COGNITIVE SCIENCE Vol 24 (1) 2000, pp. 1–52 ISSN 0364-0213

Comprehension-Based Skill Acquisition

STEPHANIE M. DOANE
Mississippi State University

YOUNG WOO SOHN
University of Connecticut

DANIELLE S. MCNAMARA
Old Dominion University

DAVID ADAMS
Washington University in St. Louis

We present a comprehension-based computational model of UNIX user skill acquisition and performance in a training context (UNICOM). The work extends a comprehension-based theory of planning to account for skill acquisition and learning. Individual models of 22 UNIX users were constructed and used to simulate user performance on successive command production problems in a training context. Comparisons of model and the human empirical data result in a high degree of agreement, validating the ability of UNICOM to predict user response to training.

I. INTRODUCTION

There are numerous theories of how we acquire knowledge and skill by engaging in problem solving episodes, and several have been implemented as computational models. For example, case-based planning (e.g., Hammond, 1989) assumes that we acquire knowledge by storing problem solving episodes in memory which are, in general terms, specific plans that may be used to solve future problems. In contrast, search-based models like SOAR learn by chunking the results of the search process used to solve a problem (e.g., Rosenbloom, Laird, Newell, & McCarl, 1991). Anderson's ACT-R model of cognition learning is governed by the use of analogical interpretive problem solving

Direct all correspondence to: Stephanie Doane, Department of Psychology, Mississippi State University, PO Box 6161, Mississippi State, MS 39762; E-Mail: sdoane@ra.msstate.edu.

processes (Anderson, 1993, 1998). Alternatively, our theoretical premise is that knowledge and skill acquisition during problem solving episodes arise from comprehension-based mechanisms identical to those used to understand a list of words, narrative prose, and algebraic word problems. The theoretical foundation of our premise rests on Kintsch's (1988) construction-integration theory of comprehension.

Specifically, Kintsch's (1988, 1998) construction-integration theory presumes that low-level associations between contextual information (e.g., task instructions) and long-term memory are constructed and used to constrain knowledge activation via a constraint-based integration process. The resulting pattern of context-sensitive knowledge activation is referred to as a situation model and represents the current state of comprehension.

Kintsch's theory has been used to explain a wide variety of behavioral phenomena, including narrative story comprehension (Kintsch, 1988), algebra story problem comprehension (Kintsch, 1988), the solution of simple computing tasks (Mannes & Kintsch, 1991), and completing the Tower of Hanoi task (Schmalhofer & Tschaitschian, 1993). This approach has also proved fruitful for understanding human-computer interaction skills (e.g., Doane, McNamara, Kintsch, Polson, & Clawson, 1992; Kitajima & Polson, 1995; Mannes & Doane, 1991). The breadth of application suggests that the comprehension processes described in Kintsch's model play a central role in many tasks, and as such may be considered a general architecture of cognition (Newell, 1990).

Although the studies described above provide important support for the centrality of comprehension in cognition, they do not directly test the ability of this approach to predict human performance in a learning environment. Such a test is necessary to support the centrality of comprehension-based processes in cognition. The present research directly tests the predictive validity of the claim that comprehension-based processes play a central role in cognition and learning (e.g., Doane, Sohn, Adams, & McNamara, 1994; Gernsbacher, 1990; Kintsch, 1988; Schmalhofer & Tschaitschian, 1993; van Dijk & Kintsch, 1983). Proponents of this view have proposed detailed cognitive models of comprehension (e.g., Kintsch, 1988), and provided evidence for the importance of comprehension for understanding cognition in general (e.g., Gernsbacher, 1990).

The main goal of the present research is to test the ability of a comprehension-linked model to predict computer user performance in the context of a training task. Although other architectures can simulate comprehension, their processes are not comprehension-linked. The present model is not simply another architecture that can do comprehension but rather is uniquely structured for that purpose. Thus use of this model allows us to test whether comprehension-linked processes can explain and predict human performance in a learning environment.

Specifically, we evaluate whether UNICOM, a construction-integration model containing knowledge of UNIX commands, can predict the impact instructions will have on user command production performance (Sohn & Doane, 1997). This was accomplished by defining comprehension-based learning mechanisms in UNICOM, and then simulating performance of 22 actual users on 21 command production tasks in an interactive training environment. Human and model performance data were compared to determine UNICOM's predictive validity.

In the following we provide a brief summary of our rationale for choosing UNIX command production as a task domain, and the comprehension-based approach as our framework. We then describe UNICOM, and our modeling experiments testing the ability of UNICOM to predict response to training instructions.

UNIX Task Domain

Rationale. The UNIX domain was chosen for several reasons. First, producing complex UNIX commands requires use of a particular set of sequence-dependent actions, which is an appropriate context for the study of skill acquisition (e.g., Ericsson & Oliver, 1988). Second, UNIX expertise can be explicitly measured, which allows us to classify users into expertise groups for analyses (e.g., Doane, Pellegrino, & Klatzky, 1990). Third, we possess a rich set of empirical data on UNIX use that serves a critical role in the evaluation of UNICOM's predictive validity.

A Brief Introduction to UNIX. A fundamental component of the UNIX user interface is that composite commands can be created by concatenating simple commands with the use of advanced features that redirect command input and output. The composite commands act as small programs that provide unique, user-defined functions. For example, the single command "ls" will print the filenames in the current directory on the screen. The single command "lpr" used in combination with a filename (i.e., lpr file) will display the contents of the file on the line printer. If a user wishes to print the filenames of the current directory on the line printer, this could be accomplished through the use of redirection symbols that concatenate commands into composite functions. In the present example, this can be accomplished in one of two major ways.

The first method requires use of redirection symbols that store intermediate steps in files. For example, the ">" symbol redirects the output of a command to a file. Thus, the composite command "ls>file" would output the names of the files in the current directory into "file." If this were followed by a subsequent command "lpr file," then the file names would be printed on the line printer. The second method requires fewer keystrokes, and as such is more streamlined. The use of pipes "|" allows commands to be combined to produce the composite command "ls|lpr." In this example, the output from the "ls" command flows directly into the "lpr" command. Thus, one can think of a pipe as a plumbing pipe, where information flowing out from one command is redirected into the next.

UNIX User Performance. The symbols that enable input/output redirection are fundamental design features of UNIX, and these features are taught in elementary computer science courses, but Doane, Pellegrino, and Klatzky (1990) demonstrate that these features can only be used reliably after extensive experience (e.g., on the average, 5 years of experience with UNIX). Doane et al. provided UNIX users at varied levels of expertise with textual descriptions to produce single, multiple, and composite UNIX commands. Their findings suggested that novice and intermediate users have knowledge of the elements of the system; that is, they can successfully produce the single and multiple commands that make up a composite. They cannot, however, put these elements together using pipes and/or other redirection symbols to produce the composite commands. This was true even though users were allowed to use any legal means to accomplish the specified composite goal (e.g., "ls>temp; lpr temp" for "ls|lpr").

Although the Doane et al. (1990) findings demonstrated systematic performance deficits, they did not clarify their cause. To determine the cognitive loci of UNIX user performance deficits, Doane, Kintsch, and Polson (1990); Doane, Mannes, Kintsch, & Polson, 1992) used the comprehension-based framework to build a knowledge-based computational model of UNIX command production skill called UNICOM (which stands for "UNIX COMmands"). Before detailing the UNICOM model we need to introduce the foundations of the comprehension-based approach.

Construction-Integration Theory

The construction-integration model (Kintsch, 1988) was initially developed to explain certain phenomena of text comprehension, such as word sense disambiguation. The model describes how we use contextual information to assign a single meaning to words that have multiple meanings. For example, the appropriate assignment of meaning for the word "bank" is different in the context of conversations about paychecks (money "bank") and about swimming (river "bank"). In Kintsch's view, this can be explained by representing memory as an associative network where the nodes in the network contain propositional representations of knowledge about the current context or task, general (context-independent) declarative facts, and If/Then rules that represent possible plans of action (Mannes & Kintsch, 1991). The declarative facts and plan knowledge are similar to declarative and procedural knowledge contained in ACT-R (e.g., Anderson, 1993).

When the model simulates comprehension in the context of a specific task (e.g., reading a paragraph for a later memory test), a set of weak symbolic production rules *construct* an associative network of knowledge interrelated on the basis of superficial similarities between propositional representations of knowledge without regard to task context. This associated network of knowledge is then *integrated* via a constraint-satisfaction algorithm that propagates activation throughout the network, strengthening connections between items relevant to the current task context and inhibiting or weakening connections between irrelevant items. This integration phase results in context-sensitive knowledge activation constrained by interitem overlap and current task relevance.

The ability to simulate context-sensitive knowledge activation is most important for the present work. We are studying the construction of adaptive, novel plans of action rather than studying retrieval of known routine procedures (e.g., Holyoak, 1991). Symbolic/connectionist architectures use symbolic rules to interrelate knowledge in a network, and then spread activation throughout the network using connectionist constraint-satisfaction algorithms. This architecture has significant advantages over solely symbolic or connectionist forms for researchers interested in context-sensitive aspects of adaptive problem solving (e.g., Broadbent, 1993; Holyoak, 1991; Holyoak & Thagard, 1989; Mannes & Doane, 1991; Thagard, 1989).

Modeling UNIX User Performance

van Dijk and Kintsch (1983) and Kintsch (1988; 1994) suggest that comprehending text that describes a problem to be solved (e.g., an algebra story problem) involves retrieving relevant factual knowledge, utilizing appropriate procedural knowledge (e.g., knowledge of algebraic and arithmetic operations), and formulating a solution plan. By using Kintsch's (1988) framework, Doane et al. (1992) modeled the Doane et al. (1990) UNIX user data using a computational model called UNICOM.

In UNICOM, instructional text and the current state of the operating system serve as cues for activation of the relevant knowledge and for organizing this knowledge to produce an action sequence. The focus of our analysis is not so much on understanding the text per se, but on the way these instructions activate the UNIX knowledge relevant to the performance of the specified task.

Understanding a production task in UNICOM means generating a representation of the items in the operating system (e.g., files, commands, special input/output redirection symbols) and their interrelationships, and then using this information to generate an action plan. Knowledge of the current problem solving situation is associated with existing background knowledge, and knowledge activation is then constrained based on relevance to the problem at hand (Doane et al., 1992; Doane, Sohn, Adams, & McNamara, 1994; Sohn & Doane, 1997).

As previously mentioned, the purpose of the UNICOM analyses was to determine the cognitive prerequisites for successful production of novel UNIX command sequences. The previous knowledge and memory analyses resulting from previous UNICOM simulations directly address this question. To build a situation model effective for producing correct composite command action plans, UNICOM required four types of UNIX knowledge. In addition to knowledge required for single commands such as specific labels and functions (command syntax), the production of composites required conceptual knowledge of input/output redirection (conceptual I/O redirection), knowledge of input/output syntax (I/O syntax; e.g., "|") and knowledge regarding the input/output redirection properties for each command (command redirection; e.g., whether input and output can be redirected from/to other commands).

UNICOM also used a memory buffer that retrieved, compared, and logically ordered composite elements throughout the problem solving process. Storage resources were also necessary to keep track of the intermediate state(s) of information (file content) as input and output "flowed" between commands via redirection symbols. Thus the working memory component of UNICOM served both the storage and the processing functions described by (Just & Carpenter, 1992).

Consider the following example task to produce "sort A | head | lpr" in response to the instruction to print out the first 10 alphabetically arranged lines of a file named "A" on the line printer. To correctly produce this composite, the model activated and used command syntax knowledge relevant to sequence-dependent use of the "sort" command. It also placed in working memory the fact that the sorted contents of File A now exist. This process was repeated for the command "head." Command syntax knowledge relevant to the sequence-dependent use of “head” was activated and knowledge that the contents of file A following use of “head” now exist (the first 10 sorted lines) was placed in UNICOM’s working memory. This process was repeated a third time for the “lpr” command to print the first 10 sorted lines of the file A on the line printer. In summary, UNICOM produced the composite command by activating four types of UNIX command knowledge and by retaining results from intermediate steps in working memory.

The psychological validity of these knowledge and memory analyses was tested by asking users whose UNIX experience varied to produce composite commands. Help prompts were provided when production errors occurred (Doane et al., 1992). The help prompts were designed to assist users with the four types of UNIX knowledge and the working memory processes (i.e., sequencing the command items and keeping track of the intermediate results), required by the UNICOM simulations.

Novices in the prompting study required help retrieving and activating the four types of knowledge required for each symbol used in a composite, and this alone was not sufficient to attain correct performance. To correctly produce composites, they also required explicit help ordering composite components into the correct sequence and keeping track of intermediate results (i.e., transient states of file content). Thus, findings supported the psychological plausibility of the knowledge and memory analyses obtained from UNICOM simulations.

II. PRESENT RESEARCH

The present work provides three new contributions. First, rather than modeling prototypical users, we created individual knowledge bases for the 22 UNIX users that participated in the aforementioned prompting study. We created knowledge bases for each user based on UNIX knowledge they explicitly displayed before instruction. For example, if a simulated user did not explicitly enter the command “sort” before being prompted, then knowledge of the “sort” command was not included in their UNICOM knowledge base.

To simulate a given individual, UNICOM accessed the appropriate knowledge base, “received” task instructions, and then proceeded to produce action plans for 21 composite commands in the same order attempted by the simulated individual. If any command production errors were made, UNICOM received training (help) prompts in the order viewed by modeled individuals, reprocessed the modified knowledge base via construction-integration cycles and then tried again to produce the correct command. This attempt-prompt-attempt process was repeated until the correct command was produced, and the entire procedure was repeated twenty-one times to simulate the problem set given to the modeled users in the Doane et al. (1992) study.

Second, we developed and implemented comprehension-based learning algorithms in UNICOM. If a prompt providing command syntax knowledge about “sort” was viewed by the model, it was included as contextual information in subsequent construction-integration cycles, and retained as long as it was one of the four most activated pieces of information in working memory. Prompted knowledge not initially included in a given user’s knowledge base was “learned” if it was used in subsequent production attempts (regardless of correctness) while retained in working memory. Learning was implemented by permanently storing the prompted knowledge in the modeled individual's knowledge base. These learning algorithms are derived from Kintsch's (1988) theory of comprehension, and allows us to test the hypothesis that comprehension-based algorithms can predict learning from instructions in a rigorous manner. Indirectly this research allows us to quantitatively test the centrality of comprehension for learning.

Third, our rigorous testing methods allow us to test the predictive validity of UNICOM. Knowledge-based models have long been criticized as being descriptive representations of unfalsifiable theories (e.g., Dreyfus, 1992, 1996; Anderson, 1989). In the present work we construct individual knowledge bases on the basis of observations of a small portion of human performance data. We then use the identical UNICOM architecture and automated modeling procedures to simulate unobserved user performance. That is, once UNICOM accesses a given individuals knowledge base, the prompting simulation of that individual begins, and no individual-specific simulation modifications are made within or between participant simulations. As a result, we can quantitatively compare UNICOM and human performance as a function of prompt on each of the 21 problems.

This type of rigorous predictive validation procedure is more commonly used by researchers that develop mathematical models (e.g., connectionist models) of cognition than by researchers that develop knowledge-based models. However, descriptive models of individual performance and learning strategies have been validated (e.g., Thagard, 1989; Anderson, 1993; Lovett & Anderson, 1996; Recker & Pirolli, 1995).

The following sections provide a summary of the Doane et al. (1992) prompting study to facilitate exposition of the present research. Subsequent sections describe the UNICOM model, simulation procedures and findings.

III. EMPIRICAL PROMPTING STUDY

Method

Participants

Twenty-two engineering majors completed 21 composite command production tasks. Experience with UNIX ranged from less than 1.25 years for novices ($n = 10$), between 1.25 and 3.0 years for intermediates ($n = 8$), and greater than 3 years for experts ($n = 4$).

Procedure

All production tasks were performed on a computer. The stimuli were task statements, a fixed directory of file names, a series of help prompts, and three "error cards" presented by the experimenter. For each composite problem, participants typed their command(s) using the keyboard and then used the mouse to "click" a button displayed on the screen to have the computer score their answer for accuracy. The task instructions described actions that could best be accomplished by combining two or three commands through the use of redirection. Accompanying the task statement was a fixed directory listing of all file names that were used in the experiment.

If the computer scored an attempt as incorrect, a help prompt was displayed, and remained on the screen until the correct composite was produced. Prompts were displayed one at a time in a predetermined order regardless of the type of error made by the participant. The system simply parsed an answer to determine if it contained the required syntax in the requisite order and if it did not, then the system would display the next prompt in the sequence.

The prompts were designed to assist with the four types of knowledge and with memory processes that our previous UNICOM simulations had suggested were necessary to successfully produce composites. Figure 1a shows an example instruction and prompt screen (with all prompts displayed) for the composite "nroff -ms ATT2 > ATT1." Prompt 1 simply parses the task statement into relevant command concepts. Prompt 2 gives command syntax knowledge. Prompt 3 provides an abstract description of the relevant I/O redirection conceptual knowledge. Prompt 4 provides relevant I/O redirection syntax knowledge. Prompts 5 through 7 provide the user with working memory assistance. Prompts 5 and 7 do this by simply repeating the knowledge elements provided in preceding prompts. Prompt 6 provides the first assistance ordering the command and redirection symbols in the appropriate sequence, and this is intended to assist the user with command redirection knowledge and with keeping track of intermediate results. Ordering information is repeated in Prompt 8. Finally, Prompt 9 gives the user the correct production.

RESULTS AND DISCUSSION

Scoring Composite Productions

The computer scored user productions as correct if they matched the syntax of the idealized command (spaces were not counted). As an example, users could not substitute "sort file1>temp; head temp>file2" for the command "sort file1 | head>file2." Doane et al. (1992) organized their performance data into three problem groups based on the percentage new knowledge required to complete the composite; 0% (eight problems), 1 to 59% (10 problems), and 60 to 100% (three problems). New knowledge was defined as any of the four types of UNIX command knowledge required for a given composite not previously encountered in the experiment. For example, the command syntax knowledge for "sort" is counted as new knowledge only once; the first time it is required by a composite task.

Analyses of Correct Productions

The left side of Figure 2 shows the mean cumulative percentage of correct composite productions for novices, intermediates, and experts for each of the three percentage new knowledge problem groupings as a function of prompt.[1] The data in Figure 2 are

(a)	(b)
Task Description	
Format the text in ATT2 using the -ms macro package and store the formatted version in ATT1	(REQUEST STORE FORMATTED FILE^ATT2 USING^MS^MACRO IN FILE^ATT1) (OUTCOME STORE FORMATTED FILE^ATT2 USING^MS^MACRO IN FILE^ATT1)
Prompts	
Prompt 1 You will need to use the following command One that will format the contents of a file using the -ms macro package	Prompt 1: (KNOW FORMAT FILE^NAME USING^MS^MACRO)
Prompt 2 You will need to use this command nroff-ms will format the contents of a file using the -ms macro package	Prompt 2: (KNOW NROFF FORMAT FILE^NAME USING^MS^MACRO) (KNOW NROFF TAKES^FLAG -MS)
Prompt 3 You will need to use a special symbol that redirects command output to a file	Prompt 3: (KNOW REDIRECT INPUT TO OUTPUT FROM) (KNOW REDIRECT FROM COMMAND TO FILE)
Prompt 4 You will need to use the arrow symbol " > " that redirects output from a command to a file	Prompt 4: (KNOW FILTER2 REDIRECT FROM COMMAND TO FILE)
Prompt 5 You will need to use the arrow symbol " > " and the command nroff-ms	Prompt 5: (KNOW FILTER2) (KNOW NROFF) (KNOW NROFF TAKES^FLAG -MS)
Prompt 6 You'll need to use an nroff-ms on ATT2 (which will output the formatted contents of ATT2), and you'll need to redirect this output as input to ATT1	Prompt 6: (KNOW NROFF FILE^ATT2) (KNOW ON^SCREEN FILE^FORMATTED^ATT2) (KNOW REDIRECT FROM NROFF TO FILE^ATT1)
Prompt 7 You will need to use exactly the following command elements (though not necessarily in this order): >, nroff-ms	Prompt 7: (KNOW FILTER2) (KNOW NROFF) (KNOW NROFF TAKES^FLAG -MS)
Prompt 8 You'll need to use the command nroff-ms followed by the arrow symbol " > "	Prompt 8: (KNOW FILTER2) (KNOW NROFF) (KNOW NROFF TAKES^FLAG -MS)
Prompt 9 The correct production is nroff-ms ATT2>ATT1 Please enter this production now	Prompt 9: (KNOW NROFF FORMAT FILE^ATT2 USING^MS^MACRO) (KNOW NROFF TAKES^FLAG -MS) (KNOW FILTER2 REDIRECT FROM NROFF TO FILE^ATT1)

Figure 1. Example task description and prompts for the problem "nroff -ms ATT2 > ATT1." The left side of the figure (Figure 1a) shows the task description and prompts as shown to UNIX users and the right side (Figure 1b) shows the corresponding propositional representations used by UNICOM.

cumulative; UNIX users who produced an accurate composite at Prompt 4 were included as correct data points at Prompts 5 through 9 as well. Thus, at Prompt 9, all of the users in each expertise group were at 100% accuracy.

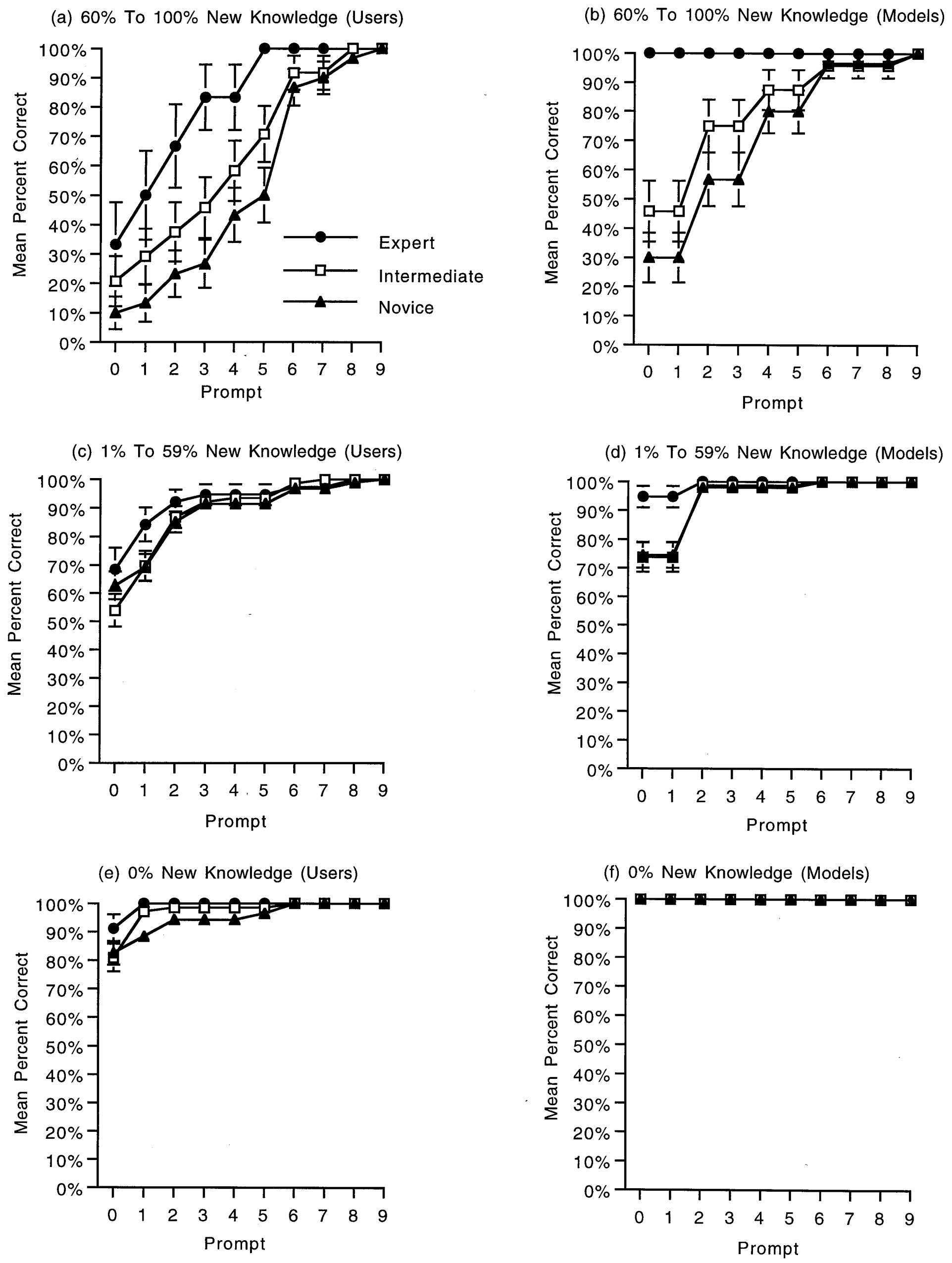

Figure 2. Mean percent correct productions for novice, intermediate, and expert UNIX user groups and corresponding user models as a function of prompt for 60 to 100%, 1 to 59%, and 0% new knowledge problems.

Looking at the left side of Figure 2, experts have the highest accuracy overall. Focusing on the 60 to 100% new knowledge problems (Figure 2a), prompts seem to differentially affect the three expertise groups. For example, the increase in performance accuracy from Prompts 1 to 2, Prompts 3 to 4, and Prompts 5 to 6 are largest for novices, suggesting that command syntax (Prompt 2), I/O syntax (Prompt 4), and command redirection (Prompt 6) prompts provide useful assistance. Other prompts show little or no effect on accuracy, suggesting that the information provided is already known to users, or simply does not facilitate production accuracy. Perfect performance is observed for Experts after relatively few prompts, whereas Novices require exposure to Prompt 9 which gives them the correct composite production (see Figure 1a).

Knowledge Scoring

The amount of correct knowledge displayed in each incorrect attempt can also be scored to allow qualitative performance analyses. For example, the problem described in Figure 1a requires each of the four types of knowledge previously described. Three pieces of command syntax knowledge are required, including "nroff" is a command name, "nroff" takes an "-ms" flag, and "nroff" takes a file argument. One I/O redirection syntax fact is required; ">" redirects output from a command to a file. The conceptual I/O knowledge required is that redirection of input and output can occur, and that I/O redirection can occur from commands to files. The required command redirection knowledge is that "nroff" output can be redirected to a file. The procedure for scoring knowledge is detailed in section A.1 of Appendix A.

Doane et al. (1992) calculated the percentage of each knowledge type displayed by each composite production attempt. In our example, an incorrect attempt of "nroff file" would be scored as containing 66% of the requisite command syntax knowledge, and 0% of the remaining three knowledge types.

Knowledge Analyses

The left side of Figure 3 shows the mean knowledge scores reported by Doane et al. (1992) for the 60 to 100% new knowledge problems.[2] The arrow markers in Figure 3 indicate the first prompt that provided information relevant to the knowledge type displayed in each graph. In Figure 3a, command syntax knowledge scores are displayed, and the arrow marker indicates that Prompt 2 was the first Prompt to provide command syntax knowledge. The graphs are cumulative, so the change in knowledge scores between Prompts 1 and 2 in Figure 3a indicates the effect of Prompt 2. The component knowledge scores are higher than the percentage correct scores shown in Figure 2a. This occurs because an attempt can show high, but not perfect component knowledge, and component knowledge must be perfect for an attempt to be entered as correct in Figure 2.

To summarize the empirical results shown on the left side of Figure 3, the prompts corresponding to three of the four knowledge types have a systematic influence on their respective knowledge scores following presentation. The exception is Figure 3c, which suggests that the conceptual I/O prompt that provides an abstract statement about redirection (e.g., see Figure 1a) does not show a systematic influence on knowledge scores.

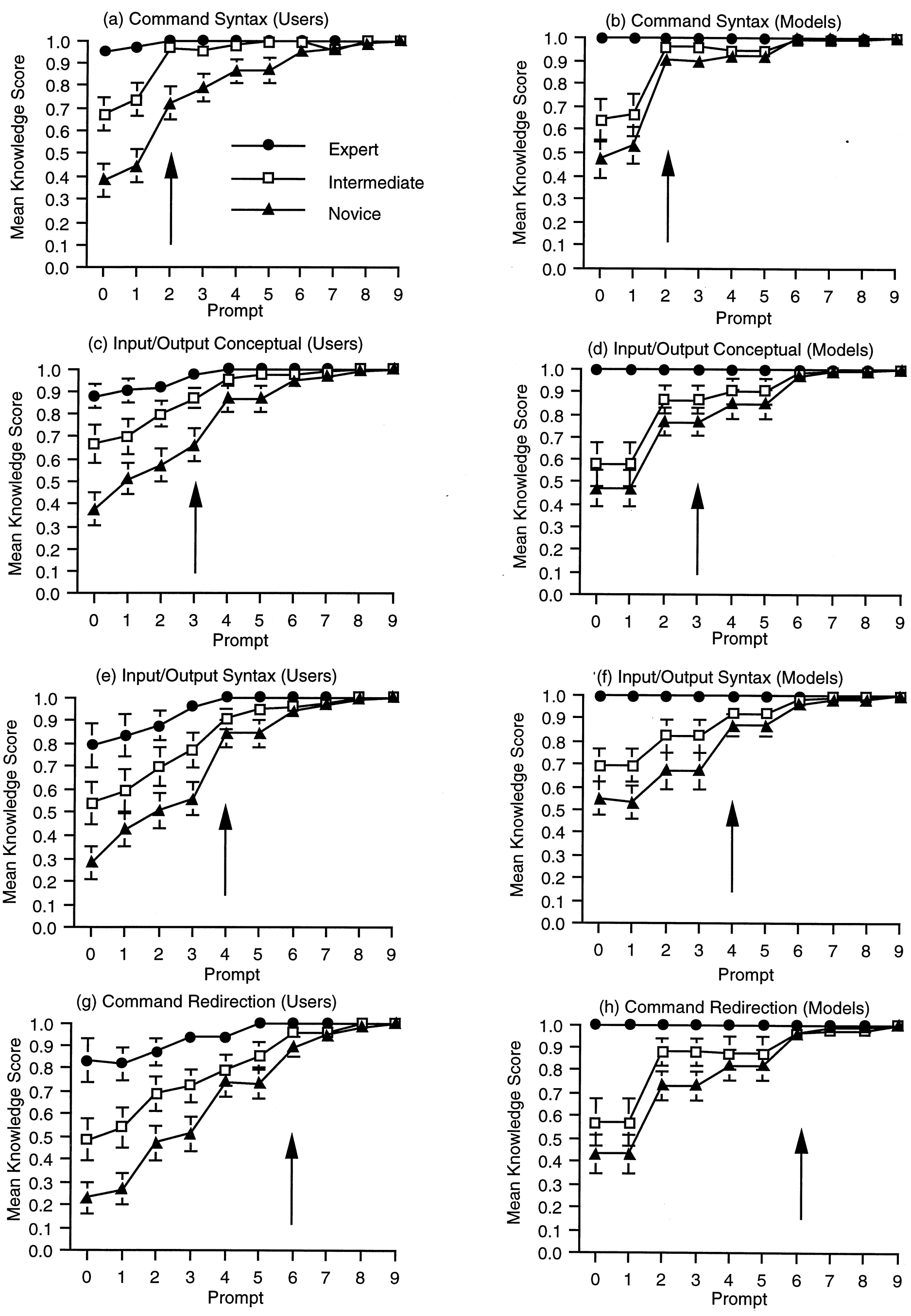

Figure 3. Mean command syntax, I/O conceptual, I/O syntax, and command redirection knowledge scores as a function of prompt for novice, intermediate, and expert UNIX user groups and corresponding user models for 60 to 100% new knowledge problems.

Overall, the results show enough systematic variability to rule out the alternative argument that increase in performance as a function of prompt is due to random trial and error.

Scoring Memory-Related Errors

To measure the influence of working memory deficits on production performance, we examined omission (deletion) and substitution errors symptomatic of working memory deficits (e.g., Anderson & Jeffries, 1985; Doane et al., 1992; Murdock, 1974; Sohn & Doane, 1997). Deletion errors were scored by counting the number of component items omitted in each command attempt. Possible omitted items included filenames (e.g., “JOB.P,” “OLDDT”), utilities (e.g., “ls,” “sort”), and specialty symbols (e.g., “|,” “>”). Component items that seemed to be transposed (e.g., “sort | head” for “head | sort”) for filenames, utilities, and specialty symbols and attempts that included true command substitutions (e.g., “tail | sort” for “head | sort”) were counted as substitution errors. The scoring procedure for production errors is detailed in section A.2 of Appendix A.

Error Analyses

The left side of Figure 4 shows the mean user group deletion and substitution errors as a function of prompt for the 60 to 100% new knowledge problems. The novices make the most deletion and substitution errors, as expected. Most interesting is the large drop in novice substitution errors following the presentation of Prompt 6. This prompt is the first to provide ordering information intended to reduce the working memory load caused by keeping track of intermediate results.

IV. SIMULATIONS

Before describing methods used in the simulation studies, we need to summarize the structure of the UNICOM model. This is followed by a description of procedures used to construct twenty-two initial knowledge bases to represent each user in the Doane et al. (1992) prompting study. Then, we will describe the procedures used to simulate a UNIX user’s prompt study performance.

UNICOM Knowledge Representations

UNICOM represents human memory as an associative network in which each node in the network corresponds to propositional representations of knowledge. Each proposition contains a predicate and some number of arguments, which in UNICOM represent knowledge about the computing domain or the present task. For example, the propositionalization of the sentence “A file exists in a directory.” would appear as (EXIST IN^DIRECTORY FILE). Long predicates or arguments use a “^” (e.g., IN^DIRECTORY) to represent a single semantic unit.

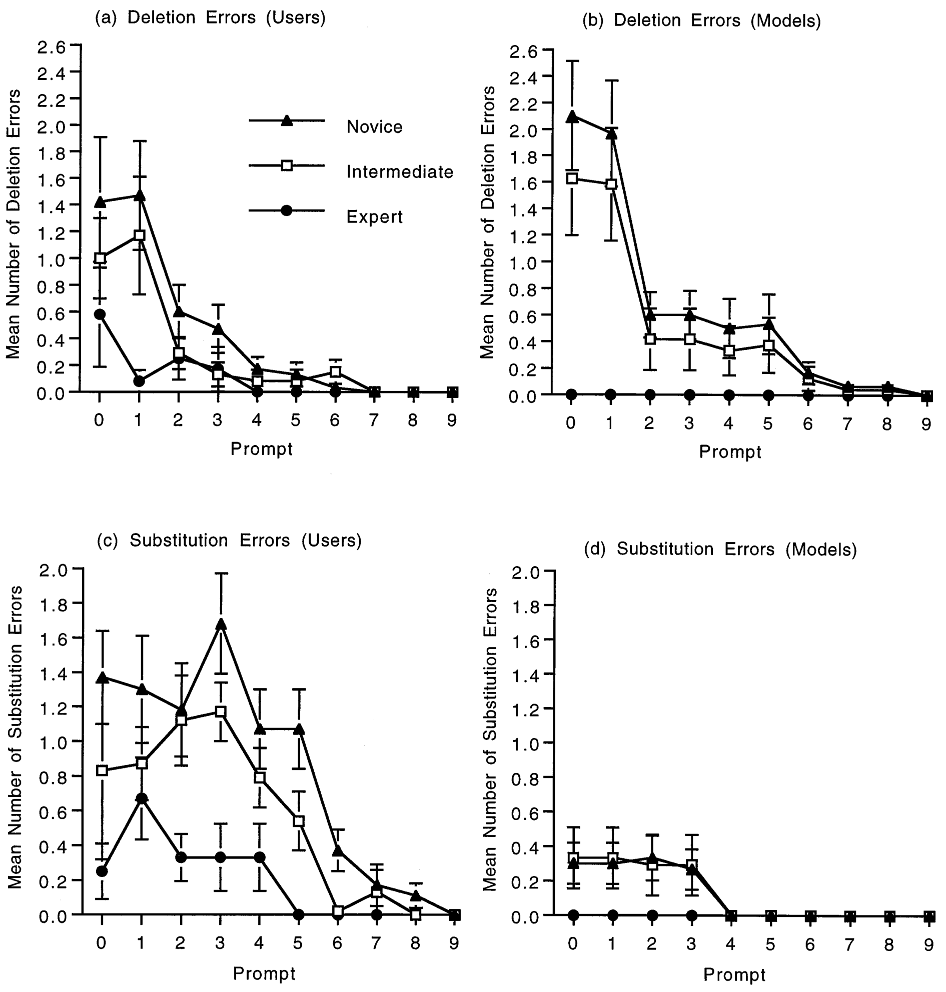

Figure 4. Mean number of deletion and substitution errors as a function of prompt for novice, intermediate, and expert UNIX user groups and corresponding user models for 60 to 100% new knowledge problems.

UNICOM represents the three major classes of knowledge proposed by Mannes and Kintsch (1991); world, general (e.g., declarative facts), and plan element knowledge (e.g., procedural knowledge represented as if/then rules; see Table 1). We will now describe how each knowledge class was represented in UNICOM to simulate UNIX command production performance in a training environment.[3]

World Knowledge

The first class of knowledge, world knowledge, represents the current state of the world. Examples of world knowledge in UNICOM include knowledge of the current task,

TABLE 1
Examples of Knowledge Representations in UNICOM

Type of knowledge	Abbreviated propositional representation	
World knowledge		
	File exists in directory	(EXIST IN^DIRECTORY FILE)
	At system level	(AT^LEVEL SYSTEM)
	Goal is to format file	(OUTCOME FORMAT FILE)
General knowledge		
Command syntax	"nroff" formats file	(KNOW NROFF FORMAT FILE)
I/O syntax	">" redirects output from command to file	(KNOW FILTER2 REDIRECT FROM COMMAND TO FILE)
Conceptual I/O	I/O can be redirected from command to file	(KNOW REDIRECT FROM COMMAND TO FILE)
Com. redirect.	"nroff" output can be redirected to file	(KNOW REDIRECT FROM NROFF TO FILE)
Plan knowledge		
Name:	Format contents of a file	(DO FORMAT FILE)
Preconditions:	Know "nroff" formats file	(KNOW NROFF FORMAT FILE)
	Know "nroff" "-ms" flag	(KNOW NROFF TAKES^FLAG -MS)
	Know file exists in directory	(KNOW IN^DIRECTORY FILE)
Outcome(s): formatted file exists		(KNOW FORMATTED^FILE ON^SCREEN)

existing files on the current directory, and the system in use (UNIX). These facts are contextually sensitive and fluid, changing as the task and simulated performance progresses. For example, if a file is deleted, if the state of the system changes (e.g., a user moves from the system level to the editor), or if new task instructions (prompts) arrive, the world knowledge will reflect this change.

General Knowledge

The next class of knowledge, general knowledge, refers to factual (declarative) knowledge about UNIX. In UNICOM, general knowledge includes the aforementioned command syntax, I/O syntax, conceptual I/O redirection, and command redirection UNIX knowledge required to produce correct composite commands. We can provide a specific example of the four types of general knowledge by considering the command "nroff -ms ATT2 > ATT1," that formats the contents of the file ATT2 using the utility nroff with the -ms flag, and then stores the results in the file ATT1. The required command syntax knowledge is knowing the nroff command and the -ms flag. The I/O redirection syntax knowledge is knowing the ">" redirection symbol. Conceptual facts about I/O redirection include the knowledge that redirection of input and output can occur between commands. This is separate from the syntax specific knowledge of I/O redirection symbols. Command redirection knowledge is knowing that the output of nroff can be redirected to a file.

Plan Element Knowledge

The third class of UNICOM knowledge, plan elements, represent "executable" (procedural) knowledge. Plan elements describe actions that can be taken in the world, and they specify conditions under which actions can be taken. Thus, users have condition-action rules that can be executed if conditions are correct. Plan elements are three-part knowledge structures, including name, preconditions, and outcome fields (see Table 1). The name field is self explanatory. The preconditions refer to knowledge of the world or general knowledge that must exist before a plan element can be executed. For example, a plan element that formats file content requires that the unformatted contents of the file exist in the world before it can be fired. Plan element outcome fields are added to the model's world knowledge when the plan element is fired. For example, once a file is formatted, the world knowledge will change to reflect the fact that the formatted contents of the file exist in the world.

Constructing Individual Knowledge Bases

We built twenty-two starting knowledge bases by evaluating each user's UNIX knowledge using a small portion of empirical performance data. This determined the initial contents of the 22 knowledge bases accessed by UNICOM to simulate each individual. We scored missing knowledge as well as incorrect "buggy" knowledge using an overlay method (see VanLehn, 1988). To determine the starting state of a participant's UNIX background knowledge, we scored each knowledge component of the participant's response as to whether it was made before or after explicit instruction. If it was made before instruction regarding that specific knowledge component, it was assumed the participant had the knowledge before the experimental session and it was thus added to their starting knowledge base. If it was made only after instruction, it was not added. For example, if a user used the command "nroff" before we provided any information about the command, then the user's starting knowledge base would include nroff command syntax knowledge and the plan to produce nroff. If the user tried to redirect the input or output of nroff before we provided information about nroff redirection properties (e.g., "nroff -ms filename | lpr") then their starting knowledge base include the command redirection knowledge that nroff output can be redirected as input to another command.

To account for users' erroneous as well as correct command productions, we also scored each user's answers to determine what incorrect knowledge the user displayed before instruction on that knowledge. For example, if a user incorrectly used a pipe ("|") instead of a filter (">") to redirect output from a command as input to a file, then the incorrect knowledge that pipe redirects output from a command to a file would be included in their knowledge base.

Executing Individual Models

UNICOM was run to simulate each user's performance in command production tasks, responding to instructional and help prompts identical to those given to each of our users. The procedures used to simulate a UNIX user's performance are schematically represented in Figure 5. The left side of Figure 5 depicts the major steps required to complete a construction-integration cycle (C/I cycle). The right side provides an abstract represen-

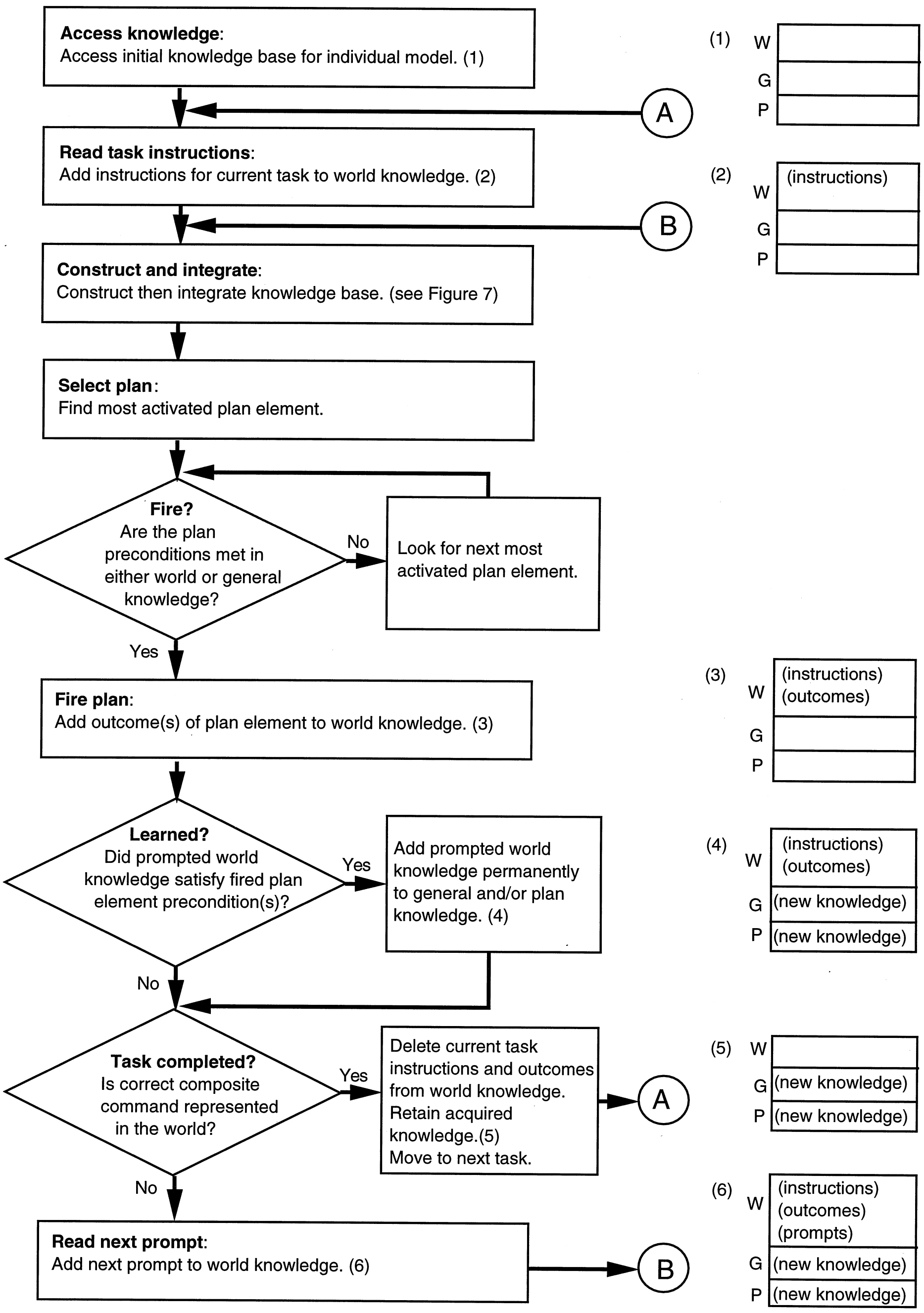
Access knowledge:
Access initial knowledge base for individual model. (1)
Read task instructions:
Add instructions for current task to world knowledge. (2)
Construct and integrate:
Construct then integrate knowledge base. (see Figure 7)
Select plan:
Find most activated plan element.
Fire?
Are the plan preconditions met in either world or general knowledge?
No
Look for next most activated plan element.
Yes
Fire plan:
Add outcome(s) of plan element to world knowledge. (3)
Learned?
Did prompted world knowledge satisfy fired plan element precondition(s)?
Yes
Add prompted world knowledge permanently to general and/or plan knowledge. (4)
No
Task completed?
Is correct composite command represented in the world?
Yes
Delete current task instructions and outcomes from world knowledge. Retain acquired knowledge.(5) Move to next task.
No
Read next prompt:
Add next prompt to world knowledge. (6)
A
B
(1) W G P
(2) W (instructions) G P
(3) W (instructions) (outcomes) G P
(4) W (instructions) (outcomes) G (new knowledge) P (new knowledge)
(5) W G (new knowledge) P (new knowledge)
(6) W (instructions) (outcomes) (prompts) G (new knowledge) P (new knowledge)

Figure 5. Schematic representation of the procedures used to simulate one UNIX user's prompt study performance. (The left side of the figure shows the procedural sequence of steps and the right side graphically depicts corresponding changes in the user's knowledge base. W = World knowledge; G = General knowledge; P = Plan knowledge).

tation of the contents of a user knowledge base following each major step. The steps depicted in Figure 5 are discussed in turn below.

Accessing User Knowledge and Task Instructions

After knowledge for a given user was represented in an initial knowledge base, the problem description for the first composite production task was added to the in-the-world knowledge (see Figure 5). At this point the knowledge base was considered ready to begin a series of construction-integration cycles with the goal of selecting a series of plan elements that together constituted an action plan. Specific simulation details are provided in Appendix B.

Relating Knowledge in UNICOM

Knowledge about the domain and a particular task is represented in a distributed manner although the node content remains symbolic and identifiable. It is the pattern of activation across nodes that determines the current model of the problem situation. The following sections detail how these symbolic nodes are interrelated in two distinct stages that are uniquely structured to represent comprehension.

Construction

During construction, UNICOM computes relationships between propositions in the knowledge base (k) to construct a task-specific network of associated knowledge. The model uses low-level rules to construct a symmetric task connectivity matrix (c), where each node (c(i,j)) contains a numeric value corresponding to the calculated strength of the relationship between k(i) and k(j). Equations for the strength calculations are detailed in section C.1 of Appendix C. The resulting network, depicted in Figure 6, represents the unconstrained relationships between knowledge brought to bear to accomplish this specific task. The low-level rules used to determine if two nodes were related did not vary between or within simulations. In fact, the internode relationships have not changed since Kintsch's (1988) model introduction.

Binding. Before constructing a task connectivity matrix, UNICOM is given the task description in propositional form, as shown in Table 2. UNICOM binds specific objects mentioned in the task description (e.g., the file name PROP1) to proposition fields that contain the appropriate variable (e.g., FILE^NAME). For example, if a task description mentions the existence of a particular file called PROP1, all the plan elements with the argument FILE^NAME are duplicated and bound to the file PROP1. That is, FILE^NAME becomes FILE^PROP1, where "^" is the symbol used to concatenate the two arguments together. The binding process is repeated each time a unique filename is mentioned (e.g., PROP1 and JOB.P). If the task description includes an initial file name and the modified contents of the initial file (e.g., PROP1 and ALPHABETICALLY

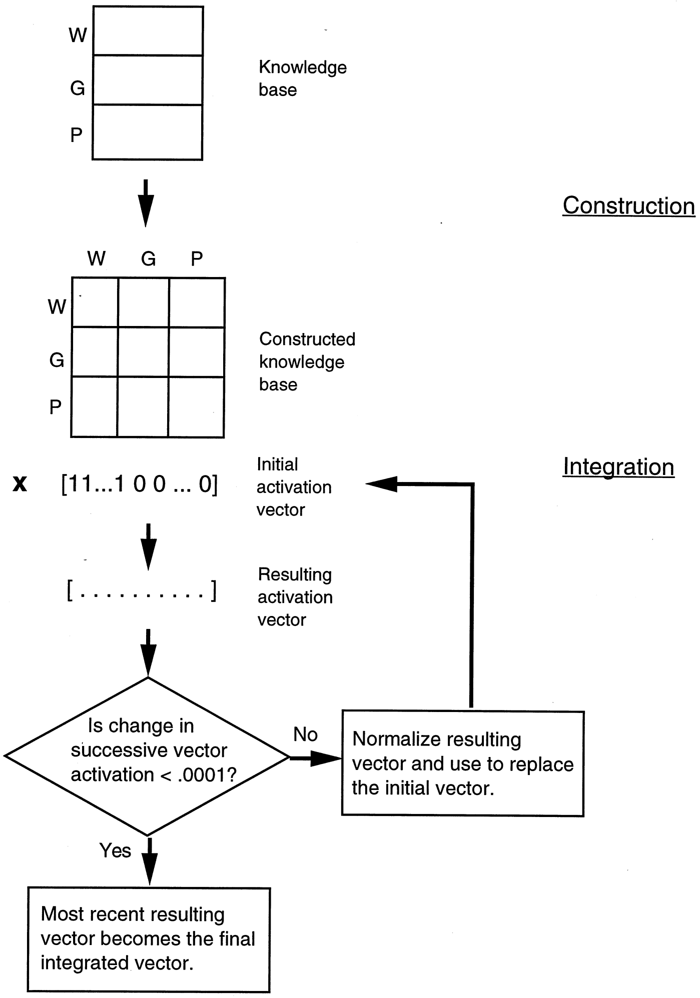
W
G
P
Knowledge base
Construction
W G P
W
G
P
Constructed knowledge base
X [11...1 0 0 ... 0]
Initial activation vector
Integration
[. . . . .]
Resulting activation vector
Is change in successive vector activation < .0001?
No
Normalize resulting vector and use to replace the initial vector.
Yes
Most recent resulting vector becomes the final integrated vector.

Figure 6. Schematic representation of UNICOM computations during one construction/integration cycle. (W = World knowledge; G = General knowledge; P = Plan knowledge).

^ARRANGED^PROP1), they are treated as unique files and propositions are duplicated accordingly. Functionally, this duplication allows UNICOM to represent the unconstrained binding process hypothesized by Kintsch (1988).

TABLE 2
Abbreviated Example Task Description Propositions for "sort PROP1 | head"

P1	(KNOW FILE^PROP1)
P2	(REQUEST DISPLAY FIRST^TEN^LINES ALPHABETICALLY^ARRANGED ON^SCREEN)
P3	(OUTCOME DISPLAY FIRST^TEN^LINES ALPHABETICALLY^ARRANGED ON^SCREEN)

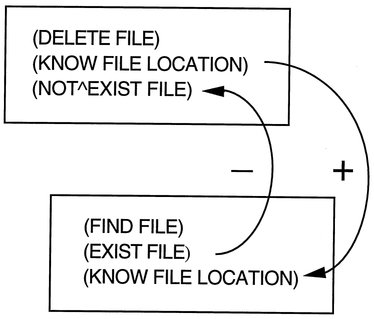

Figure 7. Example precondition and outcome interplan relationships. ("+" = positive; "−" = inhibitory).

Associative Relationships. Associative relationships between each pair of propositional nodes (c(i,j)) in the network are based on the number of shared arguments, and completely embedded propositions. Propositions are linked with a positive weight for each argument shared. For example, the propositions (KNOW NROFF FORMAT FILE) and (EXISTS UNFORMATTED FILE) share one argument (FILE). The corresponding nodes in the network (c(i,j); c(j,i)) would be positively linked with a weight of 0.4 because they share this argument. If one proposition is entirely embedded in another, the two propositions are linked with a weight of 0.8. Overlap with prompt proposition results in an overlap of 0.2. Although these relationships provide only a crude approximation of propositional relatedness, they have been effective in prior simulations of text comprehension and memory (Kintsch, 1988; Miller & Kintsch, 1980).

Plan Element Relationships. Three fields comprise a plan element; name, precondition, and outcome. Propositions representing these fields are represented as a single node in the network. Only the name field of a plan element is included in the aforementioned calculations of semantic and associative relatedness.

Overlap between plan element precondition and outcome fields is calculated to estimate causal relationships between plan elements. For example, if the outcome(s) of one plan (p(j)) satisfy the precondition(s) of another (p(i)), then a positive asymmetric weight of 0.7 will be added to the respective c(i,j) node in the task connectivity network. Functionally this allows the activation of p(i) to flow to p(j) during integration. If an outcome(s) of one plan element negates the precondition(s) of another, then an asymmetric inhibitory link of −10.0 is entered into the corresponding c(i,j) node.

Figure 7 depicts these causal relations for two abbreviated example plan elements to delete and to find a file. A positive link exists from (DELETE FILE) to (FIND FILE) because the DELETE plan element precondition (KNOW FILE LOCATION) is satisfied by the outcome of the FIND plan element. An inhibitory interplan relation from (FIND FILE) to (DELETE FILE) exists as well (see Figure 7). The outcome (NOT^EXIST FILE) of the DELETE plan element negates the precondition (EXIST FILE) for FIND.

Two relations between plans and world knowledge are calculated. First, if the outcome(s) of a plan element already exists in the world, then an asymmetric inhibitory link of −10.00 exists between each proposition in the world knowledge that matches the outcome(s) propositions. For example, if the outcome of the FIND plan element (KNOW FILE LOCATION) exists in the world, the FIND plan element is inhibited during integration. Another related inhibitory link of −0.4 is used between traces representing actions previously accomplished (e.g., TRACE FILE EXISTS IN-THE-WORLD) and name propositions of plans that will accomplish the already executed goal (e.g., FIND FILE), and share an argument overlap. These two inhibitory relationships are calculated to keep the model from repeating itself.

Second, if the name or the outcome fields of a plan element match the REQUEST and OUTCOME propositions that represent the current task in the world, a positive link of 1.5 is made between the matching propositions. Specifically, a symmetric weight of 1.5 is applied to the respective c(i,j) and c(j,i) that represent links between matching REQUEST and plan element name propositions, and OUTCOME and plan element outcome propositions.

To summarize, UNICOM uses the construction relationships and weights devised by Mannes and Kintsch (1991), including argument overlap weights of 0.4 and 0.2, a proposition embedding weight of 0.8, plan element precondition and outcome mappings of 0.7 (positive), and −10.00, and −0.4 (inhibitory), and a weight of 1.5 for the aforementioned REQUEST and OUTCOME propositions that match plan element names and outcome fields, respectively. Where no link was specified, connections were set to zero. These parameter values have been used in all of our UNICOM research, and remain constant here. As suggested by Thagard (1989), this weight stability is critical for assessing the reliability of a cognitive architecture across simulation efforts.

Integration

The constructed network of knowledge represents unconstrained relations between knowledge elements. To develop a situation model (e.g., Kintsch, 1988), this knowledge must be integrated by using constraint-based activation to spread activation throughout the network. This process essentially strengthens the activation of knowledge elements consistent with the task context, and dampens the activation of others. The simple linear algorithm used to simulate integration is illustrated in Figure 6 and formulas are provided in section C.2 of Appendix C.

Computationally, integration constitutes the repeated postmultiplication of the constructed network (matrix) by a vector. The vector contains numbers that represent the current activation of each knowledge element represented (e.g., the value of the first item in the vector represents the current state of activation of the first proposition in the knowledge base, and so on).

As depicted in Figure 6, the initial vector values corresponding to in-the-world knowledge are set to 1.0, and all others are set to 0. Functionally, this allows the in-the-world propositions that represent the current task context to drive the spread of activation. This “initial activation vector” is postmultiplied by the connectivity matrix resulting from the construction process, and a “resulting activation vector” is produced. After each multiplication, the vector weights corresponding to in-the-world items are reset to 1.0, and the remaining items are normalized to ensure their sum is a constant value across integrations.

The iterative integration process stops when the difference between two successive activation vectors is less than 0.0001. At this point, the resulting activation vector becomes the final activation vector and represents the stabilized activation of knowledge. The final activation vector is then used by the model to make executive decisions regarding the next plan element to fire.

Plan Consideration and Selection

The model finds the most activated plan element in the final activation vector, and determines whether its preconditions exist in-the-world or general knowledge. If they exist, then the plan is selected to fire, and its outcome propositions are added to the world knowledge (see Figure 5). If they do not exist, then this process is repeated using the next-most-activated plan element until a plan can be fired. Once the world knowledge has been modified, the construction phase begins again (see Figure 5). Construction-integration cycles continue until the network constructed by the model represents a plan of action that will accomplish the specified task.

Prompt Representation

If the model selects a plan of action that is not correct, (regardless of the user’s actual performance), then proposition(s) representing the appropriate prompt in sequence is added to the world knowledge (i.e., The model is given Prompt 1 after the first erroneous attempt, Prompt 2 for the second, etc., to represent the prompts shown in Figure 1a). Figure 1b shows example propositions that represent the prompts for the problem “nroff -ms ATT2 > ATT1.”

The proposition that represents Prompt 1 contains relevant command concepts. Prompts 2, 3, and 4 represent command syntax, I/O concept, and I/O syntax knowledge respectively (see Table 1). Prompts 5 and 7 propositions represent command names and symbol names. Prompt 6 provides the ordering of elements that make up the composite and assists by keeping track of intermediate results. This prompt is represented by propositions that are bound to command names, intermediate results, relevant I/O redirection concepts, and a specific file name, (e.g., file^ATT1; see Prompt 6 in Figure 1a). Prompt 8 includes the same propositions as for Prompts 5 and 7. Prompt 9, which gives users the correct production, is represented by binding command and file names to the Prompt 2 and Prompt 4 propositions.

When Prompts 2 and 4 (command and I/O syntax prompts) are added to world knowledge, corresponding plan elements are also added to plan knowledge. For example, if Prompt 2 in Figure 1b is added to the world knowledge, then the plan element that contains the procedural knowledge required to execute that command is added to plan knowledge. (If the plan element already exists in the knowledge base, then plan knowledge remains unchanged.)

Comprehension-Based Learning

For the present purposes learning is measured as the ability to use prompted knowledge in subsequent production attempts. We assume that learning occurs if the first use of prompted knowledge occurs following prompt presentation. That is, if a user does not display knowledge of the nroff command until prompted with nroff command syntax knowledge, then we assume that they learned nroff syntax knowledge from the prompt.

Mechanism. In UNICOM, the activation of prompted knowledge is a central component of the simulated learning mechanism. As previously stated, activation of knowledge is constrained associative relationships between knowledge in the world and preexisting background knowledge. The activation of prompted knowledge is presumed to dictate the probability of its use in subsequent productions. Computationally, learning is represented in UNICOM as the transfer of prompt propositions from temporary in-the-world knowledge to permanent general or plan element knowledge. To be transferred from in-the-world knowledge to general or plan element knowledge, a prompt proposition must be retained in world knowledge and must satisfy a precondition of a plan element being considered for firing.

Modeling Memory Constraints. Prompt propositions are retained in the world knowledge based on their activation. The model retains the current prompt propositions and four of the most activated previous prompt propositions in-the-world knowledge. Four was an approximation of working memory capacity that served as a constant across all simulations.

Because the activation all propositions are constrained by their relevance to the current task context, we are simulating context-sensitive working memory limitations. Previously prompted propositions that are not among the four most activated are deleted, along with any temporary plan elements that have not been used (i.e., those added for Prompts 2 and 4). Thus, if a prompt proposition is dropped from working memory before it is used to satisfy a precondition, it is not transferred to general or plan element knowledge and is deleted from the world knowledge. In other words, if this takes place UNICOM does not learn the knowledge in the prompt proposition.

Adding Correct Knowledge. If in-the-world prompt propositions satisfied the preconditions of a plan selected for firing, they (and their corresponding plan elements) were permanently added to the knowledge base. In-the-world propositions are added to general knowledge, and plan element(s) are added to plan knowledge. Thus, prompt propositions that obtain sufficient activation to be retained in the world knowledge and also satisfy preconditions of a plan element that is fired are considered as learned by UNICOM.

Deleting Incorrect Knowledge. To simulate the correction of misconceptions, the model presumed that prompted knowledge was correct. When prompted knowledge was permanently added to the knowledge base, the model checked for contradictions or negations in preexisting knowledge. If any occurred, then the erroneous preexisting knowledge was deleted.

Example UNICOM Simulation

Overview of Model Execution

For plans to be selected for execution, they must be contextually relevant to the current task (as dictated by the world knowledge) and the facts that allow them to fire (preconditions) must exist in the knowledge base. For example, to execute the plan to do a “nroff” on a file, the plan element to format a file must be activated, and to be activated, it must be relevant to the current task (i.e., the command to alphabetically arrange the contents of a file, “sort,” would not receive much activation if the model’s task was to format a file, because it is not contextually relevant). In addition, the preconditions of the plan element including the prerequisite state of the world and the presence of general knowledge must be fulfilled. For example, the model must be at the system level (state of the world knowledge), they must know the “nroff” command, and the file that is to be formatted must exist in the world.

Many plan elements are selected in sequence to form an entire action plan. The model operates in a cyclical fashion, firing the most activated plan element whose preconditions exist in the world or in general knowledge. The outcome proposition(s) of the fired plan element are then added to the world, and construction begins again with the modified knowledge base. The selection of the next plan element to fire is determined by the modified contents of the world following integration.

Example Task Performance

In this section, we clarify how UNICOM actually works throughout the planning process. Our example task is to “Display the first ten lines of alphabetically arranged contents of the file PROP1 on the screen,” and the correct solution is “sort PROP1|head.”

Idealized Expert Example. Following the previously described binding process, UNICOM constructs a task specific network by interrelating the knowledge contained in the simulated user’s knowledge base. Abbreviated examples of general knowledge and plan element propositions related to the example task are shown in Tables 3 and 4. Once construction is complete, the task network is integrated, and UNICOM searches the final activation vector for the most activated plan element. Table 5 provides an abbreviated depiction of the most activated plan elements for this example task.

Looking at the Table, the most activated plan element is “head.” During the construction phase, this particular copy of the head plan was bound to the “redirected sorted contents of PROP1,” as shown in Table 4. The plan has a precondition that these contents exist, they do not, and the plan cannot be fired. UNICOM then considers the next most activated plan element, “sort.” This copy of the sort plan was bound to the file PROP1

TABLE 3
Example General Knowledge for “sort PROP1 | head”

P10	(KNOW REDIRECT INPUT TO OUTPUT FROM)
P11	(KNOW PIPE REDIRECT FROM COMMAND TO COMMAND)
P12	(KNOW SORT ARRANGE FILE^PROP1 ALPHABETICALLY ON^SCREEN)
P13	(KNOW HEAD DISPLAY FIRST^TEN^LINES FILE^PROP1 ON^SCREEN)
P14	(KNOW REDIRECT FROM SORT TO COMMAND)
P15	(KNOW REDIRECT FROM COMMAND TO HEAD)

during construction, and its three preconditions (see Table 4) are satisfied. This sort plan element is fired, and its outcome proposition is added to the in-the-world knowledge (KNOW ALPHABETICALLY^ARRANGED FILE^PROP1 ON^SCREEN).

The modified knowledge base is constructed and integrated in a second cycle. Table 5 depicts the resulting list of most activated plan elements. Following this second cycle, the “sort” plan element is no longer one of the most activated plan elements. It is inhibited because its outcome already exists in the world. The most activated plan element is again “head,” but the redirected sorted contents of file PROP1 still do not exist, and the plan does not fire. The next most activated plan element is “use pipe,” and it requires I/O syntax knowledge of the pipe symbol, conceptual knowledge of input and output redirection between commands, and that the modified contents of the file to which it is bound exist on the screen (see Table 4). This last precondition checks to see if output from another utility is available to be redirected. The three preconditions are met, the pipe plan fires, and

TABLE 4
Plan Elements Fired to Correctly Produce “sort PROP1 | head”

Sort plan element	
Name	(DO ARRANGE FILE^PROP1 ALPHABETICALLY)
Preconditions	(KNOW FILE^PROP1)
	(KNOW SORT ARRANGE FILE^PROP1 ALPHABETICALLY)
Outcome	(KNOW ALPHABETICALLY^ARRANGED FILE^PROP1 ON^SCREEN)
Pipe plan element	
Name	(USE PIPE REDIRECT ALPHABETICALLY^ARRANGED FILE^PROP1 TO COMMAND)
Preconditions	(KNOW ALPHABETICALLY^ARRANGED FILE^PROP1 ON^SCREEN)
	(KNOW PIPE REDIRECT FROM COMMAND TO COMMAND)
	(KNOW SORT REDIRECT FROM SORT TO COMMAND)
Outcome	(KNOW REDIRECT ALPHABETICALLY^ARRANGED FILE^PROP1)
Head plan element	
Name	(DO DISPLAY FIRST^TEN^LINES ALPHABETICALLY^ARRANGED FILE^PROP1)
Preconditions	(KNOW ALPHABETICALLY^ARRANGED FILE^PROP1 ON^SCREEN)
	(KNOW REDIRECT ALPHABETICALLY^ARRANGED FILE^PROP1)
	(KNOW HEAD DISPLAY FIRST^TEN^LINES FILE^NAME)
Outcome	(KNOW FIRST^TEN^LINES ALPHABETICALLY^ARRANGED FILE^PROP1 ON^SCREEN)

TABLE 5
Abbreviated Example of UNICOM Plan Selection Sequences in the Composite "sort PROP1|head" Using an Idealized Expert Knowledge Base

Cycle number	Activation	Abbreviated plan element names	Commands or I/O symbols executed	Preconditions met?	Outcomes added to world knowledge
1	2.5	Display first ten lines of alphabetically arranged file PROP1	head	No	Know alphabetically arranged file PROP1 on screen
	1.8	Arrange file PROP1 alphabetically	sort	Yes	
	0.9	Use pipe redirect alphabetically arranged file PROP1 to command	pipe	No	
2	2.6	Display first ten lines of alphabetically arranged file PROP1	head	No	Know redirect alphabetically arranged file PROP1 on screen
	1.2	Use pipe redirect alphabetically arranged file PROP1 to command	pipe	Yes	
3	2.8	Display first ten lines of alphabetically arranged file PROP1	head	Yes	Know first ten lines of alphabetically arranged file PROP1 on screen

its outcome propositions include the redirected alphabetically arranged contents of file PROP1. This along with a trace that a piped symbol has been used is added to the model's in-the-world knowledge when the plan is fired (see Table 4).

This modified knowledge base is used in a third C-I cycle, and the result is "head" as the most activated plan element. This time, the precondition that the redirected alphabetically arranged contents of file PROP1 exist is satisfied, the plan is fired, and its outcome proposition(s) are added to the world. Because the outcome of the "head" plan element matches the OUTCOME proposition that exists in the world, UNICOM considers the task completed.

Once a task is correctly completed, the task description is deleted along with any plan outcome propositions that exist in the world. If knowledge was permanently added to the knowledge base, the modified knowledge base is retained to simulate the user's performance on the next production task. This procedure is repeated for the entire set of 21 problems for each of the 22 users modeled.

Simplified Novice Example. Performance was error free in the example given above. In reality, each of the 22 UNIX users in our empirical prompting study made errors and received prompts. We now describe UNICOM processing the same example problem ("sort PROP1|head") using a simplified novice knowledge base. The example will clarify the procedures used to temporarily add prompts and permanently add knowledge to a user knowledge base.

Unlike the idealized expert, this hypothetical novice lacks knowledge. The idealized

TABLE 6
Plan Elements Incorrectly Fired by UNICOM Using a Simplified Novice Knowledge Base While Trying to Produce the Composite "sort PROP1|head"

Tail plan element	
Name	(DO DISPLAY LAST^TEN^LINES ALPHABETICALLY^ARRANGED FILE^PROP1)
Preconditions	(KNOW ALPHABETICALLY^ARRANGED FILE^PROP1 ON^SCREEN)
	(KNOW REDIRECT ALPHABETICALLY^ARRANGED FILE^PROP1)
	(KNOW TAIL DISPLAY LAST^TEN^LINES FILE^NAME)
Outcome	(KNOW LAST^TEN^LINES ALPHABETICALLY^ARRANGED FILE^PROP1 ON^SCREEN)
Filter2 plan element	
Name	(USE FILTER2 REDIRECT ALPHABETICALLY^ARRANGED FILE^PROP1\TO COMMAND)
Preconditions	(KNOW ALPHABETICALLY^ARRANGED FILE^PROP1 ON^SCREEN)
	(KNOW FILTER2 REDIRECT FROM COMMAND TO COMMAND)
	(KNOW SORT REDIRECT FROM SORT TO COMMAND)
Outcome	(KNOW REDIRECT ALPHABETICALLY^ARRANGED FILE^PROP1)

expert knowledge base was lesioned to represent knowledge deficiencies and erroneous knowledge. Specifically, command syntax, command redirection, and procedural knowledge of the "head" command, and I/O syntax knowledge of the "pipe" (|) symbol were deleted. Incorrect I/O syntax knowledge that "filter2" (>) redirects input and output between commands was added to the knowledge base (the symbol can only be used to redirect output from a command into a file). The incorrect knowledge is represented in general knowledge and in an executable plan element shown in Table 6.

UNICOM again activates the plan elements associated with the goal, as shown in Table 7. The most activated plan element is "tail," which lists the last ten lines of a file. This plan element's arguments overlap with arguments contained in the desired goal statement (see Table 6), which facilitates its activation. The particular "tail" command activated is bound to the sorted contents of PROP1 that do not exist. As a result, the plan element cannot fire. The next most activated plan element is "sort," bound to the file PROP1. The preconditions of this plan element are satisfied, and its outcome proposition is added to the world knowledge (see Table 4).

Following the next Construction-Integration (C-I) cycle, "filter2" (>) bound to the sorted contents of PROP1 is the most activated plan element. The modeled user knows the "filter2" symbol, and possesses incorrect I/O syntax knowledge that the "filter2" redirects input and output between commands. The preconditions are satisfied, the plan element fires, and the outcome proposition is added to the world knowledge (see Table 6).

The model considers the "tail" plan element for firing following the next C-I cycle. The redirected sorted contents of PROP1 now exist, and the plan is fired. The planning process is complete (see section B.1 in Appendix B for "stopping" rules), and UNICOM determines that there is a mismatch between the final outcome and desired outcome propositions in the world. This means the planned sequence will not correctly produce the desired task, and that a prompt must be given to the modeled user.

Table 8 shows the propositions representing the first four prompts of this problem. The

TABLE 7
Abbreviated Example of UNICOM Plan Selection Sequences at Prompts 0, 2, and 4 When Using a Simplified Novice Knowledge Base Produce the Composite "sort PROP1|head"

Prompt provided	Cycle no.	Activation	Abbreviated plan element names	Commands or I/O symbols executed	Preconditions met?	Outcomes added to world knowledge
0	1	2.8	Arrange file PROP1 alphabetically	sort	Yes	Know alphabetically arranged file PROP1 on screen
		1.9	Use filter2 redirect alphabetically arranged file PROP1 to command	filter2	No	
		0.8	Display last ten lines of alphabetically arranged file PROP1	tail	No	
	2	2.1	Use filter2 redirect alphabetically arranged file PROP1 to command	filter2	Yes	Know redirect alphabetically arranged file PROP1 on screen
		1.2	Display last ten lines of alphabetically arranged file PROP1	tail	No	
	3	1.4	Display last ten lines of alphabetically arranged file PROP1	tail	Yes	Know last ten lines of alphabetically arranged file PROP1 on screen
2	1	2.6	Display first ten lines of alphabetically arranged file PROP1	head	No	Know alphabetically arranged file PROP1 on screen
		2.2	Arrange file PROP1 alphabetically	sort	Yes	
		1.4	Use filter2 redirect alphabetically arranged file PROP1 to command	filter2	No	
	2	2.6	Display first ten lines of alphabetically arranged file PROP1	head	No	Know redirect alphabetically arranged file PROP1 on screen
		1.8	Use filter2 redirect alphabetically arranged file PROP1 to command	filter2	Yes	

(table continues)

TABLE 7
Continued

Prompt provided	Cycle no.	Activation	Abbreviated plan element names	Commands or I/O symbols executed	Preconditions met?	Outcomes added to world knowledge
	3	2.7	Display first ten lines of alphabetically arranged file PROP1	head	Yes	Know first ten lines of alphabetically arranged file PROP1 on screen
4	1	2.7	Display first ten lines of alphabetically arranged file PROP1	head	No	Know alphabetically arranged file PROP1 on screen
		2.1	Arrange file PROP1 alphabetically	sort	Yes	
		1.6	Use pipe redirect alphabetically arranged file PROP1 to command	pipe	No	
	2	2.8	Display first ten lines of alphabetically arranged file PROP1	head	No	Know redirect alphabetically arranged
		2.0	Use pipe redirect alphabetically arranged file PROP1 to command	pipe	Yes	File PROP1 on screen
	3	2.9	Display first ten lines of alphabetically arranged file PROP1	head	Yes	Know first ten lines of alphabetically arranged file PROP1 on screen

propositions corresponding to Prompt 1 are added to in-the-world knowledge. Prompt 1 does not include knowledge that the modeled user is lacking, nor does it correct erroneous knowledge. As a result, the model repeats the aforementioned planning sequence, detects

TABLE 8
Example Prompt Propositions Used by UNICOM Using a Simplified Novice Knowledge Base to Produce the Composite "sort PROP1|head"

Prompt 1	(KNOW ARRANGE FILE^NAME ALPHABETICALLY ON^SCREEN)
	(KNOW DISPLAY FIRST^TEN^LINES FILE^NAME ON^SCREEN)
Prompt 2	(KNOW SORT ARRANGE FILE^NAME ALPHABETICALLY ON^SCREEN)
	(KNOW HEAD DISPLAY FIRST^TEN^LINES FILE^NAME ON^SCREEN)
Prompt 3	(KNOW REDIRECT INPUT TO OUTPUT FROM)
	(KNOW REDIRECT FROM COMMAND TO COMMAND)
Prompt 4	(KNOW PIPE REDIRECT FROM COMMAND TO COMMAND)

the mismatch between the final and the desired outcome, and then adds propositions that represent the next prompt to the model’s world knowledge. At this point the propositions from Prompt 1 are treated as “previous” prompt propositions, and Prompt 2 propositions are “current.”

Prompt 2 provides command syntax knowledge for the “sort” and “head” commands (see Table 8). The corresponding prompt propositions are added to the in-the-world knowledge, making the number of prompted propositions in the world equal four, the maximum capacity. The “head” plan element is temporarily added to plan element knowledge. A C-I cycle is performed, and the most activated plan elements are shown in Table 7.

The first cycle fires the “sort” plan element, the next C-I cycle fires the erroneous “filter2” plan element, and on the third cycle the “head” plan is fired. Because the head plan is fired, it is permanently added to the knowledge base, even though the entire plan sequence will not produce the desired command. At this point, the propositions from Prompt 2 become “previous,” joining the ranks of the Prompt 1 propositions. Because the total number of “previous” prompt propositions is four, none are deleted.

The propositions for Prompt 3 are given (see Table 8). They do not provide new information because the modeled user already has conceptual knowledge of I/O redirection. The relative activation of the most activated plan elements remains the same as for the previous planning sequence, and the incorrect sequence is produced.

Before the addition of Prompt 4 propositions to in the world knowledge, Prompt 3 propositions are classified as “previous,” and the total number of such propositions equals six, two greater than capacity. Thus UNICOM retains only the four most activated “previous” propositions, and then adds the Prompt 4 proposition(s) as “current.”

Prompt 4 provides I/O syntax knowledge about “pipe,” and includes the addition of a temporary correct “pipe” plan element to plan knowledge. This time, the model executes the correct sequence of plan elements, and the task is completed. In so doing, UNICOM uses the temporary “pipe” plan, and it is permanently added to the knowledge base along with general knowledge about the “pipe” symbol. In addition, the erroneous knowledge of “filter2” is deleted. This modified knowledge base is used to perform the next task.

Results and Discussion

The previous text described the motivation for modeling UNIX users in a training study, and detailed our simulation procedures. We now turn to the quantitative and qualitative analyses of model command production performance as a function of prompt. We used the same scoring procedures for the modeled users that were used in the previously summarized Doane et al. (1992) study for actual users. We first describe the percentage correct by prompt data, and then provide quantitative measurements of the match between models and humans using RMSD values. This structure is repeated for knowledge scoring and for error analyses.

TABLE 9
ANOVA Results for Group UNIX User and User Model Percent Correct Productions

	Modeled users				UNIX users			
Source	df	F	MSE	*p < .05	df	F	MSE	*p < .05
Expertise (E)	(2, 19)	3.9	0.13	*	(2, 19)	4.8	0.14	*
Prompt (P)	(9, 171)	36.8	0.01	*	(9, 171)	45.8	0.01	*
New knowledge (N)	(2, 38)	14.7	0.09	*	(2, 38)	10.4	0.09	*
E × P	(18, 171)	6.8	0.01	*	(18, 171)	7.7	0.01	*
E × N	(4, 38)	3.6	0.09	*	(4, 38)	2.8	0.09	*

Analyses of Correct Productions

UNICOM Performance. Each sequence of plan elements combined to create a plan were scored for correctness, using the same rules applied to actual user performance. The right side of Figure 2 shows the cumulative percentage correct performance as a function of prompt for the modeled users in each expertise group. An ANOVA was performed using UNICOM percentage correct performance as a function of Prompt, Expertise, and Percent new knowledge, and the results are summarized in Table 9. Looking at Figure 2 and the results in Table 11, it is clear that there is a significant effect of expertise, with expert users outperforming the less expert groups.

The ANOVA results also reflect the differential amounts of influence prompts have on percentage correct performance (see Table 9). For the 60–100% new knowledge problems (Figure 2b), changes in percentage correct performance from Prompt 1 to Prompt 2, Prompt 3 to Prompt 4, and Prompt 5 to Prompt 6 are largest for modeled novices and intermediates, suggesting that command syntax, I/O syntax, and command redirection information, respectively, provided significant production assistance. The analogous changes between the remaining prompts are minimal, suggesting that these prompts provide little or no assistance.

Comparison of UNICOM and UNIX User Group Percentage Correct Performance. Table 9 also summarizes the significance of ANOVA results obtained by Doane et al. (1992) for corresponding UNIX user performance. Overall, the results are comparable, with the model and actual group analyses resulting in analogous main and interaction effects of group, prompt, and percentage new knowledge.

One exception to the comparability not reflected in the ANOVA results, is apparent in Figure 2. UNICOM over predicts UNIX expert user performance. UNICOM experts obtained nearly 100% correct performance at Prompt 0 regardless of percentage new knowledge required, and this is not true for the actual UNIX experts. The actual experts displayed nearly all of the knowledge required to successfully produce each of the twenty-one composites before prompting, and this was reflected in their corresponding individual knowledge base. However, many of these experts made errors on initial problems, as shown in Figure 2. Even when the actual experts possessed the complete set of knowledge to produce a composite, they sometimes solved a problem in their own way by using more than the minimum possible commands, sometimes forgot part of the

TABLE 10
Mean RMSD Values for Percent Correct Performance for UNICOM Modeled Users and UNIX Users as a Function of Expertise and Percent New Knowledge Problems

New knowledge	Novice		Intermediate		Expert	
	M	*SD*	*M*	*SD*	*M*	*SD*
60–100%	0.22	0.18	0.20	0.22	0.20	0.21
1–59%	0.08	0.18	0.09	0.16	0.07	0.18
0%	0.06	0.15	0.03	0.09	0.01	0.04

instruction and dropped a command, or entered a typographical error. At times they would explicitly state that they were “testing” our experimental interface. The first few prompts helped them realize that they had made these kinds of errors. However, the simulation did not have knowledge of typos, and did not selectively ignore parts of the instructions (unless they received no activation), and so forth Thus UNICOM cannot account for idiosyncratic errors, and as a result provides an inflated estimate of actual user performance.

Fit between UNICOM and UNIX Individual User Percentage Correct Performance. To quantify the fit between the empirical and simulation data, root-mean-squared deviations (RMSDs) were calculated based on the number of prompts required by the actual and corresponding modeled user to produce the correct composite on each of the 60–100%, 1–59%, and 0% new knowledge problems. Table 10 shows the resulting mean RMSD values, which range from 0.06 to 0.22. The fit between the empirical and simulation data increased as the amount of new knowledge the problem required for solution decreased, though this is mainly due to a ceiling effect (see Figure 2e). Overall, the model showed a high ability to predict how many prompts actual users required to produce a correct composite. We use the term predict because the knowledge base used in each simulation was based on a subset of the performance data, knowledge displayed before prompting. Once simulations began, rules outlined in Appendix B were applied in an automated fashion by UNICOM.

Knowledge Analyses

Although results thus far suggest a high degree of match between modeled and actual user percentage correct performance, it does not tell us how well UNICOM simulates user learning patterns. That is, how well does UNICOM predict the knowledge displayed in each incorrect production attempt? To examine this, the knowledge scoring procedures devised by Doane et al. (1992) were used to calculate corresponding scores for UNICOM modeled user performance. These procedures were described in previous sections of this manuscript. To summarize, the amount of command syntax, I/O syntax, I/O conceptual and command redirection knowledge displayed in each production attempt was scored as a function of prompt for each simulated user.

TABLE 11
ANOVA Results for Group UNIX User and User Model Knowledge Scores

	Modeled users				UNIX users
Source	*df*	*F*	*MSE*	**p* < .05	**p* < .05
Expertise (E)	(2, 19)	4.3	0.19	*	*
Prompt (P)	(9, 171)	36.3	0.02	*	*
New knowledge (N)	(2, 38)	15.1	0.14	*	*
Knowledge type (K)	(3, 57)	3.0	0.01	*	*
E × P	(18, 171)	6.9	0.02	*	*
E × N	(4, 38)	4.0	0.14	*	*
P × K	(27, 513)	5.6	0.01	*	*

Note. Corresponding *df* and *F* for UNIX users are reported in Doane et al. (1992).

UNICOM Performance. The right side of Figure 3 shows the simulated group knowledge scores as a function of prompt for UNICOM novice, intermediate, and experts. The left side of the figure shows the corresponding UNIX user data. In the figure, the arrow signifies the prompt that first provided the knowledge being scored. For example, Figures 3a and 3b show command syntax scores, and the arrow signifies that Prompt 2 provided command syntax knowledge.

To determine if the amount of relevant knowledge displayed by UNICOM varies as a function of expertise, prompt and knowledge type, an ANOVA was conducted with expertise (expert, intermediate, and novice) as between-participant variables, and knowledge type (command syntax, I/O syntax, I/O conceptual and command redirection), percent new knowledge and prompt as within-participant variables. The results are detailed in Table 11. To summarize, the analysis resulted in a main effect of expertise. Experts yielded higher knowledge scores than novices and intermediates across all percentage new knowledge problem groups. There was also a main effect of knowledge type, prompt, and a prompt x expertise interaction. Knowledge scores increased in response to prompt presentation, though the amount of improvement differed for experts.

Prompts had a differential influence on each of the four types of knowledge, as evidenced by a knowledge type x prompt interaction (see Table 11). The UNICOM means shown on the right side of Figure 3 suggest that for novice and intermediates, presentation of Prompt 2 improved command syntax knowledge, Prompt 4 improved I/O syntax knowledge, and Prompt 6 improved command redirection knowledge. Prompt 3, which stated that redirection exists as a concept did not impact I/O conceptual knowledge. As previously stated, this finding was expected based on previous research regarding the efficacy of abstract versus concrete instructions (cf., Holyoak, 1991). Overall, prompts seem to aid the intended knowledge, as evidenced in improved knowledge scores following presentation.

Comparison of UNICOM and UNIX User Group Knowledge Scores. Comparable knowledge scores for UNICOM modeled user and UNIX user group scores are shown in Figure 3. The results show similar effects of expertise, prompt, and knowledge type. This match is supported by the comparable significance of ANOVA results depicted in Table 11.

TABLE 12
Mean RMSD Values for Knowledge Scores for UNICOM Modeled Users and UNIX Users as a Function of Expertise and Percent New Knowledge Problems

	Novice		Intermediate		Expert	
New knowledge	*M*	*SD*	*M*	*SD*	*M*	*SD*
60–100%	0.32	0.18	0.22	0.16	0.12	0.14
Command syntax	0.28	0.21	0.15	0.14	0.04	0.08
I/O syntax	0.30	0.24	0.22	0.24	0.17	0.27
I/O redirect	0.34	0.22	0.25	0.23	0.10	0.15
Command redirect	0.34	0.21	0.26	0.24	0.15	0.22
1–59%	0.11	0.19	0.12	0.19	0.04	0.10
0%	0.03	0.07	0.02	0.05	0.02	0.06

Fit between UNICOM and UNIX User Knowledge Scores. RMSD values were calculated to estimate the fit between UNICOM and corresponding UNIX user knowledge scores. The RMSDs are based on UNICOM modeled user and UNIX user knowledge scores as a function of prompt. As an example, consider performance by one user on one composite problem, where the user solved the problem following presentation of Prompt 4 and the corresponding UNICOM modeled user solved the identical problem at Prompt 3. Knowledge scores were calculated for UNICOM and UNIX user productions at Prompts 0 (the first attempt before prompting) through 4. This resulted in four percentage scores for each type of knowledge displayed in each of the five attempted composites. The RMSD values, then, were based on each of the four types of knowledge scores displayed at each prompt by UNICOM and by UNIX users. This procedure was repeated for each problem, and for each individual to obtain individual and group mean values.

The number of prompts included in analyses was based on the maximum number of prompts used by the UNIX user or by the UNICOM model. For example, if UNICOM produced a correct composite following presentation of Prompt 6 and the UNIX user solved the problem at Prompt 5, then the RMSD analysis for that problem and that individual was calculated for 7 attempts, from Prompt 0 to Prompt 6. This procedure was used to make sure we did not inflate the goodness of fit data by including knowledge scores of 100% after the model and the user had solved a given problem.

Table 12 shows the resulting mean RMSD values, which range from 0.02 to 0.12 for the 0% and 1 to 59% new knowledge problems, and 0.04 to 0.34 for the 60 to 100% new knowledge problems. The average fit between UNIX user and UNICOM modeled user knowledge scores was greater for experts and intermediates than for novices. The fit was greater for problems that required 0% and 1 to 59% new knowledge than for the 60 to 100% new knowledge problems. This is due to the attenuated variability of ceiling level knowledge scores for experts and intermediates, particularly for problems requiring less new knowledge.

The values for this and the other knowledge analyses for the 0% and 1 to 59% new knowledge problems are comparable to those for the percentage correct data, and suggest a strong fit between UNICOM and actual user performance. The values for 60 to 100%

TABLE 13
ANOVA Results for Group UNIX User and User Model Deletion and Substitution Errors

	Modeled users				UNIX users
Source	*df*	*F*	*MSE*	**p* < .05	**p* < .05
Deletion errors					
Expertise	(2, 19)	3.4	0.86	*	*
Prompt	(9, 171)	27.5	0.09	*	*
New knowledge	(2, 38)	12.9	0.60	*	*
Substitution errors					
Expertise	(2, 19)	3.1	0.01	*	
Prompt	(9, 171)	7.8	0.01	*	*
New knowledge	(2, 38)	12.6	0.01	*	*

Note. Corresponding *df* and *F* for UNIX users are reported in Doane et al. (1992).

new knowledge problems are higher, with UNICOM accounting for 68%, 78%, and 88% of the variance for novices, intermediates, and experts, respectively.

Thirty-two percent of the novice knowledge scores are not accounted for in 60 to 100% new knowledge problems. To determine why the RMSD values were this high (32%), we compared the errors made by UNICOM to those made by UNIX users.

Error Analyses

UNICOM Performance. Looking at the UNICOM simulated user results in the right side of Figure 4, UNICOM novices and intermediates seem to make more deletion and substitution errors than do UNICOM experts. The number of errors seems to decrease as more prompts are provided.

To determine whether deletion and substitution errors are influenced by expertise and prompt, separate ANOVAs were conducted on the UNICOM modeled user deletion and substitution errors, each analysis using expertise level (expert, intermediate, novice) as a between-participants variable, and new knowledge and prompt as within-participant variables. The results are shown in Table 13. To summarize, there were main effects of Prompt, Expertise, and Percent New Knowledge on deletion and substitution errors for both UNICOM modeled users and actual UNIX users.

As suggested by Figure 4b, UNICOM novices and intermediates make more deletion errors than do experts. The deletion errors decrease as novices and intermediates receive more prompts, showing the greatest drop following presentation of the command syntax prompt (Prompt 2). As suggested by Figure 4 days, UNICOM substitution errors decrease after presentation of the I/O syntax knowledge prompt (Prompt 4) for novices and intermediates.

Table 13 shows comparable ANOVA results for UNIX users, suggesting that the UNICOM group results show effects of Prompt and Percent New Knowledge similar to those obtained for UNIX users. However, the significant effect of expertise observed for UNICOM modeled users was not obtained by actual UNIX users. One possible reason for

TABLE 14
Mean RMSD Values for Deletion and Substitution Errors for UNICOM Modeled Users and UNIX Users as a Function of Expertise and Percent New Knowledge Problems

	Novice		Intermediate		Expert	
New knowledge	*M*	*SD*	*M*	*SD*	*M*	*SD*
Deletion errors						
60–100%	0.21	0.15	0.14	0.15	0.05	0.08
1–59%	0.07	0.13	0.08	0.14	0.02	0.06
0%	0.01	0.03	0.00	0.02	0.00	0.00
Substitution errors						
60–100%	0.19	0.12	0.18	0.15	0.08	0.09
1–59%	0.03	0.07	0.05	0.10	0.03	0.07
0%	0.02	0.06	0.03	0.07	0.03	0.08

this lack of comparability is that UNICOM over predicted expert performance, resulting in an inflated expertise effect. Overall the number of substitution errors predicted by UNICOM are lower than that for actual UNIX users. This underprediction is particularly true for the expert group.

As previously mentioned, deletion and substitution errors are suggestive of working memory limitations (e.g., Anderson & Jeffries, 1985). Recall that the previous analyses on knowledge scores resulted in a 0.32 RMSD value for Novices for 60–100% new knowledge problems. The present analyses on deletion and substitution errors suggests that UNICOM's inadequate representation of working memory may be the locus of this relatively large RMSD. That is, on problems requiring significant amounts of new knowledge, Novice knowledge scores may be constrained by working memory limitations not accounted for by UNICOM.

Fit between UNICOM and UNIX User Production Errors. To quantify the fit between UNICOM modeled user and UNIX user deletion and substitution errors, RMSD values were calculated. Table 14 shows means of the resulting RMSD values for the deletion and substitution error analyses respectively. The results must be viewed with caution, as the floor-effects caused by UNICOM's over predictions necessarily inflate the RMSD values. This is particularly true for experts, and for problems requiring less than 60% new knowledge.

Overall the error analyses suggest that UNICOM's assumptions regarding working memory need to be reconsidered. The sole memory limitation explicitly represented was the limitation on the number of previous prompt propositions in memory. Clearly additional limitations exist for actual users.

Learning Analyses

As previously mentioned, prompted knowledge was considered "learned" if it was used in a composite production attempt following explicit prompting. This was implemented in UNICOM by permanently adding the prompted knowledge to a modeled individual's knowledge base. As a result, learned knowledge was available for use should it be sufficiently activated. [This assumption is similar to Anderson's (1993) view that learned knowledge does not disappear, it simply loses activation.]

We tested how well this assumption matched user performance by calculating the number of uses of learned knowledge when it was required to correctly produce a composite. Data points included correct and incorrect productions. For example, if a user is prompted with the command syntax knowledge of "sort" and then uses "sort" in five subsequent attempts to produce composites that require this command, then their percentage of "sort" knowledge use would be 5/5, or 100%. This calculation was performed for each individual and for each component of UNIX knowledge prompted in the experiment.

The results suggest that our assumption fits quite well with the user performance data. In UNICOM, use following learning is 100%. For the users, the overall percentage of knowledge use following acquisition was 96%. The average percentages as a function of expertise were 95%, 96%, and 98% for novices, intermediates, and experts, respectively. This analysis provides some confirmation that our learning assumptions are plausible. In addition, it provides further indication that UNICOM's knowledge assumptions are not the cause of UNICOM's overprediction of user performance.

Summary of Results

UNICOM matches actual novice and intermediate performance quite well, although expert performance is grossly over predicted. It is apparent that the knowledge-based component of UNICOM can accurately predict correct performance and many aspects of incorrect performance quality as measured by knowledge scores. However, more work needs to be done to understand how working memory limitations should be incorporated further into this comprehension-based model of learning.

V. GENERAL DISCUSSION

We have shown how a comprehension-based theory of learning accounts for a significant amount of user performance when learning from technical instructions. Using a model based on the construction-integration theory of comprehension, we developed individual knowledge bases based on a small subset of user performance data. The initial knowledge bases were used by UNICOM to "participate" in the UNIX prompting study, and allowed us to predict significant aspects of user performance.

Methodologically, what we have done is applied rigorous training and testing methods more commonly used by connectionists to develop a predictive and descriptive knowledge-based model. Modeling twenty-two individuals solving twenty-one composite problems in a training environment is certainly a time consuming effort. However, this method allows us to test the qualitative and quantitative fit of the model to the actual performance data. That is, we can go beyond mere speculation that the model provides a "good description" of the data. UNICOM predicts many aspects of performance in a learning environment.

In addition, following the vision of Newell (1990), we are using a model that has been tested on a wide variety of tasks, and as such are developing an architecture of cognition. Further, our implementation satisfies many of the stringent criteria mentioned by Thagard (1989) for consistent architecture development. Specifically, we used the same parameters, relationships, and weights in each of our simulations, varying only the user knowledge base. And the variation in knowledge bases was carefully controlled and based on rigorous coding rules.

Theoretically, what we have done is to provide further evidence of the centrality of comprehension in cognition (e.g., Gernsbacher, 1990). We have accomplished this by using a comprehension-based model to simulate a complex problem solving and learning environment. In so doing, we have extended the theoretical premise of Kintsch's (1988) construction integration theory to account for how computer users learn to produce commands from instructions. This extends the theory to planning, and more importantly, suggests that the contextually constrained activation of knowledge central to the comprehension-based approach is more than descriptive, it is predictive.

This work has implications for computer-aided tutoring as well. Tutoring systems already use models to assess the untutored expertise of a student (see, e.g., Anderson, 1988; VanLehn, 1988). Using the present techniques, instructions chosen by the tutor for presentation could be optimized for a particular student model in a context-sensitive manner. If, for example, a student model had to choose among three instructional options at a given point during training, it could use our methods to determine which of the three options would receive the highest activation given the current assessment of student knowledge.

Our present efforts include extending this architecture to "ADAPT," a model of novice, intermediate, and expert piloting performance. Doane and Sohn (1997) have modeled pilot eye-scans and airplane control movements in goal-based flight segments, obtaining very high measures of fit between modeled and actual data. This effort extends even further the centrality of comprehension in cognition.

Future efforts include developing an improved comprehension-based model of working memory. The present effort did not utilize all aspects of the long term memory retrieval component of Kintsch's (1988) theory. This was done because all instructions and prompts remained on the screen in the empirical UNIX study. However, it is clear that the assumption that instructions in plain view are activated, used, and remembered is clearly unwarranted. Our future efforts will incorporate the retrieval model described by Kintsch (1988) into our comprehension-based theory of learning.

The present work does not focus on differentiating UNICOM's architecture from that used by ACT-R and Soar, two major models of cognition. The three architectures share many attributes including the use of declarative and procedural knowledge. What distinguishes the three models is how the role of problem solving context is represented, and how it influences knowledge activation and use. In SOAR, episodic knowledge is used to represent actions, objects and events that are represented in the modeled agent's memory (e.g., Rosenbloom, Newell, & Laird, 1991). This knowledge influences the use of procedural and declarative knowledge by impacting the activation of knowledge based on the context of historical use. In ACT, the analogical processes used to map similarities between problem-solving situations simulates the interpretive use of knowledge in a new context.

In the present model, context is not represented as historical memory or governed by an analogical process. Rather, the influence of context is to constrain the spread of knowledge activation based on the configural properties of the current task situation using low-level associations. Certainly this results in a model that covers a more modest range of cognitive behaviors than those examined by SOAR and ACT-R researchers (e.g., VanLehn, 1991). However, the present rigorous test of predictive validity suggests that this simplistic approach to understanding adaptive planning has significant promise.

An important strength of the present model is that it has been applied to such a wide variety of cognitive phenomenon using very few assumptions and very little parameter fitting. One weakness is that greater parsimony in these terms has lead to less than perfect model fits to the human data. There is clearly a tradeoff between parameter fitting and parsimony. In this case, a relatively parsimonious model has provided reasonable fits to highly complex human computer interactions as well as to skill and knowledge acquisition data. However, the model failed to account for small subsets of the UNIX user data. Nevertheless, this weakness has had a notable advantage in terms of indicating necessary revisions to the model's assumptions, most importantly concerning the greater working memory limitations of novice users. In addition, this work highlights the importance and necessity of turning toward the building of predictive individual models of human performance, accounting for differences at the participant level, rather than simply describing for aggregate performance. We believe that further such efforts will be essential for a true understanding of human cognition.

Acknowledgments: The authors wish to thank Gary Bradshaw, Walter Kintsch, and Suzanne Mannes for their insightful comments on earlier versions of this work.

NOTES

1. These data were previously published in Doane et al. (1992), and are included here to facilitate later comparisons with modeling results.
2. The remaining percentage new knowledge problem groups showed similar but attenuated results, and will not be discussed further.
3. The full set of UNICOM UNIX knowledge is available at http://wildthing.psychology.msstate.edu/ and can also be obtained by contacting the first author.

REFERENCES

Anderson, D. (1989). *Artificial intelligence and intelligent systems*. New York: John Wiley & Sons.
Anderson, J. R. (1996). ACT: A simple theory of complex cognition. *American Psychologist, 51,* 355–365.
Anderson, J. R. (1993). *Rules of the mind*. Mahwah, NJ: Erlbaum.
Anderson, J. R. (1988). The expert module. In M. C. Polson, & J. J. Richardson (Eds.), *Foundations of intelligent tutoring systems* (pp. 21–54). Mahwah, NJ: Erlbaum.

Anderson, J. R., & Jeffries, R. (1985). Novice LISP errors: Undetected losses of information from working memory. *Human-Computer Interaction, 1,* 133–161.

Anderson, J. R. & Lebiere, C. (1998). *Atomic components of thought.* Mahwah, NJ: Erlbaum.

Broadbent, D. E. (1993). Comparison with human experiments. In D. E. Broadbent (Ed.), *The simulation of human intelligence* (pp. 198–217). Cambridge, MA: Blackwell.

Doane, S. M., Mannes, S. M., Kintsch, W., & Polson, P. G. (1992). Modeling user command production: A comprehension-based approach. *User Modeling and User Adapted Interaction, 2,* 249–285.

Doane, S. M., McNamara, D. S., Kintsch, W., Polson, P. G., & Clawson, D. (1992). Prompt comprehension in UNIX command production. *Memory and Cognition, 20*(4), 327–343.

Doane, S. M., Pellegrino, J. W., & Klatzky, R. L. (1990). Expertise in a computer operating system: Conceptualization and performance. *Human-Computer Interaction, 5,* 267–304.

Doane, S. M., & Sohn, Y. W. (2000). ADAPT: A predictive cognitive model of user visual attention and action planning. *User Modeling and User Adapted Interaction 10,* 1–45.

Doane, S. M., Sohn, Y. W., Adams, D., & McNamara, D. S. (1994). Learning from instruction: A comprehension-based approach. In *Proceedings of the 16th Annual Conference of the Cognitive Science Society* (pp. 254–259). Atlanta, GA: Erlbaum.

Dreyfus, H. L. (1996). Response to my critics. *Artificial Intelligence, 80,* 171–191.

Dreyfus, H. L. (1992). *What computer still can't do: A critique of artificial reason*. Cambridge, MA, MIT Press.

Ericsson, K. A., & Oliver, W. L. (1988). Methodology for laboratory research on thinking: Task selection, collection of observations, and data analysis. In R. J. Sternberg, & E. E. Smith (Eds.), *The psychology of human thought* (pp. 392–428). Cambridge, MA: Cambridge University Press.

Gernsbacher, M. A. (1990). *Language comprehension as structure building*. Hillsdale, NJ: Erlbaum.

Hammond, K. (1989). *Case-based planning*. London: Academic Press.

Holyoak, K. J. (1991). Symbolic connectionism: Toward third-generation theories of expertise. In K. A. Ericsson, & J. Smith (Eds.), *Toward a general theory of expertise* (pp. 301–336). Cambridge University Press.

Holyoak, K. J., & Thagard, P. (1989). Analogical mapping by constraint satisfaction. *Cognitive Science, 13,* 295–355.

Just, M. A., & Carpenter, P. A. (1992). A capacity theory of comprehension. *Psychological Review, 99,* 122–149.

Kintsch, W. (1988). The use of knowledge in discourse processing: A construction-integration model. *Psychological Review, 95,* 163–182.

Kintsch, W. (1994). The psychology of discourse processing. In M. A. Gernsbacher (Ed.), *Handbook of psycholinguistics* (pp. 721–739). San Diego: Academic Press.

Kintsch, W. (1998). *Comprehension: A paradigm of cognition.* New York: Cambridge University Press.

Kitajima, M., & Polson, P. G. (1995). A comprehension-based model of correct performance and errors in skilled, display-based, human-computer interaction. *International Journal of Human-Computer Studies, 43,* 65–99.

Lovett, M. C., & Anderson, J. R. (1996). History of success and current context in problem solving. *Cognitive Psychology, 31,* 168–217.

Mannes, S. M., & Doane, S. M. (1991). A hybrid model of script generation: Or getting the best of both worlds. *Connection Science, 3*(1), 61–87.

Mannes, S. M., & Kintsch, W. (1991). Routine computing tasks; Planning as understanding. *Cognitive Science, 15*(3), 305–342.

Miller, J. R., & Kintsch, W. (1980). Readability and recall of short prose passages: A theoretical analysis. *Journal of Experimental Psychology: Human Learning and Memory, 6,* 335–354.

Murdock, B. B. (1974). *Human memory: Theory and data*. Hillsdale, NJ: Erlbaum.

Newell, A. (1990). *Unified theories of cognition* (The 1987 William James Lectures). Cambridge, MA: Harvard University Press.

Recker, M. M., & Pirolli, P. (1995). Modeling individual differences in students' learning strategies. *The Journal of the Learning Sciences, 4,* 1–38.

Rosenbloom, P. S., Laird, J. E., Newell, A. (1991). Toward the knowledge level in SOAR: The role of architecture in the use of knowledge. In K. VanLehn (Ed.), *Architectures for intelligence* (pp. 75–112). Hillsdale, NJ: Erlbaum.

Rosenbloom, P. S., Laird, J. E., Newell, A., & McCarl, R. (1991). A preliminary analysis of the SOAR architecture as a basis for general intelligence. *Artificial Intelligence, 47,* 289–325.

Schmalhofer, F., & Tschaitschian, B. (1993). The acquisition of a procedure schema from text and experiences. *Proceedings of the 15th Annual Conference of the Cognitive Science Society* (pp. 883–888). Hillsdale, NJ: Erlbaum.

Sohn, Y. W., & Doane, S. M. (1997). Cognitive constraints on computer problem solving skills. *Journal of Experimental Psychology: Applied, 3,* 288–312.

Thagard, P. (1989). Explanatory coherence. *Brain and Behavioral Sciences, 12,* 435–467.

van Dijk, T. A., & Kintsch, W. (1983). *Strategies of discourse comprehension*. New York: Academic Press.

VanLehn, K. (1991). (Ed.) *Architectures for intelligence* (pp. 75–112). Hillsdale, NJ: Erlbaum.

VanLehn, K. (1988). Student modeling. In M. C. Polson, & J. J. Richardson (Eds.), *Foundations of intelligent tutoring systems* (pp. 55–76). Hillsdale, NJ: Erlbaum.

APPENDICES

TABLE OF CONTENTS

APPENDIX A: SCORING PROCEDURES FOR KNOWLEDGE AND PRODUCTION ERRORS

A.1. SCORING PROCEDURE FOR KNOWLEDGE

This section will describe the scoring procedure for knowledge displayed in the empirical and simulation data. The following is an example of how a problem would be scored for any given UNIX user and modeled user. The example provided is the first problem presented to all of the users and models (i.e., edit job.p<stat2|lpr). In the first section is

a list and description of all of the knowledge necessary to correctly complete the problem. The second section presents an example of how a novice user or model responded and how that response was scored.

Necessary Knowledge

UNIX knowledge can be classified into the following four categories: (1) Input/Output Redirection, (2) Input/Output Redirection Syntax, (3) Command Syntax, and (4) Command Redirection. The necessary knowledge for the example problem, edit job.p<stat2|lpr, is listed as follows:

1. Input/Output Redirection
   - know input output redirect
   - know redirect command input output
   - know redirect input to command from file
2. Input/Output Redirection Syntax
   - know pipe |
   - know pipe | redirect command input output
   - know filter1 <
   - know filter1 < redirect input to command from file
3. Command Syntax
   - know command edit
   - know command edit takes file argument
   - know command lpr
   - know command lpr requires file argument
4. Command Redirection
   - know redirect input to edit command from file
   - know redirect output from edit to command
   - know redirect input to lpr from command

This problem requires that the user know three general things about input/output redirection. First the user must know that it is possible to redirect input and output from commands and files; second, the user must know that it is possible to redirect from command to command, and third, from command to file. To implement the above knowledge the user must know the UNIX syntax. This particular problem requires knowledge of the pipe (i.e., |) specialty symbol, and the filter 1 specialty symbol (i.e., <). The user must also know what these symbols do. For example, it is necessary to know specifically that the pipe redirects command input/output and that the filter 1 symbol redirects input to a command from a file.

In addition to the conceptual and syntactical redirection knowledge described above, it is also necessary to have syntactical and redirection knowledge about the commands in the problem. For the example problem, it is necessary to know that the command edit can take a file argument and that the command lpr requires a file argument. This type of knowledge is classified as command syntax knowledge. In contrast, knowledge about command redirection is considered more conceptual in nature. For this problem, the user must know that it is possible to redirect input to the edit command from a file and to redirect output from the edit command to another command. The user must also know that it is possible to redirect input to lpr from a command, in this case the edit command.

Example Scoring of Displayed Knowledge

When a user correctly answers the example problem, that user is scored as displaying all of the knowledge listed above. It is also possible for a user to leave out certain commands, make errors on the necessary commands, or include knowledge that was not necessarily required to correctly answer the problem. In the following example a novice entered the following response (without any aid of prompts):

```
cat stat2|vi job.p|lpr
```

This response was scored as follows. In the first category, Input/Output Redirection, the user is showing knowledge that it is possible to redirect input and output and to redirect from command to command. No other knowledge is displayed in this category. Thus, in this category the user is showing two correct knowledge propositions that are both relevant to the problem. In the second category, Input/Output Redirection Syntax, the user is showing knowledge of the pipe specialty symbol (i.e., |), and what this symbol does. Therefore, in this category the user would have a score of two correct relevant knowledge propositions. In the third category, Command Syntax, the user displays knowledge of only two of the relevant knowledge necessary to the problem, that is knowledge of the command lpr and that this command requires a file argument. However, the user is also displaying knowledge of two other commands that were not necessary to the completion of this problem, that is the commands “cat” and “vi.” Thus the user would have a score for correct irrelevant knowledge displayed under this category. In the fourth category, Command Redirection, the user would have a score for correct relevant knowledge. This would be the knowledge that it is possible to redirect input to the command lpr from a command. The user would have a score for correct irrelevant knowledge in this category. This is the knowledge of redirection of output from the cat command, redirection of output from the vi command, and redirection of input to the vi command from a command.

A.2. SCORING PROCEDURE FOR PRODUCTION ERRORS

This section will describe the rules governing the syntactic scoring of the empirical and simulation data.

Components of a Command

There are three basic components that make up the correct responses to the UNIX tasks:

Utility—command names such as head, sort, lpr, nroff, and so forth

File—objects (files) that a command or redirection symbol are intended to act upon. The names of these files are given in the task query (e.g., job1.p).

Specialty—redirection symbols (|, >, <, ≫) that are used for redirection of input and output, and enables separate UNIX calls to be combined in a single series. The & (background) symbol enables the programmer to run another process whereas keeping the current process shell operable.

Finally, some commands allow the use of special options, which add a bit of flexibility to the use of those commands. These options are often referred to as flags. For example, the command wc (for word count) counts the number of words in a file, but can instead count the number of lines by combining wc with the "−l" flag.

Scoring of the Components

Each of the components described can be scored as follows.

Delete—if the component is missing and no component is entered to take its place.

Add—if a component that is not called for is given in the response and is not a substitution for another component, or that is repeated in the response.

Substitute—if a component is entered that seems to be taking place of a component that is called for. This includes both order errors ("sort|head" for "head|sort") and true substitutions ("alpha|head" for "sort|head"), where the non-UNIX command alpha was given in place of the correct sort.

Illegal use—if a component either causes the system to send an error message or causes the system to hang.

Scoring Rules

A set of rules for determining the syntactic scoring of the user's responses, along with some examples are as follows:
Notation:

T (target answer), R (user's response), S (score).

1. Substitutions are based on the component being substituted for, not on the substituting component.

T: sort file1 | head
R: sort −10 file1
S: substitute utility

The scoring inference here is that the −10 flag is intended to behave as head.

2. Use simplest inference to reduce scoring possibilities.

T: head file1 | sort
R: cat 10 file1 | sort
S: substitute utility (cat 10 for head)

This could be scored as an add utility (cat), and substitute utility (10 for head), but in this case, the simplest inference is that (cat 10) is the substitution for head.

3. To score order and substitution errors: Check for errors involving both position and simple substitution, and account for each discretion.

T: nroff -ms apon | head | sort
R: sort apon | head | nroff -ms
S: substitute 2 utilities

Since head is in the proper position among utilities, there are only two order errors.

T: nroff -ms apon | head | sort
R: cat 10 < apon | sort |nroff -ms
S: add specialty symbol (<), substitute 4 utilities (3 for order, 1 for substitution)

If we score "cat 10" as substituting for head, then we see that all 3 utilities are out of order.

4. To score flags: If a flag exists without the utility, it's a delete utility, but if the flag lacks the "-" sign, then infer that the user lacks knowledge of the use of the utility, and score as a substitution of flag for utility.

T: nroff -ms file1 > file2
R: -ms file1 > file2
S: delete utility (nroff), illegal specialty symbol (-ms won't work by itself)

In this case, if a user has previously displayed use and knowledge of flags, then infer a lack of knowledge of nroff.

T: nroff -ms file1 > file2
R: ms file1 > file2
S: substitute utility (ms for nroff), delete specialty symbol (-ms), illegal utility (ms)

If the instructions mentioned something about a -ms macro package, and the user has not previously demonstrated use and knowledge of flags, then infer that the user is guessing about the text-processing utility.

5. Pay attention to the context in which an item is used. For example, an apparent deletion may actually be just a substitution, where the user used one item to substitute for another, where both items are part of the correct answer.

T: nroff -ms apon | sort | head > anal.c
R: nroff -ms anal.c | head | sort
S: delete file (anal.c), delete specialty symbol (>), substitute file (anal.c for apon), substitute 2 utilities (order)

In this case, anal.c is being used as apon. Thus, the real anal.c output file is deleted, whereas the anal.c in the response substitutes for apon.

6. Similarly, do not assume that a symbol is correct simply because it is given as part of the response. Again, infer the user's intention regarding use of the symbol.

T: sort file1 | head > file2
R: cat file1 | sort > file2

S: delete utility (head), delete specialty symbol (| between sort and head), add utility (cat), add specialty symbol (| between cat and sort)

In this case, because head has been deleted, the pipe given is not the same pipe expected. It comes between a different pair of commands from that of the target.

T: sort file1 | head > file2
R: alpha file1 | head > file2
S: substitute utility (alpha for sort), illegal utility (alpha)

In this case, because it's apparent that alpha is a substitute for sort, then the pipe symbol | is correct in this context.

7. For scoring redirection: Given that the response is incorrect, analyze components of the response to determine legality of the redirection symbols.

T: sort file1 | head > file2
R: sort file1 > file2 | head
S: substitute 2 specialty symbols (order), illegal specialty symbol (cannot pipe from a file to a command)

The components are those collection of units that comprise an attempt at redirection:

sort file1 > file2 and file2 | head

8. Sometimes, a redirection symbol won't cause a break, but rather is ignored because the units it connects do not work in sequence with that symbol.

T: ls > empt1
R: ls | mv empt1
S: substitute utility (| for >), illegal utility (mv)

Clearly, the pipe symbol | substitutes for >, but ls does not pipe to mv. In this case, the ls and | combination is ignored, and only the mv empt1 is evaluated by UNIX, and mv is illegal because it needs a second filename.

APPENDIX B: SIMULATION RULES FOR EXECUTING INDIVIDUAL MODELS

B.1. SIMULATION STOPPING RULES

Following a C-I cycle, the program must evaluate whether it should continue. If no further progress can be made on the present attempt to produce a composite, then the program will stop and then another system will evaluate the correctness of the plan sequence. To automate the process of determining when it is time to stop, the model tested for the following conditions:

1. If redirection symbols are repeated (not necessarily identical, just in immediate sequence), then stop.

2. If the model does not have relevant knowledge (i.e., anything that could augment knowledge score) to solve the problem, then stop.
3. If the number of actions in the problem are used and no relevant knowledge exists, then stop. If the number of actions exceeded and relevant knowledge exists, then continue to run.
4. If some commands are repeated in a row, then stop. For example, (lpr file1) (lpr file2).
5. If the output of a command or a pipe is used more than once, then stop. For example, (ls) (pipe) (head) (*tail file^ls*).
6. If the output of pipe is not used as the input of the next command, then stop. For example, (<file1) (edit file2) (pipe) (*head file3*).

If any were true, then the model presented its attempt thus far to another system, and the system graded the production for correctness. If the production was correct, then the next problem was presented. If the production was incorrect, then the next prompt was presented to the model and the cycling began again with the appropriate traces in memory.

B.2. PERMANENT ADDITION OF PROMPT KNOWLEDGE AND COMMAND REDIRECTION KNOWLEDGE

1. Any prompt propositions which were preconditions for a plan element fired during the previous prompt are added to the model's general knowledge.
2. When an I/O syntax knowledge proposition (e.g., "know filter2 from^command^redirect to^file^redirect") is added, the corresponding i/o conceptual knowledge proposition (e.g., "from^command^redirect to^file^redirect") is also added to the model general knowledge if the model does not have the i/o conceptual knowledge in its general knowledge base.

B.3. PROMPT CARRYOVER

Decide which prompt propositions to keep from the knowledge file for the previous prompt in the knowledge file for the current prompt. Only 4 prompt propositions from the previous prompt's knowledge file are retained. Looking at the activation values of the prompt propositions from the last cycle of the previous prompt, choose the 4 most highly activated unique prompt propositions. The chosen propositions will be retained as in-the-world knowledge for the current knowledge file.

B.4. ADDITION OF PLAN ELEMENTS ASSOCIATED WITH THE PROMPT PROPOSITIONS

For Prompts 2 and 4, add new plans associated with the prompt propositions just added to the current knowledge file.

B.5. ADDITION OF COMMAND NAME AND NAME INTO KNOWLEDGE BASE. (IF THE MODEL DOES NOT HAVE KNOWLEDGE OF COMMAND NAME OR SYMBOL NAME).

1. At prompt 2, when command syntax knowledge is prompted, if the model does not have knowledge of command names, then add command name knowledge (e.g., “know edit”) to general knowledge base permanently.
2. At prompt 4, when I/O syntax or background symbol knowledge are prompted, if the model does not have knowledge of symbol names, then add redirection symbol knowledge (e.g., “know filter1”) or background symbol knowledge (e.g., “know symbol^background”) to general knowledge base permanently.

Rationale: When prompted, command names and symbol names are stored in LTM as a kind of declarative knowledge. Therefore, people can recognize the commands and symbols themselves later even though they do not know what the commands and the symbols do.

B.6. INFERRED COMMAND REDIRECTION KNOWLEDGE

If a new prompt proposition contains a command syntax fact which the model does not know, add to the prompt propositions the command redirection knowledge propositions as follows:

1. If the new command is “edit” and the model already knows the command “vi,” then add command redirection knowledge propositions for edit to the prompts which correspond to the command redirection knowledge the model already knows for vi. For example, for a modeled user that did not know edit but knew vi and also knew “know vi from^file^redirect,” the prompt proposition “know edit from^file^redirect” is added.
2. Rules for adding edit knowledge when either vi or filter1 plan are missing:
   a. If the model doesn’t know vi, then give credit for maximum redirection knowledge shown in other commands except ls and lpr when adding it at Prompt 2.
   b. If the model doesn’t have “know edit from^file^redirect” and filter1 plan in its knowledge base, then delete the two propositions that correspond to these preconditions from the edit plan before adding the plan at Prompt 2. When this modeled user is prompted at Prompt 4,
      i. add general knowledge fact “know edit from^file^redirect” and filter1 plan to knowledge base
      ii. add back the two previously deleted preconditions into the edit plan
   c. If the model doesn’t know vi and edit but does know filter1, then add the command redirection knowledge “know edit from^file^redirect” in the world when the model learns edit at Prompt 2 and leave edit plan alone.
3. For any other new command, add maximum redirection knowledge shown in other commands except ls, lpr, vi, and edit.

B.7. ADDITION OF COMMAND REDIRECTION KNOWLEDGE RELEVANT TO FILTER2 OR FILTER3 WHEN THE MODEL DOES NOT KNOW ANY COMMAND OTHER THAN EDIT, VI, LS, OR LPR

1. At Prompt 4, if the model has knowledge of either filter2 or filter3, then give full set of command redirection knowledge needed to solve the problem.
2. At Prompt 6 or 7, if the model has knowledge of neither filter2 nor filter3, give the command redirection knowledge prompted.
3. After Prompt 6 or 7, if the problem isn't solved, then just add knowledge that are prompted until it is solved at the last prompt.

B.8. DELETION OF INCORRECT I/O SYNTAX KNOWLEDGE AND PLAN ELEMENTS

Delete incorrect I/O syntax knowledge and incorrect plan elements from the knowledge file immediately following their first correct usage. For example, if a modeled user had incorrect I/O syntax knowledge that the symbol "*" would redirect input and output between commands, this knowledge would be deleted from the model once the correct symbol was prompted (i.e., "|") or when the correct symbol used to redirect command input and output.

B.9. INFERENCE OF "WC" DEFAULT KNOWLEDGE FROM "WC-C" KNOWLEDGE

1. For experts, add knowledge of wc default to the general knowledge after knowledge of wc-c is prompted.
2. For novices or intermediates, do not infer wc default knowledge.

Rationale. Problems with wc-c precede problems with wc default. UNIX user production data showed that 100% of the experts who did not initially know wc-c displayed knowledge of wc default before being prompted with this knowledge. However, a little more than half of the novices and intermediates (56% and 67%, respectively) who did not initially know wc-c displayed this knowledge before they were prompted. Therefore, experts are more likely to infer knowledge of wc default from knowledge of wc-c than novices and intermediates.

APPENDIX C: EQUATIONS FOR CONSTRUCTION AND INTEGRATION

C.1. CONSTRUCTION

At the construction process, the following equations define the strengths of relationships between propositions represented in the knowledge base.

1. Argument overlap

The strengths (W_{ij}, W_{ji}) of relationships for P_i, $P_j \in$ {world knowledge, general knowledge} are defined as follows:

For $i \neq j$,

$W_{ij} = W_{ji} = W_{overlap} \times$ (number of arguments overlapped),

where $W_{overlap} = 0.4$ (but 0.2 for P_i or $P_j \in$ {prompt proposition}).

For i=j,

$W_{ij} = 1.0$

The strengths (W_{ik}, W_{ki}) of relationships for the same set of P_i and $L_k \in$ {plan knowledge} are defined as follows:

$$W_{ik} = W_{ki} = W_{overlap} \times \text{(number of arguments overlapped)},$$

$$\text{where } W_{overlap} = 0.4 \text{ (but 0.2 for } P_i \text{ or } P_j \in \text{\{prompt proposition\})}.$$

2. Plan-world inhibition

If outcomes of L_k exist in the world knowledge, the strength W_{ki} of an asymmetric relationship for $L_k \in$ {plan knowledge} from $P_i \in$ {world knowledge} is defined as follows:

$$W_{ki} = W_{inhibition}, \text{ where } W_{inhibition} = -10.0$$

If P_j, an outcome of L_k, with the TRACE predicate exists in the world knowledge, the strength W_{kj} of an asymmetric relationship for $L_k \in$ {plan knowledge} from $P_j \in$ {world knowledge} is defined as follows:

$$W_{kj} = W_{overlap} \times \text{(number of arguments overlapped), where } W_{overlap} = -0.4$$

3. Interplan relationships

Asymmetric causal relationships for L_k, $L_l \in$ {plan knowledge} are defined as follows:

For $k \neq l$,

$$\text{if } L_k \text{ supports } L_l,\ W_{kl} = W_{excitation}, \text{ where } W_{excitation} = 0.7$$

$$\text{if } L_k \text{ inhibits } L_l,\ W_{kl} = W_{inhibition}, \text{ where } W_{inhibition} = -10.0$$

For $k = l$,

$$W_{kl} = 1.0$$

C.2. INTEGRATION

A set of nodes interconnected by the construction process is represented by:

$$(X_1, \ldots, X_i, \ldots X_M, Y_1, \ldots, Y_j, \ldots Y_N)$$

where X_is represent world knowledge and Y_js represent general and plan knowledge, and M and N are the number of nodes representing the respective knowledge. The pattern of activation after v-th iteration can be expressed by a vector,

$$\overline{A}^{(v)} \equiv (A_{X_1}, \ldots, A_{X_i}, \ldots, A_{X_M}, A_{Y_1}, \ldots, A_{Y_j}, \ldots, A_{Y_N})$$

where A_{X_i} and A_{Y_j} represent activation values of world knowledge and the other knowledge respectively, and the strengths in the constructed network by a matric, C.

The initial activation is set as follows:

$$A_{X_i}{}^0 = 1.0 \qquad (1 \le i \le M)$$

$$A_{Y_j}{}^0 = 0.0 \qquad (1 \le j \le N)$$

The activation vector after v-th iteration, $\overline{A}^{(v)}$ is defined as follows:

$$A_{X_i}{}^{(v)} = 1.0$$

$$A_{Y_j}{}^{(v)} = \frac{\max(0.0, A^{(v)}{}_{Y_j}}{\sum_{k=1}^{N} \max(0.0, A^{(v)}{}_{Y_k})}$$

where unnormalized activation vector calculated by matrix multiplication,

$$\overline{A}^{(v)} = C \times \overline{A}^{(v-1)}$$

is normalized.

When average change of the activation vector is below 0.0001, that is,

$$\frac{1}{N} \times \sum_{j=1}^{N} | A_j^{(v)} - A_j^{(v-1)} | < 0.0001$$

the network is considered to be stabilized. The activation vector, $\overline{A}^{(v)}$ becomes the final activation vector.