766,  0000138, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0372766,  0372837,  0000071, 0001, 0000 4 22 0 0 0 0 0 0 0 0
2002,  0372837,  0373469,  0000632, 0003, 0032 0 0 0 0 0 0 0 0 0 0
0264,  0373469,  0373480,  0000011, 0001, 0000 23 0 1077 -145 0 0 0 0 0 0
0270,  0373480,  0373881,  0000401, 0001, 0000 23 0 0 0 0 0 0 0 0 0
0257,  0373881,  0374018,  0000137, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0374018,  0374095,  0000077, 0001, 0000 4 23 0 0 0 0 0 0 0 0
2005,  0374095,  0375737,  0001642, 0005, 0032 0 0 0 0 0 0 0 0 0 0
0264,  0375737,  0375743,  0000006, 0001, 0000 24 0 1724 -97 0 0 0 0 0 0
0270,  0375743,  0376429,  0000686, 0001, 0000 24 0 0 0 0 0 0 0 0 0
2005,  0376429,  0377182,  0000753, 0003, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0377182,  0377319,  0000137, 0001, 0000 1 24 0 0 0 0 0 0 0 0
2002,  0377319,  0378159,  0000840, 0003, 0032 0 0 0 0 0 0 0 0 0 0
0264,  0378159,  0378165,  0000006, 0001, 0000 25 0 1657 -179 0 0 0 0 0 0
0270,  0378165,  0378670,  0000505, 0001, 0000 25 0 0 0 0 0 0 0 0 0
```

```
0257,  0378670,  0378797,  0000127, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0378797,  0378868,  0000071, 0001, 0000 24 25 0 0 0 0 0 0 0 0
0257,  0378868,  0378934,  0000066, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0264,  0378934,  0378945,  0000011, 0001, 0000 26 0 1817 -131 0 0 0 0 0 0
0270,  0378945,  0379653,  0000708, 0001, 0000 26 0 0 0 0 0 0 0 0 0
0257,  0379653,  0379846,  0000193, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0264,  0379846,  0379862,  0000016, 0001, 0000 27 0 1885 -118 0 0 0 0 0 0
0270,  0379862,  0380631,  0000769, 0001, 0000 27 0 0 0 0 0 0 0 0 0
0257,  0380631,  0380763,  0000132, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0380763,  0380856,  0000093, 0001, 0000 24 26 0 0 0 0 0 0 0 0
0257,  0380856,  0380955,  0000099, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0380955,  0381054,  0000099, 0001, 0000 24 27 0 0 0 0 0 0 0 0
2005,  0381054,  0390463,  0009409, 0011, 0032 0 0 0 0 0 0 0 0 0 0
0258,  0390463,  0390468,  0000005, 0001, 0000 3 10 0 0 0 0 0 0 0 0
0257,  0390468,  0390611,  0000143, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0258,  0390611,  0390677,  0000066, 0001, 0000 3 11 0 0 0 0 0 0 0 0
0257,  0390677,  0391182,  0000505, 0001, 0008 0 0 0 0 0 0 0 0 0 0
0265,  0391182,  0391852,  0000670, 0005, 0008 11 0 0 0 0 0 0 0 0 0
2005,  0391852,  0392665,  0000813, 0003, 0000 0 0 0 0 0 0 0 0 0 0
0274,  0392665,  0392726,  0000061, 0001, 0000 4 10 0 0 0 0 0 0 0 0
2005,  0392726,  0399921,  0007195, 0007, 2080 0 0 0 0 0 0 0 0 0 0
0264,  0399921,  0399932,  0000011, 0001, 0000 28 0 -1063 -321 0 0 0 0 0 0
0257,  0399932,  0400432,  0000500, 0001, 0000 0 0 0 0 0 0 0 0 0 0
0265,  0400432,  0400437,  0000005, 0001, 0000 28 0 0 0 0 0 0 0 0 0
2005,  0400437,  0403162,  0002725, 0006, 0000 0 0 0 0 0 0 0 0 0 0
0271,  0403162,  0409983,  0006821, 0002, 0000 5 0 0 0 0 0 0 0 0 0
0257,  0409983,  0411296,  0001313, 0003, 0000 0 0 0 0 0 0 0 0 0 0
0271,  0411296,  0423385,  0012089, 0002, 0000 6 0 0 0 0 0 0 0 0 0
0257,  0423385,  0424533,  0001148, 0004, 0000 0 0 0 0 0 0 0 0 0 0
0271,  0424533,  0430789,  0006256, 0002, 0000 5 0 0 0 0 0 0 0 0 0
2008,  0430789,  0433233,  0002444, 0004, 0000 0 0 0 0 0 0 0 0 0 0
0271,  0433233,  0436100,  0002867, 0002, 0000 8 0 0 0 0 0 0 0 0 0
0257,  0436100,  0437441,  0001341, 0003, 0000 0 0 0 0 0 0 0 0 0 0
0271,  0437441,  0442873,  0005432, 0002, 0000 9 0 0 0 0 0 0 0 0 0
0257,  0442873,  0444213,  0001340, 0003, 0000 0 0 0 0 0 0 0 0 0 0
0271,  0444213,  0452688,  0008475, 0002, 0000 12 0 0 0 0 0 0 0 0 0
2008,  0452688,  0454533,  0001845, 0003, 0000 0 0 0 0 0 0 0 0 0 0
0271,  0454533,  0457972,  0003439, 0002, 0000 13 0 0 0 0 0 0 0 0 0
0257,  0457972,  0459323,  0001351, 0003, 0000 0 0 0 0 0 0 0 0 0 0
0271,  0459323,  0465002,  0005679, 0002, 0000 10 0 0 0 0 0 0 0 0 0
0257,  0465002,  0466447,  0001445, 0003, 0000 0 0 0 0 0 0 0 0 0 0
0271,  0466447,  0469363,  0002916, 0002, 0000 14 0 0 0 0 0 0 0 0 0
2008,  0469363,  0471423,  0002060, 0003, 0000 0 0 0 0 0 0 0 0 0 0
0271,  0471423,  0476992,  0005569, 0002, 0000 15 0 0 0 0 0 0 0 0 0
2004,  0476992,  0479178,  0002186, 0006, 2048 0 0 0 0 0 0 0 0 0 0
0271,  0479178,  0481617,  0002439, 0002, 0000 16 0 0 0 0 0 0 0 0 0
2008,  0481617,  0483122,  0001505, 0003, 0000 0 0 0 0 0 0 0 0 0 0
0271,  0483122,  0488302,  0005180, 0002, 0000 17 0 0 0 0 0 0 0 0 0
0257,  0488302,  0489581,  0001279, 0003, 0000 0 0 0 0 0 0 0 0 0 0
0271,  0489581,  0493701,  0004120, 0002, 0000 22 0 0 0 0 0 0 0 0 0
0257,  0493701,  0495085,  0001384, 0003, 0000 0 0 0 0 0 0 0 0 0 0
0271,  0495085,  0501352,  0006267, 0002, 0000 23 0 0 0 0 0 0 0 0 0
2008,  0501352,  0503543,  0002191, 0003, 0000 0 0 0 0 0 0 0 0 0 0
0271,  0503543,  0508262,  0004719, 0002, 0000 19 0 0 0 0 0 0 0 0 0
0257,  0508262,  0509470,  0001208, 0004, 0000 0 0 0 0 0 0 0 0 0 0
0271,  0509470,  0516632,  0007162, 0002, 0000 20 0 0 0 0 0 0 0 0 0
2008,  0516632,  0518159,  0001527, 0003, 0000 0 0 0 0 0 0 0 0 0 0
0271,  0518159,  0526607,  0008448, 0002, 0000 21 0 0 0 0 0 0 0 0 0
0257,  0526607,  0527864,  0001257, 0003, 0000 0 0 0 0 0 0 0 0 0 0
0271,  0527864,  0530775,  0002911, 0002, 0000 25 0 0 0 0 0 0 0 0 0
0257,  0530775,  0532077,  0001302, 0003, 0000 0 0 0 0 0 0 0 0 0 0
0271,  0532077,  0536026,  0003949, 0002, 0000 26 0 0 0 0 0 0 0 0 0
0257,  0536026,  0537213,  0001187, 0005, 0000 0 0 0 0 0 0 0 0 0 0
0271,  0537213,  0542799,  0005586, 0002, 0000 27 0 0 0 0 0 0 0 0 0
2005,  0542799,  0552279,  0009480, 0021, 2054 0 0 0 0 0 0 0 0 0 0
```

B.5 Output from Pass 3

Pass 3 reduces the Pass 2 input stream from 284 lines to 64 lines.

```
3012,  0316604,  0318571,  0001967, 0005,
3002,  0316604,  0318571,  0001967, 0005,  1 1 23 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 24 20 21 22 23
3001,  0316604,  0318571,  0001967, 0005,  24 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 25 26 27 28
2005,  0318571,  0322707,  0004136, 0008, 0002 0 0 0 0 0 0 0 0 0 0
3012,  0322707,  0322712,  0000005, 0001,
2008,  0322712,  0324733,  0002021, 0001, 0000 0 0 0 0 0 0 0 0 0 0
3001,  0324733,  0329902,  0005169, 0011,  4 1 2 3 4
2008,  0329902,  0331802,  0001900, 0001, 0000 0 0 0 0 0 0 0 0 0 0
3002,  0331802,  0332165,  0000363, 0005,  1 1 3 2 3 4
2002,  0332165,  0332840,  0000675, 0003, 0032 0 0 0 0 0 0 0 0 0 0
3003,  0332840,  0340069,  0007229, 0025,  1 1 4 5 6 8 9
2002,  0340069,  0340904,  0000835, 0003, 0032 0 0 0 0 0 0 0 0 0 0
3003,  0340904,  0343458,  0002554, 0004,  1 1 1 10
2005,  0343458,  0346160,  0002702, 0009, 0034 0 0 0 0 0 0 0 0 0 0
3003,  0346160,  0348752,  0002592, 0009,  1 1 2 11 12
2005,  0348752,  0349807,  0001055, 0005, 0032 0 0 0 0 0 0 0 0 0 0
3003,  0349807,  0352828,  0003021, 0009,  1 1 2 14 13
2002,  0352828,  0353481,  0000653, 0003, 0032 0 0 0 0 0 0 0 0 0 0
3003,  0353481,  0354789,  0001308, 0009,  1 1 2 15 16
2003,  0354789,  0355865,  0001076, 0003, 0032 0 0 0 0 0 0 0 0 0 0
3001,  0355865,  0359293,  0003428, 0005,  2 17 18
2005,  0359293,  0360402,  0001109, 0003, 0000 0 0 0 0 0 0 0 0 0 0
3010,  0360402,  0360545,  0000143, 0001,  1 1 1 18
2002,  0360545,  0361188,  0000643, 0003, 0032 0 0 0 0 0 0 0 0 0 0
3010,  0361188,  0361281,  0000093, 0001,  1 1 1 17
2002,  0361281,  0362094,  0000813, 0003, 0032 0 0 0 0 0 0 0 0 0 0
3003,  0362094,  0364307,  0002213, 0009,  1 1 2 19 20
2005,  0364307,  0369306,  0004999, 0005, 0032 0 0 0 0 0 0 0 0 0 0
3003,  0369306,  0369789,  0000483, 0004,  1 1 1 21
2008,  0369789,  0372151,  0002362, 0001, 0000 0 0 0 0 0 0 0 0 0 0
3003,  0372151,  0372837,  0000686, 0004,  1 1 1 22
2002,  0372837,  0373469,  0000632, 0003, 0032 0 0 0 0 0 0 0 0 0 0
3003,  0373469,  0374095,  0000626, 0004,  1 1 1 23
2005,  0374095,  0375737,  0001642, 0005, 0032 0 0 0 0 0 0 0 0 0 0
3001,  0375737,  0376429,  0000692, 0002,  1 24
2005,  0376429,  0377182,  0000753, 0003, 0000 0 0 0 0 0 0 0 0 0 0
3010,  0377182,  0377319,  0000137, 0001,  1 1 1 24
2002,  0377319,  0378159,  0000840, 0003, 0032 0 0 0 0 0 0 0 0 0 0
3003,  0378159,  0381054,  0002895, 0014,  1 1 3 25 26 27
2005,  0381054,  0390463,  0009409, 0011, 0032 0 0 0 0 0 0 0 0 0 0
3005,  0390463,  0391852,  0001389, 0009,
3006,  0390463,  0391852,  0001389, 0009,  1 11 0
2005,  0391852,  0392665,  0000813, 0003, 0000 0 0 0 0 0 0 0 0 0 0
3010,  0392665,  0392726,  0000061, 0001,  1 1 1 10
2005,  0392726,  0399921,  0007195, 0007, 2080 0 0 0 0 0 0 0 0 0 0
3011,  0399921,  0400437,  0000516, 0003,
2005,  0400437,  0403162,  0002725, 0006, 0000 0 0 0 0 0 0 0 0 0 0
3007,  0403162,  0430789,  0027627, 0013,
3008,  0403162,  0430789,  0027627, 0013,
2008,  0430789,  0433233,  0002444, 0004, 0000 0 0 0 0 0 0 0 0 0 0
3007,  0433233,  0452688,  0019455, 0012,
2008,  0452688,  0454533,  0001845, 0003, 0000 0 0 0 0 0 0 0 0 0 0
3007,  0454533,  0469363,  0014830, 0012,
2008,  0469363,  0471423,  0002060, 0003, 0000 0 0 0 0 0 0 0 0 0 0
3007,  0471423,  0476992,  0005569, 0002,
2004,  0476992,  0479178,  0002186, 0006, 2048 0 0 0 0 0 0 0 0 0 0
3007,  0479178,  0481617,  0002439, 0002,
2008,  0481617,  0483122,  0001505, 0003, 0000 0 0 0 0 0 0 0 0 0 0
3007,  0483122,  0501352,  0018230, 0012,
2008,  0501352,  0503543,  0002191, 0003, 0000 0 0 0 0 0 0 0 0 0 0
3007,  0503543,  0516632,  0013089, 0008,
2008,  0516632,  0518159,  0001527, 0003, 0000 0 0 0 0 0 0 0 0 0 0
3007,  0518159,  0542799,  0024640, 0019,
2005,  0542799,  0552279,  0009480, 0021, 2054 0 0 0 0 0 0 0 0 0 0
```

B.6 Output from Pass 4

Pass 4 reduces the Pass 3 input stream from 64 lines to 10 lines.

```
4002,  0316604,  0322707,  0006103, 0013, 1026 0 0 0 0 0 0 0 0 0 0
4007,  0322707,  0324733,  0002026, 0002, 0002 0 0 0 0 0 0 0 0 0 0
4001,  0324733,  0331802,  0007069, 0012, 0000 0 0 0 0 0 0 0 0 0 0
4002,  0331802,  0355865,  0024063, 0087, 0034 0 0 0 0 0 0 0 0 0 0
4001,  0355865,  0360402,  0004537, 0008, 0032 0 0 0 0 0 0 0 0 0 0
4003,  0360402,  0375737,  0015335, 0043, 0032 0 0 0 0 0 0 0 0 0 0
4001,  0375737,  0377182,  0001445, 0005, 0032 0 0 0 0 0 0 0 0 0 0
4003,  0377182,  0390463,  0013281, 0029, 0032 0 0 0 0 0 0 0 0 0 0
4005,  0390463,  0392665,  0002202, 0012, 1056 0 0 0 0 0 0 0 0 0 0
4006,  0392665,  0552279,  0159614, 0143, 3110 0 0 0 0 0 0 0 0 0 0
```

B.7 Output from Pass 5

This final pass of the parser combined the results of the previous passes to produce a parse tree and a summary of the session.

B.7.1 Parse Tree

Pass 5 combines elements of passes 0, 1, 3, and 4, producing a parse tree with 450 nodes.

```
Mon Feb 13 22:52:46 1989, File C: RCD3B06.TMP
|  Define Hierarchies [61.03 secs, 13 ops]
|  |  Start Over in the Workspace and ...
|  |  Created a new tree: (1 -> 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 24 20 21 22 23)  and ...
|  |  Created 24 solo nodes: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 25 26 27 28 [19.67 secs, 5 ops]
|  |  |  pause [2.81 secs]
|  |  |  Leave Prose II for another Application [0.77 secs]
|  |  |  pause [0.43 secs]
|  |  |  pause [2.53 secs]
|  |  |  Open a workspace file [13.13 secs]
|  |  Refocus [41.36 secs, 8 ops]
|  |  |  pause [6.70 secs]
|  |  |  pause [8.29 secs]
|  |  |  Move the Map Window [0.06 secs]
|  |  |  pause [0.33 secs]
|  |  |  Show the Map Window [0.05 secs]
|  |  |  pause [2.91 secs]
|  |  |  Zoom in the Map Window [1.65 secs]
|  |  |  pause [21.37 secs]
|  New Workspace [20.26 secs, 2 ops]
|  |  Start Over in the Workspace [0.05 secs, 1 op]
|  |  |  Start a new workspace [0.05 secs]
|  |  Pause [20.21 secs, 1 op]
|  |  |  pause [20.21 secs]
|  Exploration [70.69 secs, 12 ops]
|  |  Created 4 solo nodes: 1 2 3 4 [51.69 secs, 11 ops]
|  |  |  Create node 1 [0.11 secs]
|  |  |  Label node 1 [5.99 secs]
|  |  |  pause [5.77 secs]
|  |  |  Create node 2 [0.16 secs]
|  |  |  Label node 2 [7.42 secs]
|  |  |  pause [2.47 secs]
|  |  |  Create node 3 [0.11 secs]
|  |  |  Label node 3 [17.47 secs]
|  |  |  pause [2.19 secs]
|  |  |  Create node 4 [0.11 secs]
|  |  |  Label node 4 [9.89 secs]
|  |  Pause [19.00 secs, 1 op]
|  |  |  pause [19.00 secs]
|  Define Hierarchies [240.63 secs, 87 ops]
|  |  Created a new tree: (1 -> 2 3 4)  [3.63 secs, 5 ops]
|  |  |  Link nodes (1 -> 2) [0.77 secs]
|  |  |  pause [0.88 secs]
|  |  |  Link nodes (1 -> 3) [0.55 secs]
|  |  |  pause [0.71 secs]
|  |  |  Link nodes (1 -> 4) [0.72 secs]
|  |  Cleanup [6.75 secs, 3 ops]
|  |  |  pause [2.64 secs]
|  |  |  Tidy the workspace [0.05 secs]
|  |  |  pause [4.06 secs]
|  |  Grew an existing tree: (1 -> 5 6 8 9)  [72.29 secs, 25 ops]
|  |  |  Create node 5 [0.17 secs]
|  |  |  Label node 5 [5.49 secs]
|  |  |  pause [2.53 secs]
|  |  |  pause [5.88 secs]
|   |   |  Create node 6 [0.16 secs]
|   |   |  Label node 6 [9.23 secs]
|   |   |  pause [1.32 secs]
|   |   |  Link nodes (2 -> 5) [0.60 secs]
|   |   |  pause [0.82 secs]
|   |   |  Link nodes (2 -> 6) [0.77 secs]
|   |   |  pause [1.38 secs]
|   |   |  Create node 7 [0.11 secs]
|   |   |  Label node 7 --cancelled-- [9.44 secs]
|   |   |  pause [3.85 secs]
|   |   |  Delete node 7 [0.05 secs]
|   |   |  pause [2.09 secs]
|   |   |  Create node 8 [0.11 secs]
|   |   |  Label node 8 [10.60 secs]
|   |   |  pause [1.54 secs]
|   |   |  Create node 9 [0.11 secs]
|   |   |  Label node 9 [8.07 secs]
|   |   |  pause [5.99 secs]
|   |   |  Link nodes (3 -> 8) [0.71 secs]
|   |   |  pause [0.72 secs]
|   |   |  Link nodes (3 -> 9) [0.55 secs]
|   |   Cleanup [8.35 secs, 3 ops]
|   |   |  pause [2.47 secs]
|   |   |  Tidy the workspace [0.05 secs]
|   |   |  pause [5.83 secs]
|   |   Grew an existing tree: (1 -> 10)  [25.54 secs, 4 ops]
|   |   |  Create node 10 [0.05 secs]
|   |   |  Label node 10 [22.63 secs]
|   |   |  pause [1.92 secs]
|   |   |  Link nodes (3 -> 10) [0.94 secs]
|   |   Refocus [27.02 secs, 9 ops]
|   |   |  pause [3.02 secs]
|   |   |  Tidy the workspace [0.05 secs]
|   |   |  pause [3.41 secs]
|   |   |  Show the Map Window [0.05 secs]
|   |   |  pause [2.20 secs]
|   |   |  Zoom in the Map Window [0.82 secs]
|   |   |  pause [10.71 secs]
|   |   |  Zoom in the Map Window [1.27 secs]
|   |   |  pause [5.49 secs]
|   |   Grew an existing tree: (1 -> 11 12)  [25.92 secs, 9 ops]
|   |   |  Create node 11 [0.05 secs]
|   |   |  Label node 11 [5.94 secs]
|   |   |  pause [1.81 secs]
|   |   |  Link nodes (3 -> 11) [0.88 secs]
|   |   |  pause [1.59 secs]
|   |   |  Create node 12 [0.11 secs]
|   |   |  Label node 12 [13.13 secs]
|   |   |  pause [1.48 secs]
|   |   |  Link nodes (3 -> 12) [0.93 secs]
|   |   Refocus [10.55 secs, 5 ops]
|   |   |  pause [3.02 secs]
|   |   |  Tidy the workspace [0.06 secs]
|   |   |  pause [3.13 secs]
|   |   |  Zoom in the Map Window [1.32 secs]
|   |   |  pause [3.02 secs]
|   |   Grew an existing tree: (1 -> 14 13)  [30.21 secs, 9 ops]
|   |   |  Create node 13 [0.05 secs]
|   |   |  Label node 13 [6.93 secs]
|   |   |  pause [1.48 secs]
|   |   |  Create node 14 [0.11 secs]
|   |   |  Label node 14 [16.81 secs]
|   |   |  pause [1.97 secs]
|   |   |  Link nodes (4 -> 14) [1.10 secs]
|   |   |  pause [0.99 secs]
|   |   |  Link nodes (3 -> 13) [0.77 secs]
|   |   Cleanup [6.53 secs, 3 ops]
|   |   |  pause [3.07 secs]
|  |  |  Tidy the workspace [0.06 secs]
|  |  |  pause [3.40 secs]
|  |  Grew an existing tree: (1 -> 15 16)  [13.08 secs, 9 ops]
|  |  |  Create node 15 [0.11 secs]
|  |  |  Label node 15 [3.57 secs]
|  |  |  pause [1.71 secs]
|  |  |  Create node 16 [0.11 secs]
|  |  |  Label node 16 [3.57 secs]
|  |  |  pause [1.48 secs]
|  |  |  Link nodes (4 -> 15) [0.66 secs]
|  |  |  pause [0.88 secs]
|  |  |  Link nodes (4 -> 16) [0.99 secs]
|  |  Cleanup and Take Stock [10.76 secs, 3 ops]
|  |  |  pause [2.19 secs]
|  |  |  Tidy the workspace [0.06 secs]
|  |  |  pause [8.51 secs]
|  Exploration [45.37 secs, 8 ops]
|  |  Created 2 solo nodes: 17 18 [34.28 secs, 5 ops]
|  |  |  Create node 17 [0.17 secs]
|  |  |  Label node 17 [3.62 secs]
|  |  |  pause [1.98 secs]
|  |  |  Create node 18 [0.11 secs]
|  |  |  Label node 18 [28.40 secs]
|  |  Refocus [11.09 secs, 3 ops]
|  |  |  pause [3.07 secs]
|  |  |  Zoom in the Map Window [0.94 secs]
|  |  |  pause [7.08 secs]
|  Top Down Construction [153.35 secs, 43 ops]
|  |  Hooked existing nodes to a tree: (1 -> 18)  [1.43 secs, 1 op]
|  |  |  Link nodes (1 -> 18) [1.43 secs]
|  |  Cleanup [6.43 secs, 3 ops]
|  |  |  pause [2.58 secs]
|  |  |  Tidy the workspace [0.06 secs]
|  |  |  pause [3.79 secs]
|  |  Hooked existing nodes to a tree: (1 -> 17)  [0.93 secs, 1 op]
|  |  |  Link nodes (4 -> 17) [0.93 secs]
|  |  Cleanup [8.13 secs, 3 ops]
|  |  |  pause [1.98 secs]
|  |  |  Tidy the workspace [0.05 secs]
|  |  |  pause [6.10 secs]
|  |  Grew an existing tree: (1 -> 19 20)  [22.13 secs, 9 ops]
|  |  |  Create node 19 [0.11 secs]
|  |  |  Label node 19 [9.66 secs]
|  |  |  pause [1.60 secs]
|  |  |  Create node 20 [0.11 secs]
|  |  |  Label node 20 [5.16 secs]
|  |  |  pause [2.86 secs]
|  |  |  Link nodes (18 -> 19) [0.71 secs]
|  |  |  pause [0.88 secs]
|  |  |  Link nodes (18 -> 20) [1.04 secs]
|  |  Refocus [49.99 secs, 5 ops]
|  |  |  pause [2.69 secs]
|  |  |  Tidy the workspace [0.06 secs]
|  |  |  pause [15.82 secs]
|  |  |  Zoom in the Map Window [1.26 secs]
|  |  |  pause [30.16 secs]
|  |  Grew an existing tree: (1 -> 21)  [4.83 secs, 4 ops]
|  |  |  Create node 21 [0.10 secs]
|  |  |  Label node 21 [2.70 secs]
|  |  |  pause [1.37 secs]
|  |  |  Link nodes (18 -> 21) [0.66 secs]
|  |  Pause [23.62 secs, 1 op]
|  |  |  pause [23.62 secs]
|  |  Grew an existing tree: (1 -> 22)  [6.86 secs, 4 ops]
|  |  |  Create node 22 [0.11 secs]
|  |  |  Label node 22 [4.66 secs]
|  |  |  pause [1.38 secs]
|  |  |  Link nodes (4 -> 22) [0.71 secs]
|  |  Cleanup [6.32 secs, 3 ops]
|  |  |  pause [2.80 secs]
|  |  |  Tidy the workspace [0.06 secs]
|  |  |  pause [3.46 secs]
|  |  Grew an existing tree: (1 -> 23)  [6.26 secs, 4 ops]
|  |  |  Create node 23 [0.11 secs]
|  |  |  Label node 23 [4.01 secs]
|  |  |  pause [1.37 secs]
|  |  |  Link nodes (4 -> 23) [0.77 secs]
|  |  Refocus [16.42 secs, 5 ops]
|  |  |  pause [3.02 secs]
|  |  |  Tidy the workspace [0.06 secs]
|  |  |  pause [8.73 secs]
|  |  |  Zoom in the Map Window [0.55 secs]
|  |  |  pause [4.06 secs]
|  Exploration [14.45 secs, 5 ops]
|  |  Created a solo node: 24 [6.92 secs, 2 ops]
|  |  |  Create node 24 [0.06 secs]
|  |  |  Label node 24 [6.86 secs]
|  |  Refocus [7.53 secs, 3 ops]
|  |  |  pause [3.79 secs]
|  |  |  Zoom in the Map Window [0.88 secs]
|  |  |  pause [2.86 secs]
|  Top Down Construction [132.81 secs, 29 ops]
|  |  Hooked existing nodes to a tree: (1 -> 24)  [1.37 secs, 1 op]
|  |  |  Link nodes (1 -> 24) [1.37 secs]
|  |  Cleanup [8.40 secs, 3 ops]
|  |  |  pause [3.79 secs]
|  |  |  Tidy the workspace [0.06 secs]
|  |  |  pause [4.55 secs]
|  |  Grew an existing tree: (1 -> 25 26 27)  [28.95 secs, 14 ops]
|  |  |  Create node 25 [0.06 secs]
|  |  |  Label node 25 [5.05 secs]
|  |  |  pause [1.27 secs]
|  |  |  Link nodes (24 -> 25) [0.71 secs]
|  |  |  pause [0.66 secs]
|  |  |  Create node 26 [0.11 secs]
|  |  |  Label node 26 [7.08 secs]
|  |  |  pause [1.93 secs]
|  |  |  Create node 27 [0.16 secs]
|  |  |  Label node 27 [7.69 secs]
|  |  |  pause [1.32 secs]
|  |  |  Link nodes (24 -> 26) [0.93 secs]
|  |  |  pause [0.99 secs]
|  |  |  Link nodes (24 -> 27) [0.99 secs]
|  |  Refocus [94.09 secs, 11 ops]
|  |  |  pause [3.41 secs]
|  |  |  Tidy the workspace [0.05 secs]
|  |  |  pause [6.98 secs]
|  |  |  Zoom in the Map Window [0.99 secs]
|  |  |  pause [45.97 secs]
|  |  |  Zoom in the Map Window [1.04 secs]
|  |  |  pause [4.45 secs]
|  |  |  Zoom in the Map Window [1.37 secs]
|  |  |  pause [10.93 secs]
|  |  |  Zoom in the Map Window [0.83 secs]
|  |  |  pause [18.07 secs]
|  Tree Structure Revision [22.02 secs, 12 ops]
|  |  Broke existing links and ...
|  |  Deleted a node: 11 [13.89 secs, 9 ops]
|  |  |  Break link (3 -> 10) [0.05 secs]
|  |  |  pause [1.43 secs]
|  |  |  Break link (3 -> 11) [0.66 secs]
|  |  |  pause [5.05 secs]
|  |  |  Delete node 11 [6.70 secs]
|  |  |  |  Set Delete Mode On [0.11 secs, 1 op]
|  |  |  |  pause [0.77 secs, 1 op]
|  |  |  |  Delete node 11 [0.06 secs, 1 op]
|  |  |  |  pause [5.71 secs, 1 op]
|  |  |  |  Set Delete Mode Off [0.05 secs, 1 op]
|  |  Refocus [8.13 secs, 3 ops]
|  |  |  pause [3.13 secs]
|  |  |  Zoom in the Map Window [1.16 secs]
|  |  |  pause [3.84 secs]
|  Document Revision [1596.14 secs, 143 ops]
|  |  Hooked existing nodes to a tree: (1 -> 10)  [0.61 secs, 1 op]
|  |  |  Link nodes (4 -> 10) [0.61 secs]
|  |  Refocus [71.95 secs, 7 ops]
|  |  |  pause [2.80 secs]
|  |  |  Tidy the workspace [0.05 secs]
|  |  |  pause [34.55 secs]
|  |  |  Save the workspace [8.57 secs]
|  |  |  pause [11.20 secs]
|  |  |  Zoom in the Map Window [2.70 secs]
|  |  |  pause [12.08 secs]
|  |  Unproductive work [5.16 secs, 3 ops]
|  |  |  Create node 28 [0.11 secs]
|  |  |  pause [5.00 secs]
|  |  |  Delete node 28 [0.05 secs]
|  |  Refocus [27.25 secs, 6 ops]
|  |  |  pause [12.47 secs]
|  |  |  Zoom in the main window [2.42 secs]
|  |  |  pause [2.63 secs]
|  |  |  pause [4.56 secs]
|  |  |  pause [5.11 secs]
|  |  |  pause [0.06 secs]
|  |  Edited existing nodes and ...
|  |  Revised existing nodes [276.27 secs, 13 ops]
|  |  |  Edit node 5's contents [68.21 secs]
|  |  |  |  Leave Prose II for another Application [67.50 secs, 1 op]
|  |  |  |  Edit node 5's contents [0.71 secs, 1 op]
|  |  |  pause [4.73 secs]
|  |  |  pause [3.62 secs]
|  |  |  pause [4.78 secs]
|  |  |  Edit node 6's contents [120.89 secs]
|  |  |  |  Leave Prose II for another Application [119.96 secs, 1 op]
|  |  |  |  Edit node 6's contents [0.93 secs, 1 op]
|  |  |  pause [4.07 secs]
|  |  |  pause [2.69 secs]
|  |  |  pause [4.67 secs]
|  |  |  pause [0.05 secs]
|  |  |  Edit node 5's contents [62.56 secs]
|  |  |  |  Leave Prose II for another Application [61.68 secs, 1 op]
|  |  |  |  Edit node 5's contents [0.88 secs, 1 op]
|  |  Pause [24.44 secs, 4 ops]
|  |  |  pause [15.11 secs]
|  |  |  pause [4.44 secs]
|  |  |  pause [4.84 secs]
|  |  |  pause [0.05 secs]
|  |  Edited existing nodes [194.55 secs, 12 ops]
|  |  |  Edit node 8's contents [28.67 secs]
|  |  |  |  Leave Prose II for another Application [27.96 secs, 1 op]
|  |  |  |  Edit node 8's contents [0.71 secs, 1 op]
|  |  |  pause [4.51 secs]
|  |  |  pause [4.06 secs]
|  |  |  pause [4.84 secs]
|  |  |  Edit node 9's contents [54.32 secs]
|  |  |  |  Leave Prose II for another Application [53.49 secs, 1 op]
|  |  |  |  Edit node 9's contents [0.83 secs, 1 op]
|  |  |  pause [5.49 secs]
|  |  |  pause [3.35 secs]
|  |  |  pause [4.56 secs]
|  |  |  Edit node 12's contents [84.75 secs]
|  |  |  |  Leave Prose II for another Application [84.03 secs, 1 op]
|  |  |  |  Edit node 12's contents [0.72 secs, 1 op]
|  |  Pause [18.45 secs, 3 ops]
|  |  |  pause [8.24 secs]
|  |  |  pause [5.00 secs]
|  |  |  pause [5.21 secs]
|  |  Edited existing nodes [148.30 secs, 12 ops]
|  |  |  Edit node 13's contents [34.39 secs]
|  |  |  |  Leave Prose II for another Application [33.67 secs, 1 op]
|  |  |  |  Edit node 13's contents [0.72 secs, 1 op]
|  |  |  pause [4.50 secs]
|  |  |  pause [3.74 secs]
|  |  |  pause [5.27 secs]
|  |  |  Edit node 10's contents [56.79 secs]
|  |  |  |  Leave Prose II for another Application [56.08 secs, 1 op]
|  |  |  |  Edit node 10's contents [0.71 secs, 1 op]
|  |  |  pause [4.62 secs]
|  |  |  pause [4.83 secs]
|  |  |  pause [5.00 secs]
|  |  |  Edit node 14's contents [29.16 secs]
|  |  |  |  Leave Prose II for another Application [28.12 secs, 1 op]
|  |  |  |  Edit node 14's contents [1.04 secs, 1 op]
|  |  Pause [20.60 secs, 3 ops]
|  |  |  pause [8.79 secs]
|  |  |  pause [6.48 secs]
|  |  |  pause [5.33 secs]
|  |  Edited existing nodes [55.69 secs, 2 ops]
|  |  |  Edit node 15's contents [55.69 secs]
|  |  |  |  Leave Prose II for another Application [54.93 secs, 1 op]
|  |  |  |  Edit node 15's contents [0.76 secs, 1 op]
|  |  Take Stock [21.86 secs, 6 ops]
|  |  |  pause [6.21 secs]
|  |  |  Save the workspace [3.79 secs]
|  |  |  pause [2.42 secs]
|  |  |  pause [0.88 secs]
|  |  |  pause [3.51 secs]
|  |  |  pause [5.05 secs]
|  |  Edited existing nodes [24.39 secs, 2 ops]
|  |  |  Edit node 16's contents [24.39 secs]
|  |  |  |  Leave Prose II for another Application [23.18 secs, 1 op]
|  |  |  |  Edit node 16's contents [1.21 secs, 1 op]
|  |  Pause [15.05 secs, 3 ops]
|  |  |  pause [6.92 secs]
|  |  |  pause [3.02 secs]
|  |  |  pause [5.11 secs]
|  |  Edited existing nodes [182.30 secs, 12 ops]
|  |  |  Edit node 17's contents [51.80 secs]
|  |  |  |  Leave Prose II for another Application [50.92 secs, 1 op]
|  |  |  |  Edit node 17's contents [0.88 secs, 1 op]
|  |  |  pause [4.72 secs]
|  |  |  pause [2.75 secs]
|  |  |  pause [5.32 secs]
|  |  |  Edit node 22's contents [41.20 secs]
|  |  |  |  Leave Prose II for another Application [40.43 secs, 1 op]
|  |  |  |  Edit node 22's contents [0.77 secs, 1 op]
|  |  |  pause [5.05 secs]
|  |  |  pause [3.46 secs]
|  |  |  pause [5.33 secs]
|  |  |  Edit node 23's contents [62.67 secs]
|  |  |  |  Leave Prose II for another Application [61.84 secs, 1 op]
|  |  |  |  Edit node 23's contents [0.83 secs, 1 op]
|  |  Pause [21.91 secs, 3 ops]
|  |  |  pause [4.83 secs]
|  |  |  pause [11.87 secs]
|  |  |  pause [5.21 secs]
|  |  Edited existing nodes [130.89 secs, 8 ops]
|  |  |  Edit node 19's contents [47.19 secs]
|  |  |  |  Leave Prose II for another Application [46.47 secs, 1 op]
|  |  |  |  Edit node 19's contents [0.72 secs, 1 op]
|  |  |  pause [4.00 secs]
|  |  |  pause [3.03 secs]
|  |  |  pause [4.99 secs]
|  |  |  pause [0.06 secs]
|  |  |  Edit node 20's contents [71.62 secs]
|  |  |  |  Leave Prose II for another Application [70.47 secs, 1 op]
|  |  |  |  Edit node 20's contents [1.15 secs, 1 op]
|  |  Pause [15.27 secs, 3 ops]
|  |  |  pause [6.48 secs]
|  |  |  pause [3.30 secs]
|  |  |  pause [5.49 secs]
|  |  Edited existing nodes [246.40 secs, 19 ops]
|  |  |  Edit node 21's contents [84.48 secs]
|  |  |  |  Leave Prose II for another Application [83.76 secs, 1 op]
|  |  |  |  Edit node 21's contents [0.72 secs, 1 op]
|  |  |  pause [4.72 secs]
|  |  |  pause [2.64 secs]
|  |  |  pause [5.21 secs]
|  |  |  Edit node 25's contents [29.11 secs]
|  |  |  |  Leave Prose II for another Application [28.18 secs, 1 op]
|  |  |  |  Edit node 25's contents [0.93 secs, 1 op]
|  |  |  pause [4.45 secs]
|  |  |  pause [3.46 secs]
|  |  |  pause [5.11 secs]
|  |  |  Edit node 26's contents [39.49 secs]
|  |  |  |  Leave Prose II for another Application [38.56 secs, 1 op]
|  |  |  |  Edit node 26's contents [0.93 secs, 1 op]
|  |  |  pause [3.74 secs]
|  |  |  pause [2.74 secs]
|  |  |  pause [5.28 secs]
|  |  |  pause [0.05 secs]
|  |  |  pause [0.06 secs]
|  |  |  Edit node 27's contents [55.86 secs]
|  |  |  |  Leave Prose II for another Application [55.03 secs, 1 op]
|  |  |  |  Edit node 27's contents [0.83 secs, 1 op]
|  |  Refocus [94.80 secs, 21 ops]
|  |  |  pause [5.43 secs]
|  |  |  Move the Map Window [0.06 secs]
|  |  |  pause [0.33 secs]
|  |  |  Show the Map Window [0.05 secs]
|  |  |  pause [2.26 secs]
|  |  |  Zoom in the Map Window [2.14 secs]
|  |  |  pause [5.82 secs]
|  |  |  Zoom in the Map Window [1.26 secs]
|  |  |  pause [3.96 secs]
|  |  |  Zoom in the Map Window [0.82 secs]
|  |  |  pause [7.86 secs]
|  |  |  Save the workspace [4.12 secs]
|  |  |  pause [6.64 secs]
|  |  |  Move the Outline Window [0.06 secs]
|  |  |  pause [0.44 secs]
|  |  |  Show the Outline Window [0.05 secs]
|  |  |  pause [11.59 secs]
|  |  |  Save the workspace [16.92 secs]
|  |  |  pause [4.12 secs]
|  |  |  Save the workspace [17.19 secs]
|  |  |  pause [3.68 secs]
```

B.7.2 Summary File for Session S18R0108

```
"S18R0108.P4"
    450, Number of nodes in the parse tree
    153, Number of commands

     28, Number of creates
      3, Number of deletes
      0, Number of copies
     26, Number of links
      2, Number of break links
      0, Number of moves
     14, Number of tidies
      1, Number of cancelled operations

      1, Number of opens
      5, Number of saves
      0, Number of nodes in last saved tree
      0, Number of nodes with no offspring
      0, Maximum depth of saved tree
     10, Cumulative X vector of creates
      0, Cumulative Y vector of creates
   0.81, Stage Index

     20, Number of edit node operations
      0, Number of help requests
      0, Number of long pauses
      0, Number of user comments
      1, Number of times subject left Prose II
     30, Number of constructive episodes
     21, Number of housekeeping episodes
     10, Number of phases

  276.3, Longest constructive episode, in seconds
   24.4, Longest meta-housekeeping episode, in seconds
   94.8, Longest housekeeping episode, in seconds

 1103.2, Total seconds spent editing nodes
    0.0, Total seconds spent in help
    0.0, Total seconds spent in long pauses
  891.0, Total seconds in all pauses
    0.0, Total seconds spent in comments
    0.8, Total seconds spent outside Prose II

 1634.5, Total seconds in constructive episodes
  178.6, Total seconds in meta-housekeeping episodes
  543.7, Total seconds in housekeeping episodes

 2356.8, Total seconds in this session
```

APPENDIX C. PARSER COMMAND LINE PARAMETERS

Each of the passes in the parser uses one or more command line parameters to control the work done in that pass.

Pass 0 Command Line Parameters: Pass 0 uses only one command line parameter, to control whether debugging information is to be generated in its output file.

Option	Description
-d	Turn on tracing and generate the output information used for debugging this pass. A previous try at parsing a given input file failed on this pass; the debugging information should help a human in finding problems in this pass.

Pass 1 Command Line Parameters: Pass 1 uses two command line parameters to control its operation.

Option	Description
-c	Send only the ID of the command to the output file, rather than the entire 16-tuple of information. This can be used to generate a condensed file containing all the commands issued.
-d	Turn on tracing and generate the output information used for debugging this pass. A previous try at parsing a given input file failed on this pass; the debugging information should help a human in finding problems in this pass.

Pass 2 Command Line Parameters: As with Pass 0, Pass 2 allows one optional command line parameter: to control whether debugging information should be generated because a parsing bug was found.

Option	Description
-d	Turn on tracing and generate the output information used for debugging this pass. A previous try at parsing a given input file failed on this pass; the debugging information should help a human in finding problems in this pass.

Pass 3 Command Line Parameters: Pass 3 of the parser accepts eight optional command-line flags.

Option	Description
-d	Turn on the extra tracing and information used for debugging this pass.
-f	Create a new file and write the current workspace to it in indented format every time a successful SaveWorkspace command is encountered. If multiple saves are done, these files are numbered with DOS file extensions of .1, .2, etc. This allows seeing such things as how the trees were evolving during the trace, whether trees were deep or wide or both, and how a node's position in a tree correlated with when it was created.
-l	Write a unique letter, instead of the four-digit internal parser symbol, when producing each record in the output file. This helps when reading this file as input to a spreadsheet and graphing the distribution of the types of phases.

-m	Save the file structure in spreadsheet format. This format was used to compare the order of node creation against the node's position in a pre-order tree traversal. In the following example, the first node in the tree (the root) was the seventeenth node created.

```
pre-order
position    node ID
---------   -------
    1         17
    2          1
    3          2
    4          3
    5          6
    6          4
    7          7
    8         15
    9         16
   10         12
```

-s	Output the sequence of commands that compose each episode.
-t	Output only the operators and times, rather than the full n-tuples.
-2	Output Pass 2 symbols only.
-3	Output Pass 3 symbols only.

Pass 4 Command Line Parameters: Pass 4 of the parser accepts two optional command-line flags.

Option	**Description**
-d	Turn on the extra tracing and information used for debugging this pass.
-l	When producing each record in the output file, output a unique letter, instead of the four-digit internal parser symbol. This helps when reading this file as input to a spreadsheet and graphing the distribution of the types of phases.

Summary Pass Command Line Parameters: The command line parameters for the summary pass generally controlled the format of the output. *Prose II* itself allowed tree structures to be input in several different formats; the summary pass could generate the parse tree in these same formats.

Option	**Description**
filename	This summary pass reads from the preceding intermediate files, not from **stdin**. This specifies the filename used by each of the intermediate files. It assumes the intermediate files have DOS file extensions of `.P0`, `.P1`, ..., `.P4`.
-d	Turn on the extra tracing and information used for debugging this pass.
-i	Produce an indented parse tree; that is, write the parse tree using the `.IND` file format.
-r	Produce a Ready!-compatible parse tree; that is, write the parse tree using the `.RDY` file format.
-s	Produce a Script-compatible indented parse tree, with Script "&lbrk." tags and vertical bars, |, to illustrate indentation.
-t	Write text headers in the summary file. This makes the summary information for a single protocol record easily readable by a human. If this option is not specified, the output is written to **stdout** in ASCII spreadsheet format.

REFERENCES

Aho, A.V. and J.D. Ullman. *Principles of Compiler Design.* Addison-Wesley, Reading, MA, 1977.

Ausubel, D.P. *The Psychology of Meaningful Verbal Learning.* Grune and Stratton, New York, NY, 1963.

Bates, M. The theory and practice of augmented transition network grammars. In *Natural Language Communication with Computers*, L. Bolc, Ed., Springer Verlag, New York, NY, 1978, 191-260.

Beard, D.V. and J.Q. Walker II. Navigational Techniques to Improve the Display of Large Two-Dimensional Spaces. *Behaviour & Information Technology* **9**, 6 (1990), 451-466. Also available as Technical Report TR89-042, Department of Computer Science, University of North Carolina at Chapel Hill, October 1989.

Bhaskar, R. and H.A. Simon. Problem solving in semantically rich domains: An example from engineering thermodynamics. *Cognitive Science* **1** (1977), 193-215.

Bolter, J.D. and M. Joyce. Hypertext and Creative Writing. *Hypertext '87 Papers.* Technical Report TR88-013, Department of Computer Science, University of North Carolina at Chapel Hill, (March 1988), 41-50.

Britton, B.K., B.J.F. Meyer, M.H. Hodge, and S.M. Glynn. Effects of the organization of text on memory: tests of retrieval and response criterion hypotheses. *Journal of Experimental Psychology: Human Learning and Memory* **6** (1980), 620-629.

Brooks, Jr., F.P. *The Mythical Man-Month: Essays on Software Engineering.* Addison-Wesley, Reading, MA, 1975.

Brooks, Jr., F.P. No Silver Bullet—Essence and Accidents of Software Engineering. *Computer* **20**, 4 (April 1987), 10-19.

Bryan, W.L. and N. Harter. Studies in the physiology and psychology of the telegraphic language. *Psychological Review* **4** (1898), 27-53.

Burkhart, H. and J. Nievergelt. *Structure-oriented editors.* Technical Report 38, Eidgenossische Technische Hochschule Zurich, Institute fur Informatik, Zurich, Switzerland (May 1980).

Business Week. Giving Design Engineers More Time to Design. (February 3, 1986), 63.

Card, S.K., T.P. Moran, and A. Newell. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1983.

Chomsky, N. *Aspects of the Theory of Syntax.* MIT Press, Cambridge, MA, 1975.

Collins, A.M. and E.F. Loftus. A spreading-activation theory of semantic processing. *Psychological Review* **82** (1975), 407-428.

Collins, A.M. and M.R. Quillian. How to make a language user. In *Organization of Memory*, E. Tulving and W. Donaldson, Eds., Academic Press, New York, NY, 1972.

Cox, B.J. *Object-oriented programming*. Addison-Wesley, Reading, MA, 1986.

Durding, B.M., C.A. Becker, and J.D. Gould. Data organization. *Human Factors* **19**, 1 (1977), 1-14.

Ericsson, K.A. and H.A. Simon. Verbal reports as data. *Psychological Review* **87**, 3 (May 1980), 215-250.

Ericsson, K.A. and H.A. Simon. *Protocol Analysis*. MIT Press, Cambridge, MA, 1984.

Fersko-Weiss, Henry. 3-D Reading with the Hypertext Edge. *PC Magazine* **10**, 10 (May 28, 1991), 241-282.

Flower, L.S. and J.R. Hayes. Images, plans, and prose: The representation of meaning in writing. *Written Communication* **1**, 1 (January 1984), 120-160.

Foley, J.D., V.L. Wallace, and P. Chan. The Human Factors of Computer Graphics Interaction Techniques. *IEEE Computer Graphics and Applications*, (November 1984), 13-48.

Fountain, A.J. and M.A. Norman. Modelling user behaviour with formal grammar. In *People and Computers: Designing the Interface*, P. Johnson and S. Cook, Eds., British Computer Society, (Cambridge, September 17-20, 1985), 3-12.

Greenberg, S. and I.H. Witten. How Users Repeat Their Actions on Computers: Principles for Design of History Mechanisms. In *Proceedings of the 1988 Conference on Computer-Human Interaction*, (Washington, DC, 1988), 171-178.

Greeno, J.G. Process of understanding in problem-solving. In *Cognitive Theory (vol. 2)*, N.J. Castellan, D.B. Pisoni, andP G.R. Potts, Eds., Lawrence Erlbaum Associates, Hillsdale, NJ, 1977.

Halasz, F.G., T.P. Moran, and R.H. Trigg. NoteCards in a Nutshell. In *Proceedings of the 1987 ACM Conference of Human Factors in Computer Systems*, (Toronto, Ontario, April 5-9, 1987), 45-52.

Halasz, F.G. Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems. *Hypertext '87 Papers*. Technical Report TR88-013, Department of Computer Science, University of North Carolina at Chapel Hill, (March 1988), 345-365.

Hayes, J.R. and L.S. Flower. Identifying the organization of the writing process. In *Cognitive Processes in Writing*, L.W. Gregg and E.R. Steinberg, Eds., Lawrence Erlbaum Associates, Hillsdale, NJ, 1980, 3-30.

Hayes, J.R. and L.S. Flower. Writing research and the writer. *American Psychologist* **41**, 10 (October 1986), 1106-1113.

Henry, L.K. The Role of Insight in The Analytic Thinking of Adolescents. *Studies in Education, University of Iowa Studies* **9** (1934), 65-102.

Johnson, S.C. *YACC—yet another compiler compiler*. Computing Science Technical Report No. 32, Bell Laboratories, Murray Hill, NJ, 1975.

Kieras, D. E. Initial mention as a signal to thematic content in technical passages. *Memory and Cognition* **8**, 4 (1980), 345-353.

Kieras, D. and P.G. Polson. An Approach to the Formal Analysis of User Complexity. *International Journal of Man-Machine Studies* **22** (1985), 365-394.

Kintsch, W. *The Representation of Meaning in Memory.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1974.

Kintsch, W. & Keenan, J.M. Reading rate and retention as a function of the number of propositions in base structure of sentences. *Cognitive Psychology* **5** (1973), 257-274.

Kintsch, W. and T.A. van Dijk. Toward a model of text comprehension and production. *Psychological Review* **85** (1978), 363-394.

Lachman, J.L. and R. Lachman. Comprehension and cognition: a state of the art of inquiry. In *Levels of Processing in Human Memory*, L.S. Cermak and F.I.M. Cnaik, Eds., Lawrence Erlbaum Associates, Hillsdale, NJ: 1979a, 183-210.

Landauer, T.K. Memory without organization: Some properties of a model with random storage and undirected retrieval. Paper presented at the 13th Meeting of the Psychometric Society, St. Louis, MO, November, 1972.

Lansman, M., J.B. Smith and I. Weber. *Using Computer-Generated Protocols to Study Writers' Planning Strategies.* Technical Report TR90-033, Department of Computer Science, University of North Carolina at Chapel Hill, September 1990.

Lesk, M.E. *LEX—a lexical analyzer generator.* Computing Science Technical Report No. 39, Bell Laboratories, Murray Hill, NJ, 1975.

Mackay, W.E. Video: Data for Studying Human-Computer Interaction. *Proceedings of the 1988 Conference on Computer-Human Interaction,* (Washington, DC, 1988), 133-137.

Mandler, J.M. Categorical and schematic organization in memory. In *Memory Organization and Structure*, C.R. Puff, Ed., Academic Press, New York, NY, 1979.

Matsuhashi, A. Pausing and Planning: The tempo of written discourse production. *Research in the Teaching of English* **15**, 2 (1981), 113-134.

Mead, C. and L. Conway. *Introduction to VLSI Systems.* Addison-Wesley, Reading, MA, 1980.

Meyer, B.J.F. *The Organization of Prose and its Effects on Memory.* North Holland Publishing, Amsterdam, 1975.

Meyer, B.J.F., D.M. Brandt, and G.J. Bluth. Use of top-level structure in text: key for reading comprehension of ninth grade students. *Reading Research Quarterly* **1** (1980), 72-103.

Microsoft Windows Users Guide. Microsoft Press: Redmond, Washington, 1987, document number 050050051-200-R01-0887.

Miller, G.A. The magical number seven plus or minus two: Some limits on our capacity for processing information. *Psychological Review* **63** (1956), 81-97.

Mills, C.C. *Drawing USE Transition Diagrams using TDE*, Computing Science Division, University of California, Berkeley, CA, 1984.

Neal, A.S. and R.M. Simons. Playback: A method for evaluating the usability of software and its documentation. *IBM Systems Journal* **23**, 1 (1984), 82-96.

Nisbett, R.E. and T.D. Wilson. Telling More than We Can Know: Verbal Reports on Mental Processes. *Psychological Review* **84**, 3 (May 1977), 231-259.

Palmer, S.E. Hierarchical structure in perceptual representation. *Cognitive Psychology* 9 (1977), 441-475.

Palmer, S.E. Fundamental Aspects of Cognitive Representation. In *Cognition and Categorization*, E. Rosch and B.B. Lloyd, Eds., Lawrence Erlbaum Associates, Hillsdale, NJ, 1978.

Posner, M.I. and R.E. Warren. Traces, concepts, and conscious constructions. In *Coding Processes in Human Memory*, A.W. Melton and E. Martin, Eds., Winston, Washington, DC, 1972.

Quillian, M.R. Semantic memory. In *Semantic Information Processing*, M. Minsky, Ed., MIT Press, Cambridge, MA, 1968.

Reisner, P. Formal Grammar and Human Factors Design of an Interactive Graphics System. *IEEE Transactions on Software Engineering* **SE-7**, 2 (March 1981), 229-240.

Reisner, P. Further Developments Toward Using Formal Grammar as a Design Tool. In *Proceedings of the Gaithersburg conference on Human Factors in Computer Systems*, National Bureau of Standards, (Gaithersburg, MD, March 15-17, 1982), 304-308.

Roberts, M. Brainstorming by computer. *Psychology Today* (July/August 1989), 51.

Schank, R.C. and R.P. Abelson. *Scripts, Plans, Goals, and Understanding*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1977.

Schneiderman, B. Direct Manipulation: A Step Beyond Programming Languages. *IEEE Computer* **16**, 8 (August 1983), 57-69.

Schwarz, M.N.K. and A. Flammer. Text structure and title effects on comprehension and recall. *Journal of Verbal Learning and Verbal Behavior* **20** (1981), 61-66.

Shepard, R.N. The Mental Image. *American Psychologist* **33** (February 1978), 125-137.

Shepard, R.N. Externalization of Mental Images and The Act of Creation. In *Visual Learning, Thinking, and Communication*, B.S. Randhawa and W.E. Coffman, Eds., Academic Press, New York, NY, 1978, 133-189.

Simon, H.A. *Models of Thought*. Yale University Press, New Haven, CT, 1979.

Smith, J.B. and M. Lansman. *A Cognitive Basis for a Computer Writing Environment*. Technical Report TR87-032, Department of Computer Science, University of North Carolina at Chapel Hill, June 1988.

Smith, J.B., M.C. Rooks, and G.J. Ferguson. *A Cognitive Grammar for Writing: Version 1.0*. Technical Report TR89-011, Department of Computer Science, University of North Carolina at Chapel Hill, April 1989.

Smith, J.B. and C.F. Smith. *A Strategic Method for Writing*. Technical Report TR87-024, Department of Computer Science, University of North Carolina at Chapel Hill, August 1987.

Smith, J.B., S.F. Weiss, and G.J. Ferguson. *A Hypertext Writing Environment and Its Cognitive Basis*. Technical Report TR87-033, Department of Computer Science, University of North Carolina at Chapel Hill, October 1987. Also printed in *Hypertext '87 Papers*, Technical Report TR88-013, Department of Computer Science, University of North Carolina at Chapel Hill, (March 1988), 195-214.

Smith, J.B., S.F. Weiss, M. Lansman, J.D. Bolter, and D.V. Beard. *An Experimental Study of Writers' Cognitive Strategies Using Advanced Computer Tools*. Project description submitted to the U.S. Army Research Institute for the Behavioral and Social Sciences. Department of Computer Science, University of North Carolina at Chapel Hill, September 1985.

Swarts, H., L.S. Flower, and J.R. Hayes. Designing protocol studies of the writing process: An Introduction. In *New Directions in Composition Research*, R. Beach and L.S. Bridwell, Eds., Guilford Press, New York, NY, 1984, 53-71.

Tulving, E. Episodic and semantic memory. In *Organization of Memory*, E. Tulving and W. Donaldson, Eds., Academic Press, New York, NY, 1972.

Voss, J.F., S.W. Tyler, and G.L. Bisanz. Prose comprehension and memory. In *Handbook of Research Methods in Human Memory and Cognition*, C.R. Puff, Ed., Academic Press, New York, NY, 1982.

Walker II, J.Q. A Node-Positioning Algorithm for General Trees. *Software—Practice and Experience* **20**, 7 (July 1990), 685-705.

Walker II, J.Q. Positioning Nodes For General Trees. *The C Users Journal* **9**, 2 (February 1991), 47-62.

Waterman, D.A. and A. Newell. Protocol analysis as a task for artificial intelligence. *Artificial Intelligence* **2** (1971), 285-318.

Waterman, D.A. and A. Newell. PAS-II: An interactive task-free version of an automatic protocol analysis system. In *Proceedings of the Third IJCAI*, Stanford Research Institute, Menlo Park, CA, 1973, 431-445.

West, L.H.T., P.J. Fensham, and J.E. Garrard. Describing the cognitive structures of learners following instruction in chemistry. In *Cognitive Structure and Conceptual Change*, L.H.T. West and A.L. Pines, Eds., Academic Press, Orlando, FL, 1985.

White, R.T. Interview protocols and dimensions of cognitive structure. In *Cognitive Structure and Conceptual Change*, L.H.T. West and A.L. Pines, Eds., Academic Press, Orlando, FL, 1985.

Williams, J.P., M.B. Taylor, and S. Ganger. Text variations at the level of the individual sentence and the comprehension of simple expository paragraphs. *Journal of Educational Psychology* **73**, 6 (1981), 851-865.

Woods, W.A. Transition Network Grammars for Natural Language Analysis. *Communications of the ACM* **13** (1970), 591-606.

Woods, W.A. Cascaded ATN grammars. *American Journal of Computational Linguistics* **6**, 1 (1980), 1-12.

TRADEMARKS

IBM® and IBM PC® are registered trademarks and IBM PC-AT™ is a trademark of International Business Machines Corporation.

Macintosh® is a registered trademark of Apple Computer Company.

Microsoft® and MS-DOS® are registered trademarks of Microsoft Corporation.

Ready!™ and ThinkTank™ are trademarks of Living Videotext, Inc.

Smalltalk-80™ is a trademark of the Xerox Corporation.

Sun Workstation™ is a trademark of Sun Microsystems, Inc.

Storyspace™ is a trademark of Jay David Bolter.

UNIX® is a registered trademark of AT&T.

WordPerfect™ is a trademark of WordPerfect Corporation.