인지과학, 제 14 권 제 3 호, 2003년

지시문을 통한 학습: 이해-기반 접근
Learning from Instruction: A Comprehension-Based Approach

김 신 우[*,***] 김 민 영[*] 이 지 선[*] 손 영 우[*,**]
(Shin Woo Kim) (Min Young Kim) (Jisun Lee) (Young Woo Sohn)

요 약 학습에 대한 이해-기반 접근에 따르면 새로운 정보는 기존의 배경지식과 통합되어 정신표상을 형성하며 이는 다른 새로운 정보를 결합하는데 사용된다고 가정한다. 지시문을 통한 학습상황에서 인간과 계산적 모형의 수행비교를 통해 이 접근법이 타당하다는 것을 보여주었다. 구성-통합 이론(Kintsch, 1988; 1998)에 근거한 계산적 모형(ADAPT-UNIX)은 사용자들이 UNIX 복합 명령문을 생성하는데 도움을 주기위해 제시된 지시문 학습에 높은 예측력을 보였다. 더불어, 제시된 지시문을 사용하여 올바른 복합명령문을 생성하는 과제수행도 실제 인간수행과 높은 유사성 보였다. 배경지식의 수준에 따라 지시문이 학습과 적용에 차별적인 영향을 미친다는 교육적 함의와 이해-기반 인지모델의 이론적 함의가 논의되었다.
주제어 계산적 인지모형, 이해, 학습, 유닉스

Abstract A comprehension-based approach to learning assumes that incoming information and background knowledge are integrated to form a mental representation which is subsequently used to incorporate new knowledge. It is demonstrated that this approach can be validated by comparing human and computational model performance in the prompt learning context. A computational model (ADAPT-UNIX) based on the construction-integration theory of comprehension (Kintsch, 1988; 1998) predicted how users learn from help prompts which are designed to assist UNIX composite command production. In addition, the comparison also revealed high similarity in composite production task performance between model and human. Educational implications of present research are discussed on the basis of the fact that prompt instructions have differential effect on learning and application as background knowledge varies.
Keywords computational cognitive model, comprehension, learning, UNIX

1. INTRODUCTION

Learning from problem solving episodes has previously been investigated by different traditions and several have been implemented as computational models. For example, case-based learning (e.g., Hammond, 1989) assumes that we acquire knowledge by storing cases in memory which are, in general terms, the specific plans for different problems. Search-based models like SOAR learn by chunking the results of the search process (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 processes (Anderson, 1993; 1998). The present research builds upon a third emerging theory of learning, a comprehension based approach, which uses an association between problem descriptions (in this case, incoming instructions to produce a command) and background knowledge to first activate relevant knowledge to construct a coherent situation model. This mental representation is then used to incorporate new knowledge (e.g., Schmalhofer & Tschaitschian, 1993).

In the present research, it is hypothesized that a computational model based on the construction-integration

* 연세대학교 심리학과
** 연세대학교 인지과학협동과정
*** 연세대학교 인간행동연구소
연구세부분야: 응용인지심리
교신저자: 김신우, 016-724-1445, kims@yonsei.ac.kr
120-749 서울시 서대문구 신촌동 134번지 연세대학교 심리학과

theory of comprehension (Kintsch, 1988; 1998) can explain and predict how individual users will comprehend and learn from help instructions as they attempt to generate complex computer commands. 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 & Tschaitschain, 1993). This approach also proved to be effective 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 therefore can 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 test the ability of this approach to predict human performance in a learning environment. Therefore, the main goal of this study is to determine if this comprehension-based framework can be extended to account for learning from technical instructions.

1.1. 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 and about swimming. 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 inter-item overlap and current task relevance.

The ability to simulate context-sensitive knowledge activation is most important in present research because the task domain requires novel plans of action rather than retrieval of known routine procedures (e.g., Holyoak, 1991). Symbolic-connectionist architectures such as the one utilized in the present study use symbolic rules to interrelate knowledge in the network, and then spread activation throughout the network using connectionist constraint- satisfaction algorithms. This architecture has significant advantages over solely symbolic or connectionist forms since it enables to study context-sensitive aspects of adaptive problem solving (e.g., Broadbent, 1993; Holyoak, 1991; Holyoak & Thagard, 1989; Mannes & Doane, 1991; Thagard, 1989).

2. PRESENT RESEARCH

The main goal of present research is to determine if comprehension-based framework can be extended to account for learning from technical instructions. Specifically, it is evaluated whether the comprehension strategies of ADAPT-UNIX, a construction-integration model containing knowledge of UNIX commands, adequately account for the type of instructions users find helpful to their command production performance. Detailed empirical data on learning to produce complex, sequence-dependent commands in the UNIX operating system is available for this purpose. In previous empirical studies (Doane, McNamara, Kintsch, Polson, & Clawson, 1992; Sohn & Doane, 1997), users of varying experience with the UNIX operating system was asked to produce complex UNIX commands, and then provided help prompts when the commands they produced were erroneous. The help prompts were designed to assist users with both knowledge and processes that previous research has indicated are lacking in less expert users (Doane, Pellegrino, & Klatzky, 1990). The results showed significant differences in learning from instructions (prompts) as a function of UNIX background knowledge.

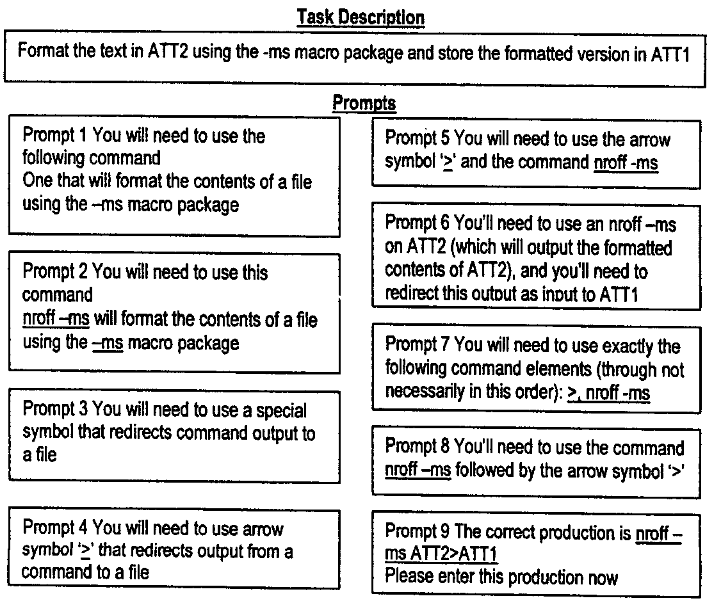
In the present research, comprehension-based computational model (ADAPT-UNIX) is extended to include learning mechanisms in order to model the individuals in the prompting study. To do so, each individual's performance was analyzed to identify their initial knowledge base, which represents the knowledge they displayed without prompting. Using this knowledge base, the same prompts which are given to participants were given to model when it executes an unsuccessful action plan, and then run the model again so the incoming prompt instructions can activate knowledge to attempt to solve the problem again. This procedure enabled to extend the theory in understanding how users learn from instructions to plan complex actions, and a detailed analysis of the match between model and actual performance could be acquired.

3. EMPIRICAL PROMPTING STUDY

The goal of the empirical prompting study was to determine more precisely what users at different levels of expertise know about UNIX, what information is lacking when users produce erroneous commands, and what information (i.e., prompt contents) help the users. The experiment used a prompting paradigm to assess the knowledge and process of users at various levels of expertise. It's assumed that users have different amounts of the required four types of knowledge (see General knowledge in Table 1) and that displaying the prompts which help with each type of knowledge will influence subsequent user performance, if they lack this knowledge.

3.1. Participants

In the full empirical study (Doane, McNamara, Kintsch, Polson, & Clawson, 1992), twenty-two engineering majors completed 21 composite command production tasks. However, the present research analyzes only subset of these participants because only two representatives from each of three expertise levels were modeled for the proposed purpose. All participants had received prior instruction about redirecting standard input and output in their coursework and had experience with using redirection symbols to complete coursework or other tasks using UNIX. 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).


Task Description

Format the text in ATT2 using the -ms macro package and store the formatted version in 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 2 You will need to use this command
nroff –ms will format the contents of a file using the -ms macro package

Prompt 3 You will need to use a special symbol that redirects command output to a file

Prompt 4 You will need to use arrow symbol '>' that redirects output from a command to a file

Prompt 5 You will need to use the arrow symbol '>' and the command nroff -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 7 You will need to use exactly the following command elements (through not necessarily in this order): >, nroff -ms

Prompt 8 You'll need to use the command nroff –ms followed by the arrow symbol '>'

Prompt 9 The correct production is nroff –ms ATT2>ATT1
Please enter this production now

(Figure 1) Example of task description and prompts for the problem 'nroff -ms ATT2>ATT1'.

3.2. Materials and Procedure

All production tasks were performed on a computer. The stimuli were task statements, a fixed directory of file names and a series of help prompts, displayed when appropriate on the screen, as well as three error cards presented by the experimenter. Participants typed their command or series of commands on the keyboard to accomplish a given task and then used the mouse to click on a display button to obtain an evaluation of their answer. The task instruction described actions that could best be accomplished by combining two or three commands through the use of redirection (i.e., composite problems; see Figure 1). Accompanying the task statement was a fixed directory listing of all file names that were used in the experiment.

For incorrect response, there was a series of help prompts designed to address specific types of deficits in the participants' UNIX knowledge. These prompts were displayed on the screen one at a time in a fixed order regardless of the type of error that the participant had made. 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.

There are seven different areas in which the help prompts could assist the participants, and these are best described by referring to the example in Figure 1. Prompt 1 parses the task statement into relevant command concepts. Prompt 2 identifies actual command syntax (command syntax knowledge). Prompt 3 explains concepts of redirection in an abstract fashion, independent of syntax (conceptual I/O redirection knowledge). Prompt 4 identifies the actual I/O redirection symbols (I/O redirection syntax knowledge) required for the problem. Prompts 5 and 7 remind the participant of the complete set of items that have already been identified in previous prompts. Prompt 6 is the first prompt that determines the order of commands and symbols for the participant (providing command redirection knowledge and help with tracking intermediate results). This information is repeated in Prompt 8. Finally, Prompt 9 gives the participant the correct production.

4. SIMULATION EXPERIMENTS

We simulated six UNIX users (two novices, two intermediates, and two experts) participated in the aforementioned empirical prompting study. To simulate a given individual, ADAPT-UNIX 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, ADAPT-UNIX received 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.

ADPT-UNIX simulation will be outlined starting from knowledge structure. This will be followed by procedures used to construct individual's initial knowledge base to simulate each participant. Then, ADAPT-UNIX model and its execution will be overviewed.

4.1. ADAPT-UNIX Knowledge Representation

ADAPT-UNIX 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 ADAPT-UNIX represent knowledge about the computing domain or the present task. ADAPT-UNIX represents the three major classes of knowledge proposed by Mannes and Kintsch (1991); World, general (e.g., declarative knowledge), and plan element knowledge (e.g., procedural knowledge represented as If/Then rules; see Table 1).

Table 1. Examples of Knowledge Representation in ADAPT-UNIX

Type of knowledge	Abbreviated propositional representation
World knowledge	
	File exists in directory
	At system level
	Goal is to format file
General knowledge	
Command syntax	"nroff" formats file
I/O syntax	">" redirects output from command to file
Conceptual I/O	I/O can be redirected from command to file
Command redirection	"nroff" output can be redirected to file
Plan knowledge	
Name:	Format contents of a file
Preconditions:	Know "nroff"formats file
	Know "nroff" "-ms" flag
	Know file exists in directory
Outcome(s):	Formatted file exists

4.1.1. World Knowledge

World knowledge represents the current state of the world. This includes knowledge of the current task, existing files on the current directory, and the system in use (UNIX). These facts are contextually sensitive and fluid as the task and simulated performance progresses.

4.1.2. General Knowledge

General knowledge refers to factual (declarative) knowledge about UNIX. In ADAPT-UNIX, general knowledge includes the command syntax, I/O syntax, conceptual I/O redirection, and command redirection UNIX knowledge required to produce correct composite commands.

4.1.3. Plan Element Knowledge

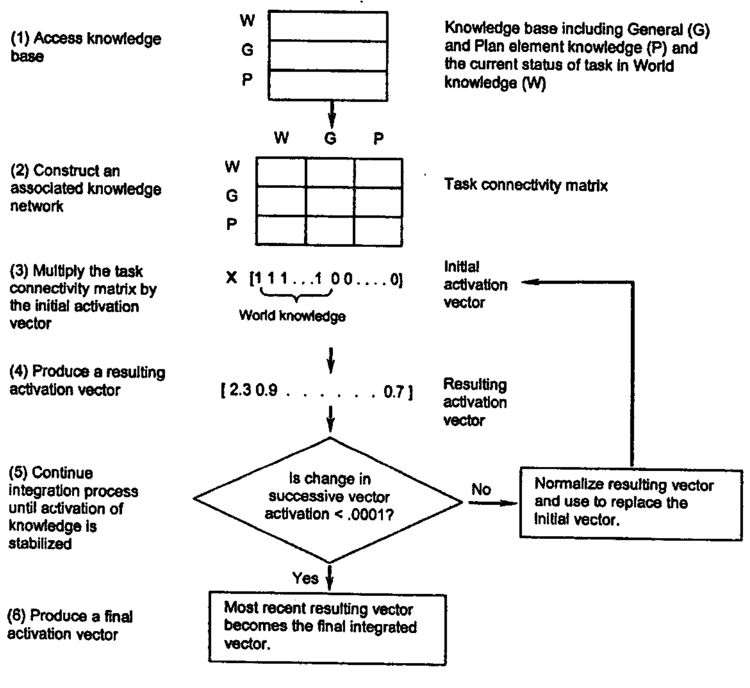
Plan element knowledge represent executable (procedural) knowledge. Plan elements describe actions that can be taken in the world, and they specify conditions under which actions can be taken. Thus, users have condition-action rules that can be executed if conditions are correct. Plan elements are three-part 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 knowledge can be executed. When plan element fires, plan element outcome fields are added to the model's world knowledge.

4.2. Constructing Individual Knowledge Base

Each individual's starting knowledge was constructed by evaluating each user's UNIX knowledge using a small portion of empirical performance data. This determined initial contents of individual knowledge base were accessed by ADAPT-UNIX to simulate each individual. Using overlay method (see VanLehn, 1988), missing knowledge as well as incorrect knowledge was scored. To determine the starting state of a participant's UNIX background knowledge, each knowledge component of the participant's response was scored as to whether it was made before or after explicit instruction. To account for users' erroneous as well as correct command productions, each user's answers were also scored to determine what incorrect knowledge the user displayed before instruction on that knowledge.

4.3. ADAPT-UNIX Construction-Integration Cycle

The participant's knowledge was entered into an initial


(1) Access knowledge base
W
G
P
Knowledge base including General (G) and Plan element knowledge (P) and the current status of task in World knowledge (W)
(2) Construct an associated knowledge network
W G P
W
G
P
Task connectivity matrix
(3) Multiply the task connectivity matrix by the initial activation vector
X [1 1 1 . . .1 0 0 . . . . 0]
World knowledge
Initial activation vector
(4) Produce a resulting activation vector
[2.3 0.9 . . . . . . 0.7]
Resulting activation vector
(5) Continue integration process until activation of knowledge is stabilized
Is change in successive vector activation < .0001?
No
Normalize resulting vector and use to replace the initial vector.
Yes
(6) Produce a final activation vector
Most recent resulting vector becomes the final integrated vector.

(Figure 2) Schematic representation of construction-integration process.

knowledge base, and the model was given the initial problem statement in the world knowledge. Knowledge about the domain and a particular task is represented in a distributed manner, and the pattern of activation across nodes determines the current model of the problem situation. These symbolic nodes are interrelated in two distinct stages that are uniquely structured to represent comprehension.

4.3.1. Construction

During construction, the model computes relationships between propositions in the knowledge base (k) to construct a task-specific network of associated knowledge as depicted in Figure 2, steps 1-2. The model uses low-level rules to construct asymmetric 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). The resulting network, depicted in Figure 2, 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 with simulations since Kintsch's (1988) model introduction.

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)

4.3.1.1. Binding

Before construction a task connectivity matrix, ADAPT-UNIX is given the task description in propositional form, as shown in Table 2. ADAPT-UNIX 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 call 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^ARRANGED^PROP1), they are treated as unique files and propositions are duplicated accordingly. Functionally, this duplication allows ADAPT-UNIX to represent the unconstrained binding process hypothesized by Kintsch (1988).

4.3.1.2. 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 and for each proposition embedded. 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).

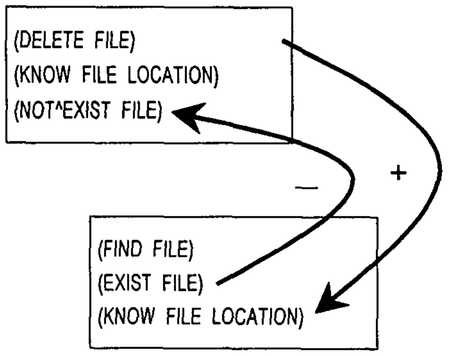


Figure 3. Example precondition and outcome interplan relationships. ("+" = positive; "-" = inhibitory).

4.3.1.3. Plan element relationships

Three field comprise a plan element; name, precondition, and outcome. Propositions representing these fields are represented as a single node in the network. Overlap between plan element precondition and outcome fields is calculated to estimate casual 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. 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 3 depicts these casual relations for two abbreviated example plan element 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 3). 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 exist in the world, then an asymmetric inhibitory link of -10.0 exists between each proposition in the world knowledge that matches the outcome(s) propositions. For example, if the out come 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, ADAPT-UNIX 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 -10.0, and -0.4 (inhibitory), and a weight 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 C-I modeling, and remain constant here. As suggested by Thagard (1989), the weight stability is critical for assessing the reliability of a cognitive architecture across simulation efforts.

4.3.2. Integration

The constructed network of knowledge represents unconstrained relations between knowledge elements. To develop a situation model (e.g., Kintsch, 1988), this knowledge is integrated by using a simple linear constraint-based algorithm 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 integrate the constructed knowledge base in illustrated in Figure 2, steps 3-6.

Computationally, integration constitutes the repeated post-multiplication of the constructed network (matrix) by a vector. The vector values 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 2, the initial vector values corresponding to the 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 post-multiplied by the connectivity matrix resulting from the construction process (see step 3 in Figure 2), and a resulting activation vector is produced (see step 4 in Figure 2). After each multiplication, the vector weights corresponding to the current set of world propositions are reset to 1.0, and the remaining propositions are normalized to ensure their sum is a constant value across integrations (see step 5 in Figure 2).

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

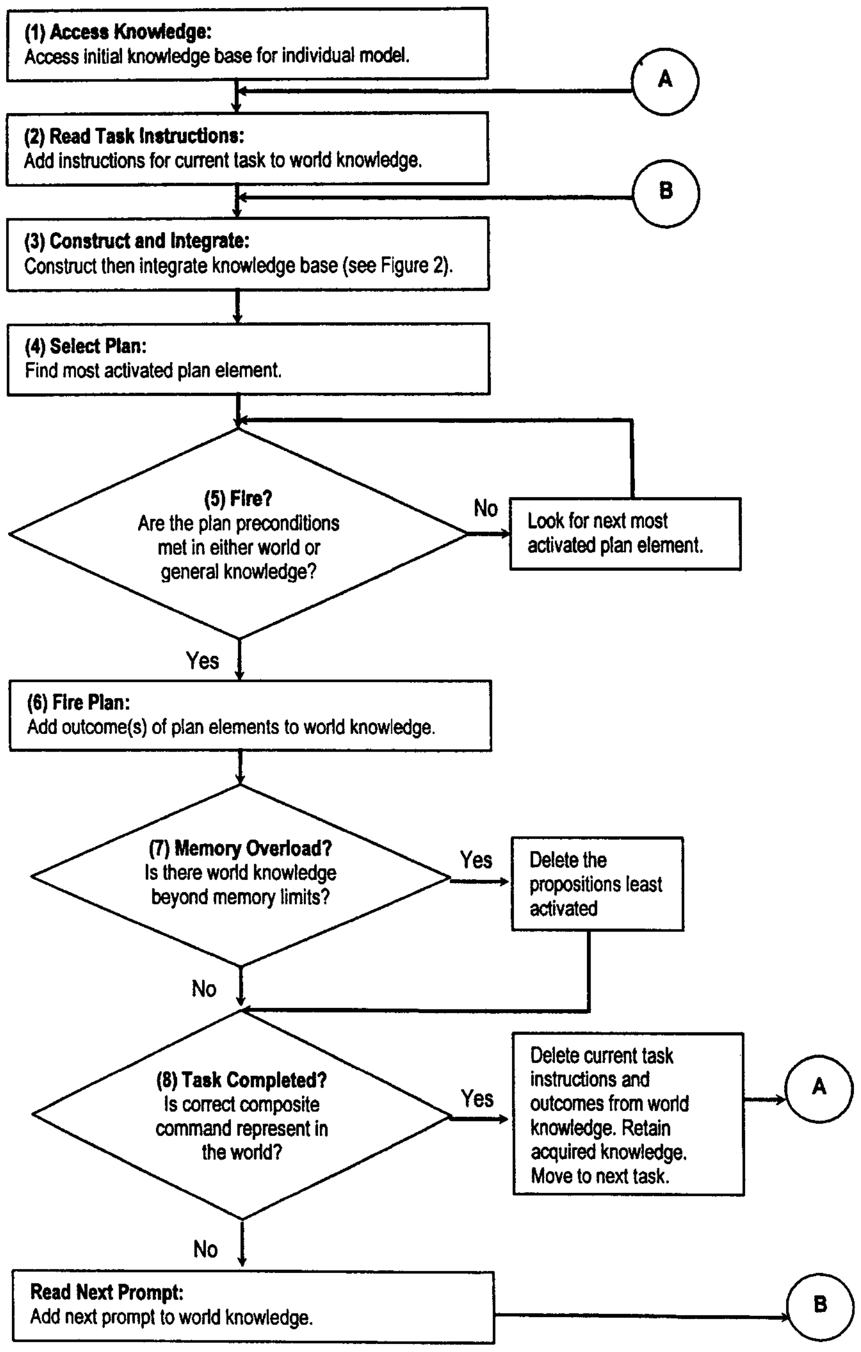

(1) Access Knowledge:
Access initial knowledge base for individual model.
A
(2) Read Task Instructions:
Add instructions for current task to world knowledge.
B
(3) Construct and Integrate:
Construct then integrate knowledge base (see Figure 2).
(4) Select Plan:
Find most activated plan element.
(5) Fire?
Are the plan preconditions met in either world or general knowledge?
No
Look for next most activated plan element.
Yes
(6) Fire Plan:
Add outcome(s) of plan elements to world knowledge.
(7) Memory Overload?
Is there world knowledge beyond memory limits?
Yes
Delete the propositions least activated
No
(8) Task Completed?
Is correct composite command represent in the world?
Yes
Delete current task instructions and outcomes from world knowledge. Retain acquired knowledge. Move to next task.
A
No
Read Next Prompt:
Add next prompt to world knowledge.
B

Figure 4. Schematic representation of the procedure used to simulate each UNIX user's command production.

the final activation vector and represents the stabilized activation of knowledge. The final activation vector is then used by the model to make execution decisions regarding the next plan element to fire (see steps 5-6 in Figure 2).

4.4. Overview of the Model Execution

ADAPT-UNIX was run to simulate each individual's performance in command production tasks, responding to help prompts identical to those given to each of the users. The simulation procedures are schematically represented in (Figure 4).

4.4.1. Plan Selection

A given user's knowledge base is accessed by ADAPT-UNIX, and the problem description for the first composite production task is added to the world knowledge. The model executes a construction-integration cycle, and finds the most activated plan element 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. If they do not exist, then this process is repeated using the next-most-activated plan element until a plan can be fired. The outcome field of the fired plan element is added to the world knowledge to update the world situation. Then, construction-integration begins again with the modified knowledge base until the model represents a plan of action (made up of a sequence of fired plan elements) that will accomplish the specified task in the world (see steps 4-6 in Figure 4).

4.4.2. 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.

In ADAPT-UNIX, 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 ADAPT-UNIX as the transfer of prompt propositions from temporary in-the-world knowledge to permanent 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.

4.4.3. Memory Constraints

A memory component (see step 7 in Figure 4) is incorporated in the ADAPT-UNIX simulations to represent working memory capacity limitations. Capacity limitations are represented by deleting in-the-world propositions once their number exceeds working memory capacity limitations (represented as a parameter value), where propositions with the lowest activation are deleted first. For example, if the capacity limits are set to 4, the in-the-world propositions that are not among the fourth most activated are deleted. Because proposition activation is constrained by relevance to the current task context, this procedure simulates context-sensitive working memory limitations. This capacity function is applied only to the in-the-world propositions that represent instructional information. However, a decay component of memory is not incorporated in the ADAPT-UNIX simulations because all instructions remain on the screen in the empirical UNIX study.

5. RESULTS AND DISCUSSION

To determine if comprehension-based approach can be extended to account for learning from technical instructions, results from empirical and simulation study were compared regarding two aspects of the data; correct command production and four types of UNIX knowledge.

5.1. Correct Command Production

5.1.1. Scoring Correct Command Production

For the empirical work, productions were scored as correct by the computer if they matched the syntax of the idealized command (spaces were not included). Thus, a participant had to produce the command that required the least number of keystrokes (i.e., participants could not substitute 'sort file 1>temp; head temp>file 2' for the command 'sort file 1| head>file 2'). Productions produced by each model were scored using the same rules of correctness. The problems simulated in this paper are those requiring the

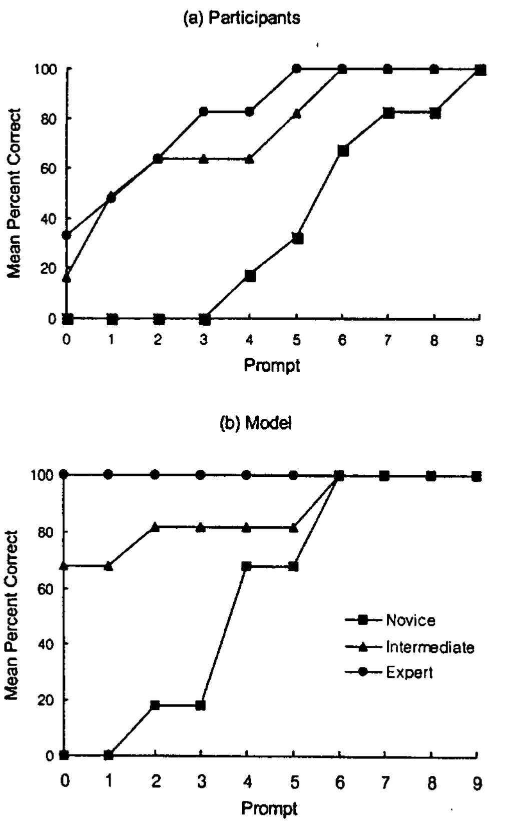


Figure 5 (*a & b*). Mean percent correct productions for novice, intermediate, and expert participants and models.

greatest percentage (60-100%) of new knowledge for solution, as detailed in Doane et al. (1992). This subset of problems is discussed in this paper.

5.1.2. Correct Command Production as a Function of Prompt

Figure 5 shows the cumulative percentage of correct composite productions for the three participant groups and models as a function of prompt. The data are cumulative; participants (and modeled participants) who correctly produced a problem at Prompt 4 were included as correct data point as Prompts 5-9 as well. Thus, at Prompt 9, all of the participants in each expertise group were at 100% correct performance. Looking at the empirical results in Figure 5(a), experts have the highest correct percentage overall, followed by the less expert groups. Prompts have differential influences on correcting performance for the three expertise groups. For example, the change in percent correct performance from Prompt 3 to Prompt 4 is zero · for intermediates and experts, suggesting that Prompt 4, which gives I/O syntax information (see Figure 1) provide little or no new information to them. Conversely, the same change between Prompts 3 and 4 for the novices is large, suggesting that this prompt does provide them with significant new information. Experts and intermediates require fewer prompts in order to obtain perfect performance. Novices, in contrast, only obtain perfect performance once they are exposed to the final prompt which gives the exact command, Prompt 9. Looking at the modeling results in Figure 5(b), the basic expertise effect was obtained.1) For the model, as for the participants, Prompt 4 helps the novices but not the intermediate group.

5.2. Knowledge Analysis

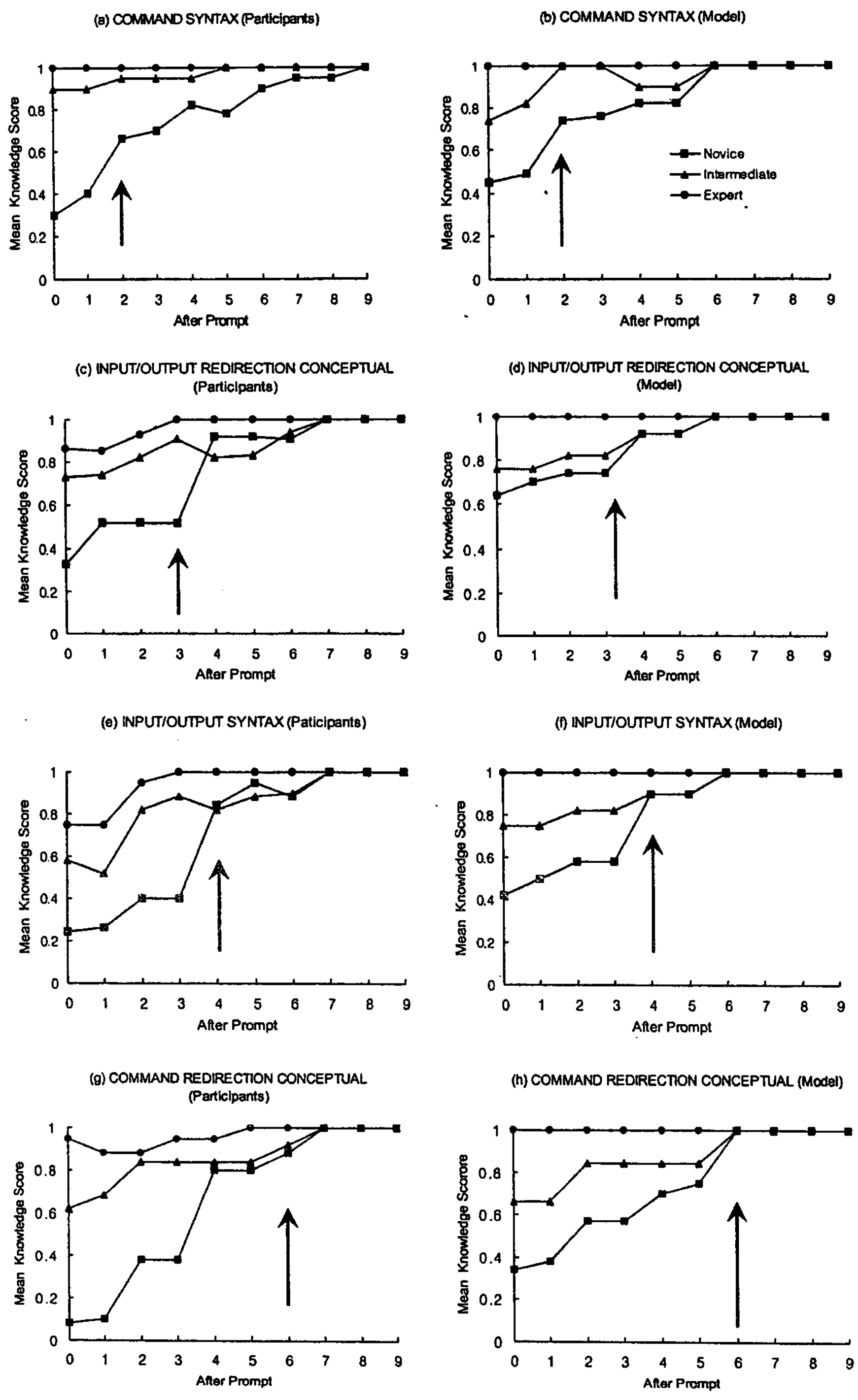
5.2.1. Scoring of Knowledge

Each of the problems given to participants and the model required a certain amount of the four types of general knowledge discussed earlier. Answers for the present tasks were scored for the percentage of each type of knowledge displayed by a participant and by a model of the participant at each prompt level.

5.2.2. Knowledge Score after Each Prompt

Figure 6 shows the mean knowledge scores for the three expertise groups after Prompts 0-9 for both the participants and models. The arrow markers specify which prompt first provided information relevant to the knowledge type displayed in the graph. For example, in Figure 6(a), Prompt 2 is the first prompt that describes all of the command syntax knowledge required to complete the task (see Figure 1 for an example of all prompt types described in this section), and the arrow indicates knowledge displayed after presentation of Prompt 2. The change in the knowledge score for command syntax between Prompts 1 and 2 indicates the effect of Prompt 2. The component knowledge shown is higher than the percent correct scores shown in Figure 5(a). This is because an attempt can show high, but

1) Two representative experts were scored across their performance, and they displayed all of the requisite knowledge without prompting. This led to 100% correct performance by the model (see Figure 5(b)).



(Figure 6) *(a-h)*. Mean command syntax knowledge, I/O conceptual knowledge, I/O syntax knowledge, and command redirection knowledge for novice, intermediate, and expert participants and models.

not perfect component knowledge, and component knowledge must be perfect for an attempt to be entered as correct in Figures 5(a) and 5(b).

The difference between percent correct performance and the amount of knowledge displayed in an attempt can be examined by comparing the knowledge scores shown in Figures 6(a) and 6(b) with the percent correct performance shown in Figures 5(a) and 5(b). Figure 6(a) suggest that for novice and intermediate groups, presentation of Prompt 2 improves command syntax knowledge, but only intermediates show improvement in percent correct performance (see Figure 5(a)). The lack of change in percent correct performance for the novice groups (see Figure 5(a)) suggests that for them, the prompt is not sufficient to guarantee that subsequent attempts will show perfect command syntax knowledge. Figure 6(b) shows similar pattern of improvement for modeled novices and intermediates in response to the command syntax prompt. The novice model shows the greatest improvement in command syntax knowledge at Prompt 2. The novice model also shows an increase in correct performance (see Figure 5(b) at Prompt 2), which differs from the participant performance. The remaining Figures (Figures 6(c-h)) can be examined in a similar fashion, where the comparison show that the model does a good job of predicting what type of information is important to improve the amount of knowledge displayed in an attempt for novices, and slightly less so for intermediates.

To quantify the fit between the participant and the model data, correlations between participants and their corresponding models were performed on the knowledge scores as a function of prompt and on the change in knowledge scores as a function of prompt. The change scores provide a more stringent test of the fit between the model and the participant data because it pinpoints the changes between prompts rather than the general increase in knowledge. Table 3 shows the resulting correlation values. Descriptively speaking, Table 3 suggests that the model does a good job for predicting the pattern of improvement in percent correct performance, showing the best fit with the novice data. The change scores indicate that the model does a good job of predicting the syntax-based knowledge for novices, and command redirection knowledge for the intermediates. The fit between expert's performance and the model's predictions was not calculated due to ceiling performance.

The analysis suggests that there is a good match between what the model learns from instructions, and what the actual participants learn. The differences in what knowledge is relevant as a function of expertise is consistent for the actual novice and intermediate participants, and their models.

6. GENERAL DISCUSSION

We have shown that the construction-integration model can be extended to account for learning from technical instructions. Using this comprehension-based approach, it was possible to predict what prompt instructions users will apply and learn as a function of their background knowledge. This work has implications for computer-aided instruction and intelligent tutoring. If we can specify what instructions will be effective based on the activation resulting from the overlap between incoming instructions and background knowledge, then we can design more effective instructional systems without having human participants interact with the complex system.

Previous research (Doane et al., 1992) on UNIX command production have suggested that having all the necessary knowledge including declarative and procedural knowledge of the component commands is not sufficient for novices to accurately produce correct commands. Rather, they also needed help in ordering the elements which imposes significant load on working memory. Later, Sohn and Doane (1997) have pinpointed the locus of this deficit which was working memory limitations and found out that specially designed graphical aids reduce working memory demands and thereby improves novice performance on correct command production. In the present simulation experiment, ADAPT-UNIX was able to mimic this critical feature of novice UNIX user cognition, which is working memory deficit, and enabled us to identify what knowledge UNIX beginners lack and what knowledge will help their command production as well. In conjunction with Sohn and Doane (1997), we will be able to design more effective UNIX programming interface by providing prompts and graphical aids (as suggested by Sohn & Doane, 1997) which contain specific knowledge (identified by ADAPT-UNIX) the user may require in accordance with their background knowledge in UNIX programming.

Theoretically, the present research provides further evidence of the centrality of comprehension in cognition (e.g., Gernsbacher, 1990). This was accomplished by using a comprehension-based model to simulate a complex problem solving and learning environment. This enabled to extend 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 of planning, and more importantly, suggests that the contextually constrained activation of knowledge central to the comprehension-based approach is not only descriptive but also predictive.

As Newell (1990) has once pointed out, the model utilized in the present research has been tested on a wide variety of tasks, and as such an architecture of cognition is under development. Further, the implementation satisfied many of the stringent criteria mentioned by Thagard (1989) for consistent architecture development. Specifically, the same parameters, relationships, and weights are used, and only the user knowledge base varied. And the variation in knowledge bases was carefully controlled and based on rigorous scoring rules.

The present work does not focus on differentiating ADAPT-UNIX'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, Laird, & Newell, 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 process 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. 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. In this case, a relatively parsimonious model has provided reasonable fits to highly complex human computer interactions as well as skill and knowledge acquisition.

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, the assumption that instructions in plain view are activated, used, and remembered is clearly unwarranted. Therefore, next step is to incorporate the retrieval model described by Kintsch (1988) into this comprehension-based theory of learning.

REFERENCES

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. (1993). *Rules of the mind.* 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., Kintsch, W. & Polson, P. (1989). Action Planning: Producing UNIX commands. *Proceedings of the 11th Annual Conference of the Cognitive Science Society.* (pp. 458-465). Erlbaum.

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.

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.

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

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.

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.

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

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.

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

접 수	2003년 06월 30일
게재승인	2003년 09월 20일