COGNITIVE SCIENCE **20,** 301–356 (1996)

Learning Consistent, Interactive, and Meaningful Task-Action Mappings: A Computational Model

ANDREW HOWES
RICHARD M. YOUNG
MRC Applied Psychology Unit
Cambridge, England

Within the field of human–computer interaction, the study of the interaction between people and computers has revealed many phenomena. For example, highly interactive devices, such as the Apple Macintosh, are often easier to learn and use than keyboard-based devices such as Unix. Similarly, consistent interfaces are easier to learn and use than inconsistent ones. This article describes an integrated cognitive model designed to exhibit a range of these phenomena while learning task-action mappings: action sequences for achieving simple goals, such as opening a file in a word processor. The model, called TAL, is of a user who is familiar with the basic operations of a keyboard and mouse, but unfamiliar with the particular menu structures, words, and actions required to use the device. The model is constructed in Soar and employs a single set of architectural mechanisms. It exhibits behavior that captures human preference for consistent, interactive, and meaningful task-action mappings.

INTRODUCTION

Users of information processing devices must acquire knowledge of how to map the tasks they wish to achieve into sequences of actions on the device. This article focuses on how people acquire knowledge of the mappings from tasks, such as saving a file in a word processor, to a sequence of low-level actions, such as moving a mouse-pointer to a menu item, pressing the mouse button, and then typing in a file name. These task-action mappings, as Payne and Green (1986) termed them, form a substantial part of the learning load for a novice user.

Stephen Draper, Thomas Green, and Stephen Payne have all made invaluable comments on this work. The research was supported by the UK's Joint Councils' Initiative in Cognitive Science and Human–Computer Interaction.

Correspondence and requests for reprints should be sent to Andrew Howes, School of Psychology, Cardiff University of Wales, P.O. Box 901, Wales, CF1 3YG. E-mail: HowesA@cf.ac.uk.

Some task-action mappings are easier to learn than others. Psychologists have observed that users of computer interfaces find those that are consistent, interactive, and meaningful easiest of all (Draper, 1986; Lewis, 1988; Payne & Green, 1986). Briefly, a task-action mapping is consistent if it shares structural properties, such as syntax, with other task-action mappings; it is interactive if the device and its display implicitly suggest the correct mapping; and it is meaningful if the actions can be generated from the semantics of the task.

In this article we review the literature on consistency, interactivity, and meaningfulness and then describe an integrated model of the phenomena. The model, called Task-Action Learner (TAL), is implemented in the Soar problem-solving architecture. Soar is a problem space theory of the human cognitive architecture proposed by Laird, Newell, and Rosenbloom (1987), and more recently described by Newell (1990). Its origin was as a system for solving well-structured toy problems such as the 8-puzzle or the Tower of Hanoi but it has more recently been turned to supporting the investigation of less well-structured and more interactive tasks (Howes & Young, 1991; Huffman & Laird, 1993; Vera, Lewis, & Lerch, 1993; Young, Howes, & Whittington, 1990).

TAL is a model of a user who is familiar with the basic operations of a keyboard and mouse but unfamiliar with the particular menu structures, words, labels, and actions required to achieve tasks. In particular the modeled user is assumed to know that moving a mouse on the table results in the movement of a pointer on the display; that pressing the button on the mouse constitutes an action on one of the items displayed in the same location as the pointer; and that pressing a key on the keyboard results in the character written on that key being communicated to the device. The user also brings to the device some knowledge of lexical semantics. For example, in the domain of text editing, a user may consider that the words *file* and *folder* refer to similar kinds of object. Lexical knowledge of this form enables tasks that the user has in mind to be articulated in a number of different ways.

Further, TAL is an integrated model. The role of integrated models of cognition has been championed by Newell (1972, 1990), who claimed that psychology has determined sufficient empirical regularities for cognitive science to commence the search for a single unified theory of cognition. Following Newell, TAL does not consist of a "consistency" component, an "interaction" component, and a "meaning" component; rather it consists of a single set of mechanisms that work together to produce the desired behaviors. Also, the model will capture both how task-action mappings are learned and how they are performed.

This article shows that TAL's behavior is consonant with empirical observations of human performance. The idea is not that the model should

be able to learn arbitrarily complex task-action mappings, rather that with a very simple learning mechanism it should learn devices that people find easy to learn. Some of these devices are characterized as walk-up-and-use (Polson & Lewis, 1990). For example, automatic teller machines (ATMs) are intended to be usable without any prior training. As we will see, TAL is not a powerful learning mechanism. For example, although it models some of the errors that people make, it does not model the way that they recover from those errors.

The remainder of this introduction reviews the literature on consistency, interactivity, and meaningfulness, and provides a basis for defining the behaviors to be modeled. Subsequent sections describe TAL, relate TAL to human behavior, and discuss the implications of the model.

Consistency

Consistency is one form of simplicity, and it is therefore not surprising that it plays an important role in learning and performance. The consistency of a device constrains learning when users make the assumption that task-action mappings learned for one task will apply to all semantically similar tasks. For example, if a task-action mapping to open a file has a certain syntax then people will assume that the mapping for closing a file will have the same syntax.

There are many ways in which a computer interface can be consistent. It can be consistent with the underlying system model, or with the user's task model (Gentner & Grudin, 1990). It can be internally consistent and/or consistent with other interfaces, or it can consistently employ a particular metaphor. In this article, the primary interest is in the internal consistency of task-action mappings; that is, the consistency of the task-action mappings with each other for a single device.

Payne and Green (1986, 1989) reported a number of experiments to show that consistent interfaces are easier to learn and use. For example, Payne and Green (1989) found that the number of syntax errors made by participants increased with the use of devices with inconsistent grammars. Other studies, for example, Barnard, Hammond, Morton, Long, and Clark (1981), Green and Payne (1984), Payne (1985), Kellogg (1987), and Lee, Foltz, and Polson (1994) also indicate the advantage of consistent design.

Human-computer interaction has seen the development of a number of formal languages for describing users' knowledge of interfaces that seek to make the consistency of task-action mapping knowledge explicit. These have progressed from Reisner's (1981) use of Backus-Naur Form (BNF) to specify precisely the syntax of interfaces, through Payne's (1985) Set Grammar to Payne and Green's (1986) Task-action Grammar (TAG). The crucial point has been to recognize that a large part of consistency should be measured relative to the semantics of the users' tasks (see Green, Schiele, &

Payne, 1988, for a review). For example, the organization of knives in a house should be viewed relative to their function. Thus knives for cutting food are kept in the kitchen and knives for cutting wood are kept in the garage or basement (cf. Grudin, 1989).

In particular, the TAG notation assumes that people organize task-action mappings according to the semantic similarities and differences of the tasks, and that perceived consistency is an emergent property of this configural organization of mapping knowledge. It thereby captures a number of aspects of the consistency of task-action mappings, including the syntactic consistency and mnemonic consistency. An interface is syntactically consistent if the order in which types of items occur across a broad set of tasks is the same. For example if in the command to copy a file the word *copy* must be typed before the description of the "file" then the user may expect that in the command to delete a file, the word *delete* must also be typed first. The mnemonics of a language are consistent if the way in which abbreviations are derived from command names is the same across many tasks. For example, if the command to "remove" a file is executed by taking the first two consonants of the command name then it will help learnability if other commands also are executed by taking the first two consonants of their names.

Task-Action Grammars capture consistency by representing the competence of a skilled user in a formal notation. The formalism consists of a notation for describing the knowledge that a user needs in order to execute a set of tasks with an interface. For example where T is a task such as changing the font in a word processor W, a TAG would describe the set of rules that map T into the sequence of actions that are required to execute T on W. Rules take the following form:

IF T **THEN** Action-1 + Action-2 + . . . Action-N.

Many rules of the form T→A would be required to describe the interface. However, if the action sequences for two or more tasks were consistent, for example, if they shared the same syntax, then the TAG notation would allow just one rule to be written for those tasks. In this way, the number of rules required to describe the knowledge used for a particular interface would be reduced with the consistency of the interface.

Interaction

The second constraint in which we are interested is interaction. It has long been realized that to study cognition one must also study the environments in which cognition occurs. For example, Newell and Simon (1972) stated that problem solving will depend on the external task environment every bit as intimately as it depends on the nature of the "built-in" internal memories (Newell & Simon, 1972, p. 801). In fact the interaction between people and

their environments is a major source of constraint on the acquisition and use of knowledge.

Interactivity has been studied in a number of fields, including the social sciences (Lave, Murtaugh, & de la Rocha, 1988), artificial intelligence (Agre & Chapman, 1990; Chapman, 1990), human–computer interaction (Briggs, 1990; Draper, 1986; Howes & Payne, 1990; Kitajima, 1989; Mayes, Draper, McGregor, & Oatley, 1988; Payne, 1990; Suchman, 1987), and educational research (Collins, Brown, & Newman, 1986; Miller & Gildea, 1987). One emergent view is that interaction provides essential constraints on many aspects of human behavior, including performance, learning, planning, and communication.

In the planning literature, it was realized early on that plans are not detailed, blueprint-like specifications of every action (Miller, Galanter, & Pribram, 1960). Plans are created and used as a resource during behavior, and do not completely predetermine it. Models that are based on this view include Dibs (Larkin, 1989). Dibs is a model of display-based problem solving, which is characterized by (amongst other things), (a) not being degraded by interruption, (b) being easily modified, and (c) having steps that are performed in a variety of orders. Dibs employs a simple problem-solving strategy: Attempt to get objects to "where they want to be." For example, in a coffee-making example described by Larkin, the recognition of the coffee filter cues the knowledge that the filter "wants to be" in the filter holder, and that the coffee powder "wants to be" in the mug. Dibs' methods thereby manifest all three characteristics: They are not degraded by interruption because they do not rely on internal state; they are easily modified because the knowledge consists of many small components; and their subcomponents are performed in many orders because at any stage many objects may cue knowledge of where they want to be. Further, Dibs makes cognitively plausible errors, for example, errors that are due to hidden state information in the display. Dibs has been used to model how people make coffee and how they solve simple algebra problems.

Like Dibs, the Pengi and Sonja systems (Agre & Chapman, 1990; Chapman, 1990) hardly do planning at all. Agre and Chapman (1990) described these models in terms of the plan-as-communication view. This is a view in which plans are thought of as natural language instructions: communicative objects that do not directly determine user behavior. Instead, using the plan requires figuring out how to make it relevant to the situation at hand, a process of interpretation that can be complicated and draw on a wide variety of resources.

In human–computer interaction, the interest in interaction has been in response to formal models of the user that fail to reflect the fundamental contrast between the learnability of keyboard-based and display-based or WIMP (window, icon, and mouse pointing) devices. Performance of a task

on a keyboard-based or "command-oriented" device involves the user recalling from memory a whole sequence of actions. These are typed on a keyboard and echoed back on a display. The device provides the user with no assistance or cues as to the required actions.

In contrast, Draper (1986) pointed out that display-based interfaces allow one partner's "utterances" to be referred to by the other's. For example if the computer (Partner 1) displays a button for confirming an action then a user (Partner 2) may refer to this button (by selecting it) in order to complete the task. Draper suggested that this facility promotes a style of activity at user interfaces that is more fruitfully analyzed in terms of the rapid, back-and-forth information flow between user and machine than as the accomplishment by the user of memorized, autonomous methods for performing tasks.

Draper's information-flow view of skill is at odds with that embodied in some formal models of the user (see Green et al., 1988, for a review). These include the Task-Action Grammar model (Payne & Green, 1986) reviewed earlier, GOMS (Card, Moran, & Newell, 1983), and CCT (Kieras & Polson, 1985). Howes and Payne (1990), Payne (1991), and Howes (1992) contain reviews describing how these models fail to capture interactive phenomena. They agree with Draper that the structure and content of expertise are subtly dependent on the details of the interaction between the cognitive system and the outside world. Expert users do not know "everything" about the device; rather, in service of their tasks, they become skilled at picking up and utilizing information provided on the display.

In response to this observation, Howes and Payne (1990) described Display-based Task-Action Grammars (D-TAG). These coordinate knowledge about the task with knowledge about the display in order to determine the next action. D-TAG rules take the following form:

IF Task + Display-state **THEN** Action.

D-TAG rules can thereby be written so that they are sensitive to the display state as well as the description of the task. Consequently, the rules can then express the competence to use a menu-based system in a way that does not involve recalling the names of menu labels. To understand this, consider experienced users performing the task of invoking a spelling checker with the command *spell*. If interacting with a keyboard-based interface, then the user will find that the device offers no cues as to the command name. The user must therefore recall the name from memory. In contrast, if interacting with a display-based interface, then the required command name will be presented on the device display, perhaps in the form of a menu. In this situation the user does not have to recall the required name, but instead can scan the items on the display until the name *spell* is recognized. It is the competence for this latter behavior that D-TAG captures.

A consequence of the recognizational rule structure is that the sequence in which D-TAG rules fire may be determined by the state changes in the device. For example, consider the task for invoking a spell checker with a particular dictionary. The user of a keyboard-based device would have to know the order in which to specify the command name (e.g., *spell*), and the name of the dictionary (e.g., *my_dictionary*). Most keyboard-based systems would allow the user to generate and type either order, but would only accept one of these, usually, *spell* followed by *my_dictionary*. An error would be the result of the user generating any other order. In contrast, the user of a display-based interface would find that the interface would allow only one order to be generated. This is because the items *spell,* and *my_dictionary* would only be presented on the display one at a time. In this circumstance there is no chance of error, and the order does not have to be committed to memory (Duff, 1989). By modeling this kind of competence, the skill expressed in a D-TAG is intrinsically bound to the device display. The knowledge encoded in the rules only determines a task-action mapping when combined with the cues provided by the device display.

As well as constraining expert skill, the interactivity of a device can also constrain the acquisition of new knowledge. It does this, in part, by constraining the set of hypotheses that the user develops about how to use the device (e.g., Payne, Squibb, & Howes, 1990; Shrager & Klahr, 1986). The fact that a device display presents candidates for action means that a user who does not know the correct action can guess by selecting the presented candidate that best matches the semantics of the user's current task. In contrast, the user of a keyboard-based interface must guess by generating candidate actions from memory and then typing them. Using this strategy there may be many more possible guesses: Speller, spl, spell, spell-checker, dictionary, and so on.

Another aspect of the way in which interactive devices constrain their use is the fact that an object perceptually cues the actions that can be performed on or with it. Gibson (1979) referred to this property of an object as an *affordance.* Physical input devices and representations on device displays offer affordances for action, for example, buttons afford being pushed and sliders afford being slid (Norman, 1988). Here we say that affordances provide a physical constraint on the set of actions that a learner needs to consider.

Designers use a number of variations on this technique (e.g., Gaver, 1991), and indeed, the subtlest aspects of how an object is represented can affect the perceived affordances. For example, in the Apple Macintosh interface, words on menus are sometimes greyed out. We might assume that the greying of a menu word is supposed to reduce the possibility that the action "read me" is perceived as an affordance of the word.

For many reasons then, there is a considerable gulf between the ease of using highly interactive devices and the difficulty of using keyboard-based

devices. In contrast to keyboard-based devices, display-based devices constrain the next action that the user performs by providing cues to the next action. During performance, this means users can examine the display in the expectation that it will cue a memory of the required action, and during learning they can examine the display in the expectation that it will provide an action that can be semantically matched to the task.

Meaningfulness

Command names are usually meaningful. Most designers attempt to select names that carry some of the semantic content of what the task achieves. This selection is not easy. Users bring word meanings from everyday usage and must adapt their meanings to their precise usage in the device. The correspondence may not always be immediately apparent. For example, Payne et al. (1990) reported a series of experiments that investigated the effectiveness of different menu labels in helping users to infer the operation of a copy/paste buffer. They found that replacing the word *copy* with the word *store* significantly improved the efficiency with which users inferred that a text item stayed in the buffer after having been pasted into the text. Clearly, subtle differences in the meaning of labels are important to how easily a device is learned.

Semantics are used not just in the inference of the effects of a command, but also in determining which of a host of possible words to use to communicate some task to the device. For example, a user may describe a task as, "search for information about Smith," but the device may require it described as, "find record Smith." Given a task description, a user must translate it into the words used by the device.

However, users frequently fail to achieve this translation (Furnas, Landauer, Gomez, & Dumais, 1987). Furnas et al. referred to this failure as the *vocabulary problem.* In a series of laboratory and field experiments, Furnas and colleagues found that if a designer chooses a command name purely on the basis of a personal and intuitive preference, then the chance that a user would generate the same command name on the first try would be between 10% and 20%. This chance is very low, and gives the user of a system a chance of between 80% and 90% of failing to guess the correct command name.

In addition, Furnas et al. tested the success of designing a command name empirically, by determining the command name that new users most frequently generate without being instructed. They found that although there was an improvement over a designer's personal preference, empirically determining a command name still resulted in a 65% to 85% chance of a user failing to guess the right name.

As Furnas et al. observed, people very rarely agree on the words that they use to describe things. Consider as an example the task to get rid of a file,

which is implemented with the word *remove.* Furnas et al.'s results suggest that most new users would not generate the word *remove.* Although some undoubtedly will, others will generate *cut,* or *get rid of,* or *rub out,* or *delete,* or one of a host of other possibilities. We can say that the vocabulary problem is a bound on a user's ability to generate a particular word given a task description.

Even though a command name may not be generated by new users, the use of a meaningful name can improve the probability that a user will remember it once it is known. For the task to get rid of a file, most users will find it easier to remember the command name if it is *remove* than if it were some word without meaning in this task context, for example, *rhino.* This claim becomes truer as the number of task-action mappings increases. If, for example, all actions in a word processing language consisted of typing an animal's name, then considerable confusion will arise. A user may be able to remember which animal names are used somewhere in the language but not which name is used for which task.

Lastly, we can draw a distinction between learning by rote and learning with understanding. Not surprisingly, students who learn with understanding have better recall and transfer of the studied material (Chi, Bassok, Lewis, Reimann, & Glaser, 1989; Lewis, 1988). If a user is given a task to action mapping, perhaps by instruction, then that user's ability to learn the role of each individual action in achieving the overall task will be affected by the everyday meaning of the words used in the actions. Chi et al. (1989) found that students who produce self-generated explanations during problem solving perform much better on posttrial tests. They found that "Good" students learn with understanding. In learning task-action mappings, learning with understanding means learning by associating the individual actions of a mapping with particular features of the task. For example, if a user is told that the action sequence for a task to encrypt a file is type "code -o filename," then they will be able to determine that the word *code* communicates the encryption aspect of the task. This association can be determined using the meaning of *code* and *encrypt.* In contrast, determining the role of "-o" is much harder. This is because "-o" has no everyday meaning on which the user can draw in order to determine its role.

TASK-ACTION LEARNER

How then does Task-Action Learner (TAL) model how people perform when learning and using consistent, interactive, and meaningful devices? TAL is a computationally implemented process model. It models a user who starts with knowledge of the primitive interface actions, for example, how to use a keyboard and a mouse, but without knowledge of how to combine these actions into sequences that will achieve new goals. Given a task, TAL interacts with a device simulation and an instructor (Figure 1). The display of the

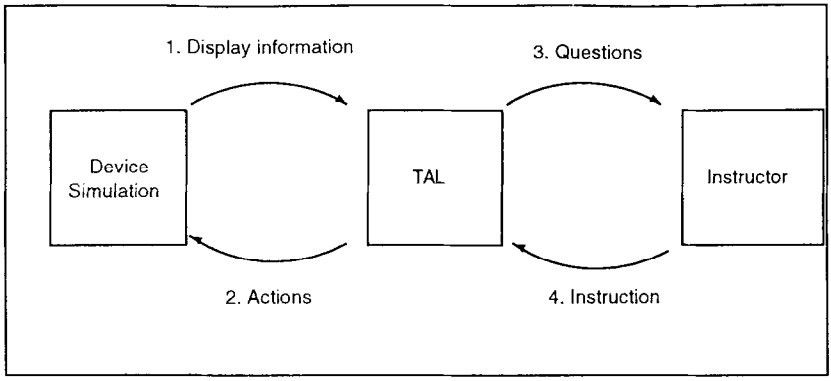


Figure 1. TAL interacts with a device simulation and an advisor

simulated device is examined by TAL to provide information about the display state (item 1 in Figure 1). This includes information such as which menus are visible, what the contents of those menus are, and what the status of the mouse button is (i.e., whether it is being held down or not). If TAL has a rule that determines an action to be performed on the device then it may go ahead and perform that action (item 2 in Figure 1). If, on the other hand, it does not know which action to do, then it requests instruction from a human instructor (item 3 in Figure 1). The instructor responds by giving an instruction (item 4 in Figure 1). TAL interprets the instruction, uses it to determine its next action, and then performs this action. In this process TAL learns chunks, or rules that encode an "episodic" memory of the interpretation of the instructions.

TAL will be illustrated with examples taken from two devices. The first is the Microsoft Word 4 word processor (henceforth MS Word). TAL will be introduced with a description of its performance on a range of file manipulation tasks with this highly display-based device. The second interface is the Unix cshell command language. TAL's performance on this highly keyboard-based interface will be described and contrasted to its performance on MS Word.

In what follows, we refer to the model as having "knowledge" of how to achieve task T if when attempting to execute T the model can produce the required sequence of actions. This behavioral definition of knowledge is taken from Newell (1982, 1990).

Learning Display-Based Task-Action Mappings

A user interacts with the MS Word application via a keyboard, mouse, and display. Moving the mouse on the tabletop causes a pointer to move over the display on which there are a number of visually presented objects. They include an area showing the text of the file being edited, and a number of

different types of menu item. A menu item can be selected by moving the mouse pointer to it and then pressing, clicking, releasing, or double-clicking the button on the mouse. Which action is used depends on the type of menu item to be selected. Types of menu item include menu-bar items, pull-down items, file names, and buttons.

Imagine that a user is given a task of showing the contents of a document in MS Word and that this file contains an invitation. We refer to this as the show-invite task. Executing the task involves the sequence of actions: Move the mouse pointer to the "File" option on the menu bar; press (and hold down) the mouse button; move the pointer to the "Open" option on the pull-down menu; release the mouse button; move the mouse to the filename "invitation": double-click the mouse button.[1]

In TAL, task such as show-invite are represented as a set of attribute/value pairs that capture the semantic content of the task. The task to show a document is represented as:

T1: Task [Effect = make-visible,
Class = text-object,
Contents = invite,
Location = internal-disk,
Size = one-page].

Collectively, these features model what the user believes needs to be communicated to the device in order for the device to achieve what the user desires. As we will see, the set of features does not have to contain only those features that will actually be communicated in the action sequence, nor must it necessarily contain all of the features that need to be communicated. The modeled user has not tried this task before and is ignorant of what the device demands.

Taking Instruction

Figure 2 illustrates a trace of TAL's behavior when given the show-invite task (T1). Some lines of the trace consist of a description of the contents of the device display (flagged "D," e.g., lines 1 to 3, Figure 2). Initially, the display contains some menu-bar items, the mouse button is "up" (i.e., unpressed), and the mouse pointer is on the workspace (lines 1 to 4, Figure 2). Other lines of the trace consist of a description of the task (flagged "T," e.g., line 4), a description of an action performed by TAL on the device (flagged "A," e.g., line 8), or a description of a request for instruction (indented and flagged "Help?," e.g., line 5). The analyst's response to a request for instruction is shown in a bold font. In this example there are 13 requests for instruction. Each request has been made because there was insufficient knowledge to determine the next action.

[1] The sequence of display screens during the execution of this task can be found in Appendix A.

```
1.      D: menu-bar-item Apple, File, Edit, View, Insert, Format...
2.      D: mouse-button up
3.      D: pointer-at workspace
4.      T: achieve Effect=make-visible, Class=text-object, Contents=invite, Location=...
5.              Help? select-with-mouse file
6.      T: communicate file Class=text-object
7.              Help? move-to file
8.      A: move-to file
9.      D: pointer-at file
10.             Help? press mouse-button
11.     A: press mouse-button
12.     D: mouse-button down
13.     D: pull-down-item New, Open, Close, Save, Save-as, ...
14.             Help? finish
15.             Help? select-with-mouse open
16.     T: communicate open Effect=make-visible
17.             Help? move-to open
18.     A: move-to open
19.     D: pointer-at open
20.             Help? release mouse-button
21.     A: release mouse-button
22.     D: mouse-button up
23.     D: invitation, news.old, letter, open, close
24.             Help? finish
25.             Help? select-with-mouse invitation
26.     T: communicate invitation Contents= invite
27.             Help? move-to invitation
28.     A: move-to invitation
29.     D: pointer-at invitation
30.             Help? double-click mouse-button
31.     D: contents-of-invitation
32.             Help? finish
33.             Help? finish
```

Figure 2. Trace of TAL learning the show-invite task.

The task is the show-invite task. The first instruction given by the analyst (line 5, Figure 2) is to "select-with-mouse file," which is interpreted as meaning that the next subtask should be to select the file item on the menu bar (line 6, Figure 2). Subsequently, on line 7, without the knowledge of how to do "select-with-mouse file," TAL again requests instruction.

On this occasion, the instruction is to move the mouse-pointer to the file item (line 7, Figure 2). TAL immediately performs this action in line 8, and the simulation of the device responds by moving the mouse pointer to the "file" item on the display (line 9, Figure 2). Then in line 10, the instructor inputs into TAL the instruction to "press mouse-button," and in line 11 this action is immediately executed on the device simulation. The devices state changes accordingly (lines 12 and 13, Figure 2): The status of the mouse-button changes to "down" and the new pull-down menu items are, "New, Open, Close." Next, in line 14 (Figure 2), TAL is instructed to "finish" the select-with-mouse file subtask.

This process of taking instruction, performing the instructed actions, and reading the simulated device state continues until the show-invite task

has been completed. The 13 requests for instruction that are required consist of four instructions to select the "file" item, four to select the "open" item, four to select the "invitation" item, and one to finish the task.

That TAL can achieve a single task given such detailed instruction is not surprising. What is interesting is how the model interprets and generalizes from this instruction.

Interpreting Instructions

TAL models a human user who, when given an instruction, first interprets that instruction: The learner determines why, in terms of the task, the instruction was given. Huffman (1994), Newell (1990), and Lewis, Newell, and Polk (1989) all described Soar models of instruction taking and interpretation.

If given the example task, show-invite, and the instruction "select-with-mouse file," TAL's interpretation involves using the meaning of the words in the instruction to determine that the instruction was given because the task involves manipulating a text-object, and not because the task involves an invite (see task T1).

The advantage to the user of interpreting the meaning of instructions in terms of the task is threefold. First, it supports a check that communication of the instruction was successful, that is, that the user has understood what the instructor intended and that the instruction "makes sense." Second, it allows the user to generalize the conditions under which the instruction is relevant (cf. Mitchell, 1986), for example, in the show-invite example, the "move-to file" instruction is relevant for all tasks to do with documents regardless of their contents. And third, by processing the instruction "more deeply" the user is more likely to be able to recall it when it is required (Chi et al., 1989). We will show how TAL captures the last two of these advantages, generalization and memorability.

In TAL, interpreting instructions involves matching the task description to the instruction using a a rule base of semantic links between features of the task and items on the display. The rule base models a user's everyday semantic associations between words, for example, the association between the words *text-object* and *file*. It is accessed via a function called lexical-match that takes two parameters: A semantic feature of the task SF and a lexical item LI. The function returns "true" if the lexical item LI can be used to communicate the semantic feature SF, and "false" otherwise.

Lexical-match does not model the representations and algorithms by which people determine the usefulness of lexical items, but merely specifies what the input and output requirements of the algorithm are. This is an example of what Marr (1982) referred to as a computational theory. Figure 3 illustrates a few example matches in the rule base used for learning MS Word. For example, Rule 1 indicates that there is a match between the task feature Class = text-object and the word *file*. This list models a hypothetical individual user. We know that the set of words that users associate with a

1. lexical-match(Class=text-object, Word=file) → true.
2. lexical-match(Effect=put-away + Word=file) → true.
3. lexical-match(Class=text-object + Word=document) → true.
4. lexical-match(Effect=make-visible + Word=view) → true.
5. lexical-match(Effect=make-visible + Word=open) → true.

Figure 3. Examples of lexical-match rules.

particular task varies immensely (Furnas et al., 1987) and later in this article we consider the consequences of the presence or absence of these associations; that is, we model both the user who believes that there is a lexical match between some task feature and some word and the user who does not.

Learning From Interpreted Instructions

When the instruction "move-to-file" is given, using lexical-match TAL interprets the instruction and finds an association between the instructed word *file* and the task feature Class = text-object. It also notices that the word *file* appears on the device display. It now has the information required to learn a new rule. As a Soar model, TAL creates rules, or chunks, that code an "episodic" memory of problem-solving experience while taking and interpreting instruction. Chunks are rules that consist of a left-hand side of conditions and a right-hand side action. For example, from the "move-to file" instruction TAL learns a chunk of the form:

C1: IF the task contains the feature Class = text-object
& the item "file" is on the display
THEN move the mouse pointer to the item "file."

Chunk C1 encodes the conditions under which the instruction should be used and once acquired is sufficient for TAL to perform the "move-to file" action of the show-invite's task-action mapping. However, although C1 will eventually be acquired it cannot be learned immediately. To explain why, we need to make a short diversion to look in more detail at how Soar's chunking mechanism determines the conditions in the left-hand side of the rule and the action in the right-hand side. This diversion will reveal the data-chunking problem, which was first observed by Rosenbloom, Laird, and Newell (1987, 1989).

Chunking can be viewed as a mechanism that caches the results of problem solving so that on future occasions the results can be retrieved without further processing. As an example consider the task to compute the sum of a list of numbers: Compute-sum of [1,2,3,4]. Imagine that the algorithm used consists of starting with a sum of $S = 0$ and then adding the value of each number in the list to S. S would first take the value 1, then 3, then 6,

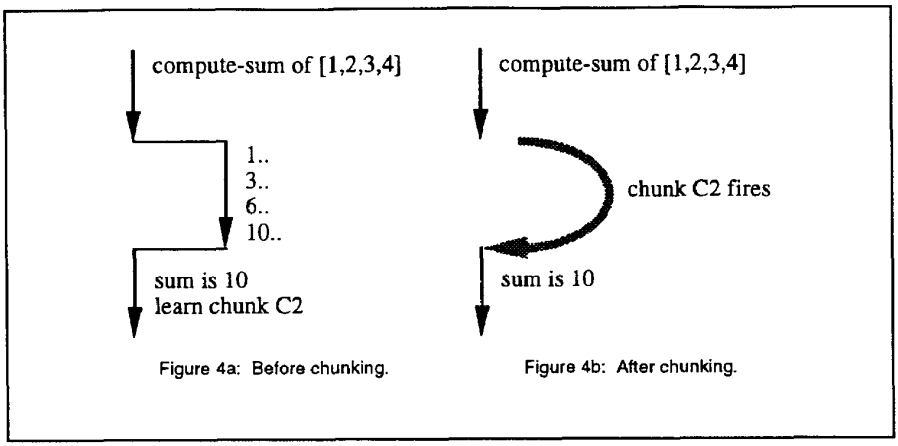


Figure 4a: Before chunking.

Figure 4b: After chunking.

Figure 4. Chunking encodes an episodic memory of problem-solving activity.

and lastly 10, which would be returned as the result. This process can be visualized with Figure 4a. It results in the acquisition of the chunk:

C2: IF the task is to compute-sum of [1,2,3,4]
THEN respond "sum is 10."

The action or right-hand side of chunk C2 is to return the result "sum is 10." The chunk's conditions, or left-hand side, consist of all of the items in the initial state that were required to find the result, which in this case means the function name and the numbers that were summed.

The acquisition of chunk C2 buys the problem solver an efficiency advantage. The chunk is stored in a parallel recognition memory and on future occasions it will propose its answer as soon as its conditions are met, thereby obviating the need to recompute a result by summing each individual number in the list. This process is illustrated with Figure 4b: The calculation necessary in Figure 4a has been replaced by the firing of chunk C2.

The chunking mechanism models a person who codes an episodic memory of their experience computing sums. Further, in the terms of Explanation-Based Learning (Mitchell, 1986) the chunk is an operationalized form of the knowledge of how to compute sums: It improves the efficiency of the problem solver.

The conditions of chunk C2 are an example of the consequences of one of the central hypotheses of the Soar problem-solving architecture:

H1: The conditions of a chunk that returns a result R consist of all of the items in the state used in order to form R.

This hypothesis has what at first may appear to be surprising consequences. In particular, constructing chunks from instruction is not as straightforward as it might seem. Consider again the situation in which TAL is given the instruction "move-to file" (Figure 2). From the instruction, the model can determine that the action required is to move the mouse pointer to the "file" item on the display. It can therefore learn a rule that has this action as its right-hand side.

But what will the conditions of the rule be? In accordance with hypothesis H1, TAL determines the conditions of the rule by gathering together the information that contributed to the proposal of the action. In the compute-sum example, these conditions were the function name and the numbers that were summed. These can be thought of as the reasons that a sum of 10 was proposed. However, in the "move-to file" example, the reason that the chunk proposes to move the mouse to the "file" item is because TAL was instructed that this was the right thing to do for the show-invite task and because the model has successfully provided an interpretation, or rationale, for the instruction in terms of the task and the state. Therefore, rather than chunk C1, chunking actually learns:

C3: IF the task contains the feature Class = text-object
& there is an instruction "move-to file"
& the item "file" is on the display
THEN move the mouse pointer to the item "file."

This chunk is not what is required and it raises a problem. In fact the chunk appears to be useless as it will not recall the required action—"move the mouse pointer to the file item"—given just the task description and the display state. Instead, it will only fire when there is an instruction present. These conditions seem to imply that despite the fact that TAL is instructed how to do the show-invite task, it will continue to require instruction on subsequent trials.

We refer to chunk C3 as an instruction-based chunk. Chunks like this form whenever instruction is used to determine the right-hand side of a rule. The apparent uselessness of instruction-based chunks appears to suggest that although chunking is good at learning the results of computations that the problem solver already knows how to do (e.g., sums), it is unable to learn new rules that do not derive directly from knowledge the system already has. Fortunately, Rosenbloom and Aasman (1992) and Vera, Lewis, and Lerch (1993) articulated a different possibility.

Reconstructing Instruction

The problem is how to use instruction-based chunks when there is no instruction available. The first part of the solution is to realize that these chunks can be used to recognize instructions that have been given on a previous occasion. When an instruction, such as "move-to file" is given for

the first time, TAL must problem solve in order to construct an instruction-based chunk C3. But if the model is given the same instruction for a second time, the chunk C3 will fire, obviating the need to redo the interpretation. The fact that the chunk has fired can be used by TAL to infer that the "move-to file" instruction had already been given on some previous occasion; that is, to recognize the instruction.

Rosenbloom and Aasman (1992) proposed that in the absence of external instruction the solution to recalling what was instructed is to reconstruct the instruction internally; that is, to guess what the instruction might have been and then hope that the instruction-based chunk fires, thereby "recognizing" which instruction had previously been given. The question is how to construct appropriate guesses.

In previous work on task-action mapping we have reported models that use the display as a generator of possible actions (Howes & Payne, 1990; Howes & Young, 1991). Vera et al. (1993) tied this idea into Soar as a solution to the reconstruction of instruction. The idea is that the display items are examined until an instruction-based chunk fires. TAL uses this solution: It models a user who proceeds by examining each display item in turn (e.g., "Apple, File, Edit...") and internally "imagining" that an instruction had been given to select that item. Display items are examined until the conditions of an instruction-based chunk are met, and the chunk fires, recognizing the display item. In this way the display of the device acts as a "reminder" of the content of the instructions.

For example, if having learned a set of instruction-based chunks, TAL is starting the show-invite task for a second time, then there are two possible courses of action: Either ask for instruction again, or attempt to remember an instruction by generating and recognizing possible instructions. Imagine that the model chooses to generate and recognize and it selects as a generator the labels on the display. First, it examines the "apple" menu item and constructs the instruction, "move-to apple," but no instruction-based chunks fire. Next it examines the word *document* and constructs the instruction, "move-to document," but again no instruction-based chunk fires. Next it examines the word *file* and constructs the instruction, "move-to file." For this instruction the conditions of chunk C3 are met, the chunk fires, and proposes the action of moving the mouse to the "file" item. The required action is thereby recalled by the use of a generate and recognize algorithm (Figure 5a).

In this process a new chunk is formed that operationalizes the problem solving that has found the "move-to file" action. This time there was no instruction, only an internally imagined instruction. What did contribute to the formation of the chunk was the display and the task, or to put it another way, the function was to compute an action given the Task and Display. The new rule is therefore, C1: If the task contains the feature Class = text-object and the item "file" is on the display then move the mouse pointer to

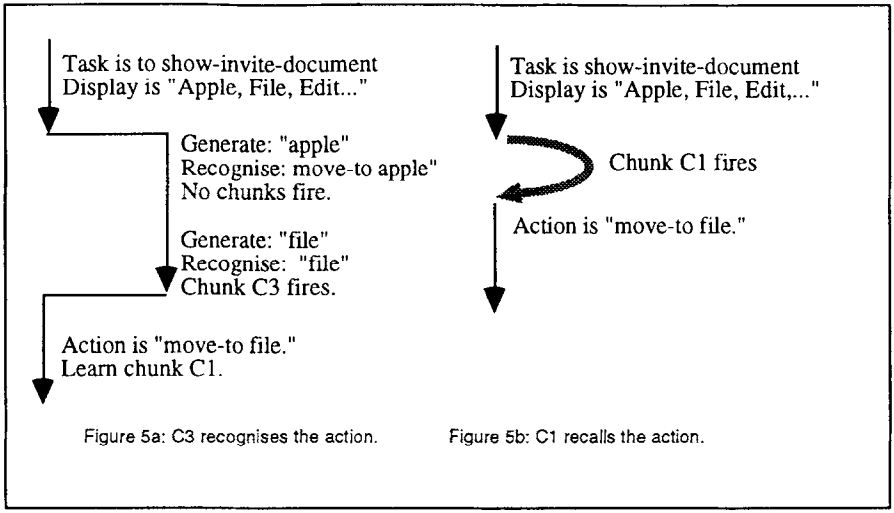


Figure 5. Learning to recall an instructed action by generating and recognizing candidate actions.

the item "file." Chunk C1 does not require the instruction to be present in order for it to fire (Figure 5b). It is no longer instruction based, but as the display was used to generate the action, the new rule is display based.

Reconstructing Button-Action Instructions

We have seen that menus can be used as a generator for reconstructing instructions such as "move-to file." Similarly, the physical input devices, such as the mouse, its button, and the keyboard, can be used to reconstruct instructions such as "press mouse button." The actions for manipulating these input devices can be generated from their physical properties, or affordances (Gibson, 1979). An object's affordances are the functions or uses that its physical properties, as perceived by the user, suggests that it has. There are a number of different actions that are afforded by the Apple mouse button: It can be pressed (and held in), released, clicked, (a press, followed quickly by a release), or double-clicked.

The instruction-based chunks for these kinds of action take a similar form to the instruction-based chunks we have so far seen. For example, the chunk learned from the "press mouse button" instruction is:

C4: IF the task involves using the mouse
& there is an instruction "press mouse button"
& the mouse-pointer is over a menu-bar item
THEN press the mouse button.

The chunk is similar in that the action has been associated with the task using lexical-match (mouse button matches to mouse) and in the inclusion of

the instruction in the conditions. It is different in that unlike the instruction "move-to file," the instruction "press mouse button" could not be interpreted in terms of the semantic features of the overall task (neither the word *press* nor the word *mouse button* has any semantic link to the features of show-invite). Instead TAL interpreted the instruction in terms of the items in the current display state, which in this case is the item under the mouse-pointer. This interpretation models a user who knows the relationship between the mouse, mouse button, and mouse-pointer, but not exactly what the effects of each action will be on each kind of display item. This latter information is acquired from the instruction and encoded in the chunk.

Reconstruction of the "press mouse button" instruction involves examining each afforded mouse action (press, click, double-click) until chunk C4 fires. As a result a new chunk is built:

C5: IF the task involves using the mouse
& the mouse-pointer is over a menu-bar item
THEN press the mouse button

Transferring Instruction

Chunks learned for one task may transfer to many similar tasks. The more similar the task semantics then the greater the transfer. For example, because chunk C1 is sensitive to the Class = text-object feature, it will transfer from the show-invite task to all of the tasks that include this feature. These include closing a file, creating a new file, and saving a file. As a consequence of the transfer, the "file" menu will be selected in all of these tasks. In this way, chunk C1 captures MS Word's consistent use of the "file" menu to communicate the Class = text-object task feature.

Some chunks are even more generally applicable than C1 and will transfer to a broader set of tasks. For example, chunk C5 encodes the action for selecting a menu-bar item. While it has been learned in the context of selecting "file," it will transfer to all task-action mappings that involve the use of the menu bar, including tasks as diverse as checking the spelling and changing the font. We can say that chunk C5 captures the consistent use of "press" for selecting menu bar items.

A significant effect of transfer on TAL's behavior is that if the transfer is successful then it will require less instruction to learn a new set of task-action mappings. Figure 6 illustrates the number of instructions required for various tasks in MS Word with and without transfer from the show-invite task. Appendix B lists the chunks learned by TAL while performing these tasks.

Summary

We have seen that to learn the task-action mappings of a display-based device, TAL learns by requesting and interpreting instructions using lexical-

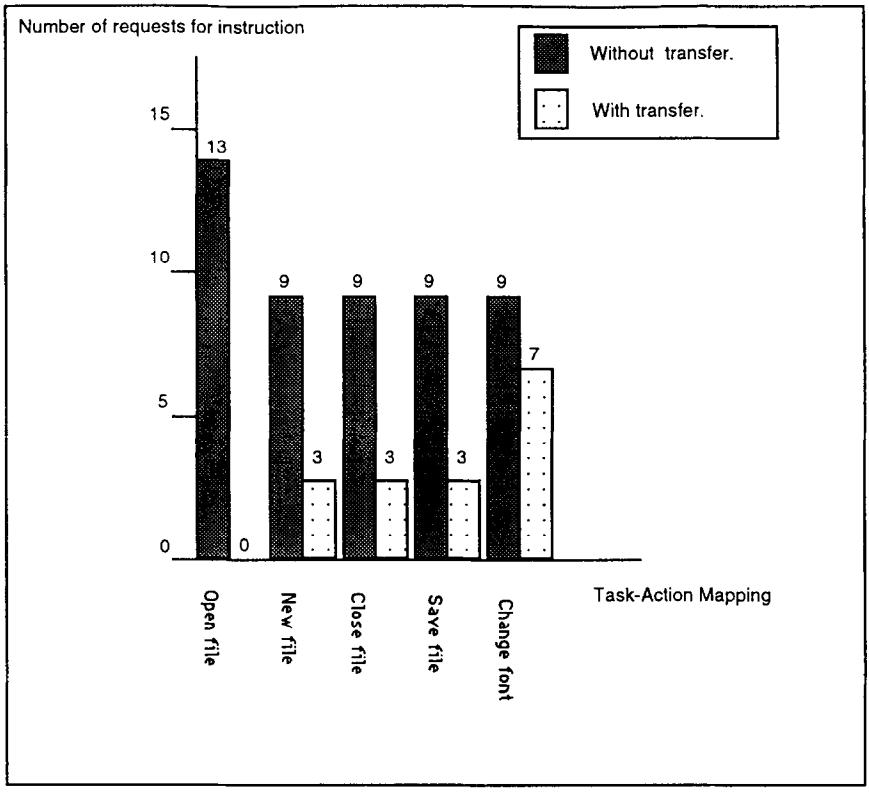


Figure 6. Transfer from the show-invite task to other MS Word tasks.

match; acquires instruction-based chunks; reconstructs the content of instructions by generating candidate actions and recognizing which has been instructed; and lastly, transferring instruction to similar tasks. These properties contribute to the model manifesting behavior that takes advantage of the consistency, interactivity, and meaningfulness of the MS Word file manipulation mappings:

1. TAL takes advantage of the consistency of the MS Word task-action mappings by transferring knowledge learned for one task to other similar tasks. Behaviorally this manifests itself in fewer requests for instruction (Figure 6). Not all interfaces support transfer between task-action mappings as productively as MS Word. In the following sections, we show that when TAL is learning an inconsistent interface, it either requires more instruction or it makes errors because the rules that are learned are too general.

2. TAL uses the interactive properties of the device as a source of candidate operators. In particular, the display is used as a generator or "reminder" of what action was instructed. Not all interfaces are as interactive as MS

Word. In the following sections, we show how when TAL is learning an interface that does not provide the user with display-based cues to the available actions, the model is forced to use other less effective sources of candidate actions.

3. TAL utilizes the meaningfulness of task-action mappings to interpret instructions and "explain" why they were given. This supports the acquisition of more general chunks and their subsequent transfer. In particular, where instructed menu selections were related to a specific feature of the task, then the acquired chunk is general enough to be recalled for all tasks containing the feature, regardless of the other task features.

To appreciate the full implications of these three claims it is necessary to compare TAL's performance on the MS Word task with its performance on other less learnable interfaces. We will show that with less learnable interfaces, TAL makes errors, requires more instruction, and takes longer to reconstruct instruction.

Learning Keyboard-Based Task-Action Mappings

The Unix cshell provides the user with a command-line prompt (usually a short sequence of characters, e.g., "csh%"), followed by a cursor (a flashing rectangle). The screen is otherwise blank. Characters typed by the user on the keyboard appear on the display at the position of the cursor and the cursor is moved right one position. Task-action mapping consists of typing the particular sequence of characters required for that task. Some feedback may be given once the character sequence has been completed, but otherwise the only feedback is the appearance on the display of the typed characters.

As with the MS Word example, we illustrate TAL's performance on this device with a range of file manipulation tasks. We start with the task to wipe from the computer's memory a document containing a list of addresses. We refer to this task as wipe-address-file. It can be described in semantic features as:

Task [Effect = wipe, Class = text-object, Name = address-list, Version = old]

The action sequence required to achieve this task is: "type rm, type 'space,' type home.address, type 'carriage-return.'" TAL models a user who starts without knowledge of this action sequence, and learns it by taking instruction. Figure 7 is a trace of the instruction taking episode. The trace is in the same format as that for the MS Word show-invite task: Lines that begin "D" describe the display state; lines that begin "T" describe the task; lines that begin "A" describe actions performed by TAL; and lines marked "Help?" describe requests for instructions.

The first instruction (line 3, Figure 7) is to use the word *remove*. As before, TAL calls the lexical-match function to interpret the instruction and

```
1.      D: prompt "%"
2.      T: Effect=wipe, Class=text-object, Name=address-list, Version=old
3.              Help? use-word remove
4.      T: communicate=Effect
5.      T: communicate=Effect, word=remove
6.              Help? use-abbrev first-two-consonants
7.      T: communicate=Effect, word=remove, abbrev=ftc
8.      A: calculate ftc
9.              Help? use keyboard
10.     A: type rm
11.     D: "% rm"
12.             Help? finish
13.     T: Effect=wipe, Class=text-object, Name=address-list, Version=old
14.             Help? use-word home.address
15.     T: communicate=Name
16.     T: communicate=Name, word=home.address
17.             Help? use-abbrev whole
18.     T: communicate=Name, word=home.address, abbrev=whole
19.     A: calculate whole
20.             Help? use keyboard
21.     A: type home.address
22.     D: "% rm home.address"
23.             Help? finish
24.     T: Effect=wipe, Class=text-object, Name=address-list, Version=old
25.             Help? type return
26.     A: type return
27.     D: "%rm home.address
28.        %"
29.             Help? finish
```

Figure 7. Trace of TAL learning the wipe-address-file task.

determine that the instruction was given in order to communicate the Effect = wipe feature of the task. However unlike the MS Word, because the instruction could not be matched to the display, TAL uses the interpretation of the instruction in a different way: It sets its current subtask to be communicate the Effect feature with the word *remove.* The implications of this difference are discussed later. The next instruction (line 7, Figure 7) is to take the "first-two-consonants" of the word *remove* (i.e., "rm"), and then type these consonants on the keyboard (line 9, Figure 7). This last action is performed in line 10 and the changes to the display state are described in line 11. Having typed "rm," the mapping of the Effect feature is finished. The rest of the instructions detail how to communicate the Name of the file.

Obviously these instructions are different to those for the MS Word show-invite task. They are, after all, for a different task and device. Rather than directing TAL to use a mouse to select menu items, they direct it to use the keyboard to type commands. In addition, some instructions (e.g., "use-abbrev first-two-consonants") direct TAL to perform internal computations rather than immediate external actions.

Despite these differences, learning from instruction still involves an interpretation process and is still subject to the constraints of Soar's chunking mechanism. On this first trial, TAL learns a set of instruction-based chunks

(rules that code TAL's interpretation of the instructions) and subsequently, learning to achieve this task without instruction is based on using instruction-based chunks to recognize the required actions.

Reconstructing Instructions

For MS Word, TAL used the menu items on the device display and the physical affordances of the mouse for reconstructuring instruction. The display of a keyboard-based device, such as the Unix cshell, can also, in principle, be used to generate candidate actions, although in a far more impoverished way. Consider the Unix command that displays a list of file names. After the system has displayed this list, the user can examine the individual file names. In this circumstance, a user who did not know the exact name of a required file could use the display as a reminder. However, although the display may be used in this way in some circumstances, it is more often the case that the user of a keyboard-based device must generate required actions without display support. Here we therefore assume that the display is not used as a generator by users working with the Unix cshell.

The use of the display in reconstructing instructions is an example of the use of an external generator. The display is external to the problem solver's memory. In contrast, to reconstruct instructions without the display requires the use of internal generators. These draw on knowledge stored in the problem solver's long-term memory. Examples include the ability of people to generate synonyms of a word (e.g., remove, delete, rub out, etc.) and the ability of people to generate the letters of the alphabet. TAL makes use of a number of internal generators in order to reconstruct instructions for keyboard-based devices. These include generators of words from the task features and generators of abbreviations.

To learn a keyboard-based task, TAL applies internal generators hierarchically. The modeled user works top-down from the task description, decomposing it into finer grained subtasks until an action that can be performed on the device is found. In contrast to the display-based reconstruction used for MS Word tasks, this hierarchical decomposition is an example of an interleaved plan-based performance. It is interleaved because the plan is constructed depth first and each external action is executed as soon as it is determined (see Young & Simon, 1987). The following sections examine each of the internal generators. They are presented in a top-down order from the task description to the actions on the device.

Generating Subtasks to Communicate Each Feature. Consider again the situation in line 3 of Figure 7 where TAL is given the instruction to use the word *remove*. In response to the instruction, the model uses lexical-match to determine that *remove* is to be used to communicate the Effect = wipe feature, and as a consequence of this interpretation the model sets the subtask to "communicate Effect." In the process it learns the chunk:

C6: IF task contains the feature Effect
& the instruction is to use a word to communicate the Effect
& there was no previous subtask
THEN make the subtask to communicate Effect.

This chunk is of a form that we have not yet seen. Its action identifies only the relevant features of the task (i.e., "Effect"). It does not identify the full content of the instruction (i.e., "remove"), nor the value of the feature (i.e., "wipe"). In addition, the chunk is conditional on the previous subtask. For C6 to fire there should be no previous subtask, which means that "communicate Effect" is the first subtask of any task containing an Effect feature. As we will see, during reconstruction, chunks such as C6 will ensure that TAL can determine the order in which the features of the task need to be communicated to the device. In other words, these chunks encode the ordering of the task-action mapping.

TAL does not use the acquisition of chunks such as C6 in the model of the MS Word user. We suspect that whether or not people learn these chunks for display-based devices, they are redundant when a display is available as a generator. The reason for this is that display-based devices often determine the order in which task features can be communicated by constraining the set of available menu labels. Under these conditions, the user does not have to recall a syntax for the task-action mapping because the device, in effect, provides it. For example, in the show-invite task in MS Word the device ensures that the task feature Effect = make-visible must be communicated before the feature Contents = invite.

Chunk C6 is generalized in a novel way. Although some of the chunks learned for MS Word were sensitive to just one task feature, they were always sensitive to both the feature name and the feature value (e.g., both "Effect" and "wipe"). In contrast, chunk C6 is sensitive to just the feature name ("Effect"), not its value. It will therefore transfer to situations in which the feature Effect has a different value. For example, if instead of the desired Effect being to "wipe" a file from computer memory it is to "edit" a file or "send" a mail message, then chunk C6 will still be used. To put it another way, the syntax learned for the wipe-address-file task will transfer to all tasks that share the Effect and Name semantic categories. This is an assumption that empirical evidence suggests people make (Payne & Green, 1989). Learning will be more efficient if the assumption is correct. It will reduce the amount of instruction required by TAL.

Reconstructing chunk C6 involves examining each task feature until this instruction-based chunk fires. For example, first the Name feature of the task might be examined, but no chunk would fire, then the Version feature, but again no chunk would fire, and then when the Effect feature is examined, chunk C6 would fire and recognize it. As before, a new chunk would be formed that was not dependent on instruction.

```
lexical-recall(Effect=wipe) → "destroy"
lexical-recall(Effect=wipe) → "wipe"
lexical-recall(Effect=wipe) → "remove"
lexical-recall(Effect=wipe) → "delete"
```

Figure 8. Example of lexical-recall rules.

Generating Words to Communicate a Feature. Having determined which feature of the task is to be communicated, TAL needs to find the word to carry the meaning of the feature. From instruction TAL has learned the chunk,

C7: IF task is to communicate the Effect
& the instruction is to use-word "remove"
THEN use the word "remove."

To reconstruct the instruction necessary to cue this chunk, TAL employs a function that we call *lexical-recall.* The function is analogous to lexical-match but takes just one parameter—a task feature (e.g., Effect = wipe)—and returns a word that may be used in communicating the feature (e.g., "remove"). Repeated calls to the function result in it returning different words. Figure 8 illustrates a sequence of calls to lexical-recall for the feature Effect = wipe. It first returns the word *destroy,* then *wipe,* then *remove,* and finally *delete.*

As with lexical-match the intention is not to model the details of the process by which people recall words. Instead we assume that: (a) given some description, people can rephrase it using different lexical items, and (b) that the new lexical items cue further rephrasing. For example, the word *air* may cue the word *breathe,* which may cue *lung.* Importantly, the lexical-recall knowledge is not sufficient to determine the correct word to use in order to communicate a particular task feature. The lexical-recall function may return many words, usually only one of which will achieve the user's device task.

Reconstruction of the "use-word remove" instruction involves calling lexical-recall until chunk C7 fires and recognizes the instruction. Following Figure 8, lexical-recall might first return *delete,* and then *wipe,* and then *remove.* Only when *remove* has been generated will chunk C7 fire. This event determines that the word to use for the feature Effect = wipe is *remove.* As a result a new chunk is formed:

C8: IF task contains the feature Effect = wipe
& there was no previous action
THEN use the word "remove"

Chunk C8 does not refer to the instruction in its conditions and therefore supports recall of the required action even when instruction is not available. However, unlike in the MS Word chunks, this chunk is not display based. This is because lexical-recall is an internal generator and does not use the display as a means of generating possible actions.

Generating Abbreviation Functions. Having found a word to communicate the feature, TAL needs to reconstruct the instruction that determines the function to compute the abbreviation of the word. The first abbreviation for the wipe-address-document task is encoded in the following chunk:

C9: IF the task is to communicate the Effect
& there is an instruction to use the first-two-consonants
THEN use the abbreviation first-two-consonants

The function first-two-consonants is one in a set of functions that are regularly used by people to generate abbreviations. The set includes "first-letter," "first-two-letters," "first-three-letters," "first-syllable," and the option of using the whole world. If there are multiple words to be abbreviated then there are also functions that take abbreviations such as the first letter of each word.

TAL generates abbreviation functions with a function called abbrev-recall. Abbrev-recall generates ways of abbreviating a word until one "reminds" TAL of the instruction. When "first-two-consonants" has been generated, chunk C9 will fire as a consequence the following chunk will be learned:

C10: IF the task is to communicate the Effect
THEN use the abbreviation first-two-consonants

Once the abbreviation function is known, the characters required for the mapping may be computed. For the current example, the word to be abbreviated is *remove,* and the first two consonants are therefore *rm.*

Generating Actions. Having found an abbreviation, TAL needs to determine the physical actions used to communicate it to the device. During instruction TAL has learned the chunk,

C11: IF the task is to communicate the Effect
& there is an instruction to use the keyboard
THEN use the keyboard

Reconstruction of this instruction is very simple. TAL models a user who perceives that the keyboard affords typing. The user can therefore use the device as an external generator of the possible physical actions. Once the instruction "use keyboard" has been generated, chunk C11 will fire and TAL will form the following chunk:

C12: IF the task is to communicate the Effect
& there is a keyboard
THEN use the keyboard

Unlike the other chunks learned for this device (C6 to C11), this chunk (C12) is display based.

Summary. We have described a process for reconstructing instructions for how to use a keyboard-based device. The process involves first determining which feature of the task is to be communicated to the device; then determining which word is to be used in order to communicate the feature; then determining whether and how to abbreviate the word; and lastly, computing the abbreviation and typing the characters on the keyboard. Most of the generators used are internal generators; that is, the generators are based on long-term memory as a source of candidate actions.

Transferring Instructions

TAL has been tested on a set of tasks from Unix. These include moving files from one place to another, listing the files in the current directory or folder, and displaying the content of files. Positive transfer occurs to tasks that use the same abbreviations as the wipe-address-file task, for example, the task to display the names of the files in the current folder (List directory). The number of instructions required with and without transfer is shown in Figure 9. It shows that without transfer from another task, TAL requires 10 instructions to learn the "list directory" task but with transfer it only requires 3 instructions. These three instructions specify that the word *list* should be used. The other seven instructions are not required because the action sequence for the "list directory" task, which is "ls Name," shares abbreviations and syntax with the wipe-address-file task. The same is true of the tasks remove file, prepare file, and move file. Their respective action sequences are rm Name, pr Name, and mv Name. For these tasks, instructions are required to determine the words (*prepare, list, move*) but not to determine the abbreviation or syntax.

Despite the consistency of this handful of mappings, for many other Unix tasks there is negative transfer. Negative transfer occurs to tasks that share some of the semantic features of tasks that have already been learned but do not share the same task-action mapping. For example, the action sequence for the task to display the contents of a file, is "more Name."

Beside the fact that for most users the word *more* is unlikely to be derivable from the semantic features of the task, the task-action mapping does not involve abbreviating "more" to "mr." For this reason, this mapping is not consistent with those described by Figure 9. As a consequence, when given the task to display the contents of a file, TAL will make an error if it has previously learned the task-action mapping for any of the tasks in

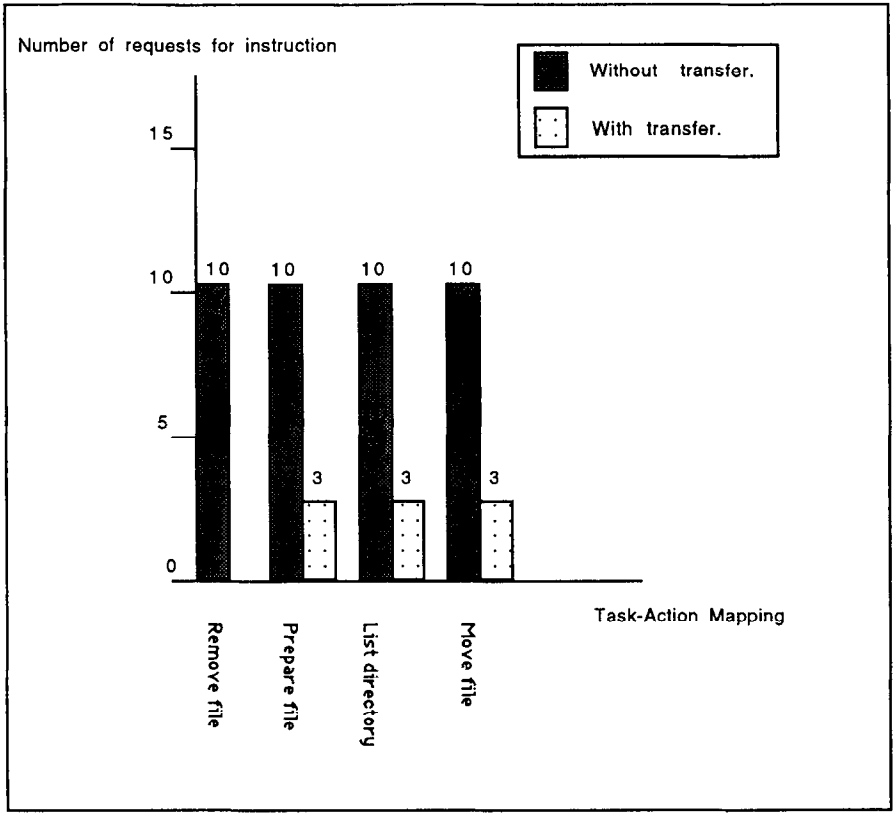


Figure 9. Transfer between Unix tasks.

Figure 9. The error will arise because TAL assumes that the abbreviation "mr" should be used.

Summary

TAL learns the task-action mappings of a keyboard-based device by requesting and interpreting instructions, and by acquiring instruction-based chunks. In this respect it uses exactly the same mechanism for a keyboard-based task as for a display-based task. However, rather than examining the display, the reconstruction of keyboard-based instructions involves hierarchically decomposing the task description using internal generators and in addition learning the order in which features of the task are to be communicated (the syntax).

Although not interactive, the Unix task-action mappings that we examined were consistent and meaningful. This was revealed in TAL's behavior:

- TAL takes advantage of the consistency of the Unix task-action mappings by transferring syntax and abbreviations learned for one task to other similar tasks. Behaviorally, this manifests itself in fewer requests for instruction (see Figure 9). However, we note that if we had examined a broader set of Unix tasks, inconsistencies would have been revealed.
- As before, TAL utilized the meaningfulness of task-action mappings to "explain" why instructions were given and thereby support generalization and transfer. In particular, where instructed words could be related to a specific feature of the task using the function lexical-match then acquired chunk would be general enough to be recalled for all tasks containing that feature, regardless of their other features. This supports transfer and is one of the factors that reduces the need for instruction. In addition, meaningful task-action mappings are those that can be generated using the function lexical-recall.

RELATING THE TAL MODEL TO HUMAN BEHAVIOR

We have described TAL's performance on two interfaces: MS Word and Unix. The model used an identical underlying learning mechanism on both of these devices, yet its behavior was markedly different on each: The number of requests for instruction and the sources used to generate candidate actions differ. In this section we summarize and expand on how aspects of these differences are due to the device design, and how TAL's behavior is consonant with observed human performance according to the interactivity, consistency, and meaningfulness of the task-action mapping.

Interaction

Is TAL consonant with empirical observations suggesting that: (a) expert users of display-based devices have poor recall for task-action mapping knowledge when away from the device (Payne, 1991); (b) the use of display-based devices is best characterized as a back-and-forth information flow (Draper, 1986) and that display-based devices are easier for people to learn and use?

Experts Have Poor Recall Away From Device

Consider the fact that chunks learned when the display is used as a source of candidate actions include the display in the conditions of the chunks (e.g., chunk C1). These chunks are display based and recognition oriented. They cannot be used to recall the task-action mapping without the cues provided by the display.

This makes no difference to TAL's performance on the task when the display is present, but if TAL were expanded to include a model of "report-

ability," that is, how task-action mapping knowledge was communicated from one user to an experimenter in the absence of the device, then there is a clear rationale for claiming that TAL would predict the experimental results of Payne (1991): The display-based chunks will fail to fire in the absence of the display.

Display-Based Devices Are Easier to Learn

To summarize the literature reviewed in the introduction, human behavior is such that display-based devices support learning and performance of task-action mappings by cueing the user as to the available actions. This facility can lead to shorter learning times (over keyboard-based devices), and reduce the number of errors made by a user (e.g., Duff, 1989). To describe how TAL is consonant with these observations we consider: (a) The reliability of internal and external generators; and (b) the physical constraint provided by the display.

Reliability. If a task-action mapping is keyboard based, then TAL uses the lexical-recall and abbrev-recall generators to reconstruct instruction. The human capability that these generators model is unreliable in two respects. First, an individual who generates an item on one occasion might fail to do so on another. This failure may be due to the underlying unreliability of the human cognitive architecture. For example, people sometimes fail to generate the name of a friend, even though on most occasions this task is trivial. Cognitive architectures that can model this kind of unreliability include ACT-R (Anderson, 1993). In ACT-R, there is always a certain probability that any rule will fail to fire. In contrast, in Soar a rule is guaranteed to fire if its conditions are met. For this reason we have chosen not to model the unreliability of the architecture.

The second way in which people are unreliable is that there is no guarantee that an individual, selected at random from a population, will generate the item at all. In this case it is a population of individuals that is unreliable. As we have said, Furnas et al. (1987) showed that there are immense variations in the words that people generate for a particular cue. According to these authors, there is a probability of between .15 and .35 that an individual will generate the same response as is generated by most of the rest of a population. Thus the human capability modeled by the lexical-recall function varies from user to user and from word to word. The probability that an individual within a population will recall a particular item can only be determined by empirical observation.

TAL is parameterized by the responses that are programmed in for the lexical-recall and abbrev-recall functions. Any particular combination of programmed responses models some hypothetical individual user. Therefore it is possible to construct a population of models of individual users.

The models are identical to each other in their underlying instruction taking and chunking mechanisms but vary in the lexical and abbreviation knowledge that they bring to bear when interpreting and reconstructing instructions. Versions of TAL that do include the required responses to lexical-recall and/or lexical-abbrev for a particular device will learn the required task-action mappings (e.g., for the Unix example given earlier). Versions of TAL that do not include the responses required to lexical-recall and/or lexical-abbrev will not be able to reconstruct instruction and will therefore fail to learn the required task-action mappings.

As lexical-recall and lexical-abbrev are used in keyboard-based devices, we can say that the probability P(generate, keyboard-based) that a randomly selected version of TAL will be able to generate the actions of a keyboard-based task-action mapping is less than 1. In contrast, learning display-based devices involves using the display as a generator. In this circumstance, TAL examines every item on the display, and therefore the required action is guaranteed to be generated if it is on that display. (If the required action is not on the display then that task-action mapping is not, by definition, display-based.) Therefore, the probability P(generate, display-based) that a randomly selected version of TAL will generate the actions of an ideal display-based task is equal to 1. This means that all versions of TAL will be able to reconstruct instructions for how to use a display-based device.

Comparing the reliability of the generators used in reconstructing task-action mappings for display-based and keyboard-based devices we see that the probability that a TAL in a population will be able to learn a display-based task is greater than the probability that it will be able to learn a keyboard-based task. In this respect, TAL is consonant with the observation that interactive devices are easier for people to learn than keyboard-based devices. To emphasize the point, however, it is not that every TAL will find a particular keyboard-based task-action mapping harder to learn than an equivalent display-based task-action mapping, it is just that there will always be some versions that cannot reconstruct keyboard-based mappings.

Physical Constraint. When we consider a whole task-action mapping rather than just the use of an individual generator we see another advantage of a display-based device. The set of items on the display changes as the mapping unfolds. For example on the Apple Macintosh, once an item has been selected from a pull-down menu, the menu is removed from the display. The advantage of this change is that it constrains the order in which the actions of the mapping can be communicated to the device; that is, there is no need for TAL to learn the syntax of the mapping. We refer to this as a form of physical constraint.

An important example of physical constraint occurs on the Macintosh mouse. Here the normal generator set for the mouse button is press, click,

double-click. However, one the mouse button is pressed down and held in, the available actions for the next selection are constrained to releasing the mouse button. As TAL uses the affordances to generate candidate mouse actions in this situation, it need consider no candidate other than "release."

Similarly, in MS Word the "open" item must be selected before the name of the file to be opened. In this way the display-based device removes the possibility of syntax errors that are otherwise common in keyboard-based interactions. This observation implies that syntactic consistency is not as important in display-based devices as it is in keyboard-based devices.

Summary. From TAL's behavior on Unix and MS Word we see that: (a) the probability of a version of TAL being able to reconstruct a set of instructions is greater for a display-based device; and (b) the fact that displays change constrains the order in which actions can be performed and may make learning syntax unnecessary. Both of these results suggest that due to constraints of interactivity, TAL finds display-based devices easier to learn than keyboard-based devices.

Consistency

TAL is sensitive to four types of consistency: lexical consistency, display-item consistency, abbreviation consistency, and syntactic consistency. These are described in the following paragraphs.

Lexical Consistency

In terms of TAL, an interface is lexically consistent if the same word is always used to communicate a particular task feature. The interface is lexically inconsistent if a feature needs to be communicated with different words in different circumstances.

We saw that the set of file manipulation tasks examined for the MS Word device were lexically consistent. The consistency manifested itself in the fact that TAL performed the set of acquired tasks without error. However, this is not true for all MS Word tasks. In particular consider a set of tasks (taken from MS Word 5) for changing the font of the selected characters. If TAL is instructed on the task to change the font size from "10 point" to "14 point," then amongst the chunks that it learns will be one that reads:

C13: IF the task contains the feature Class = text – representation
& the item "font" is on the display
THEN select "font"

If in addition it is instructed to perform the task to change the font from "bold" to "italic" then amongst the chunks that it will learn will be one that reads.[2]

[2] Learning chunk C14 involves forcing TAL to ignore chunk C13 during the learning phase.

C14: IF the task contains the feature Class = text-representation
& the item "format" is on the display
THEN select "format"

These chunks will compete for all tasks that involve the feature Class = text-representation. Sometimes chunk C13 will win and sometimes chunk C14 will win. The chunks are not discrimintated by the fact that one should only apply to tasks that involve changing the characters to "bold," "italic," "plain," or "underline," and the other only to tasks that involve changing the characters to "10 point," "12 point," or "14 point." As a consequence, the chunks will cause errors by applying in the wrong context.

Of course this analysis depends on our assumption that the modeled user will represent both types of task ("italic" and "12 point" tasks) as operations on objects of Class = text-representation. However, it is also possible, although we think unlikely, that an individual will represent these tasks with disjoint feature sets; that is, a representation in which the tasks are not semantically related. Although this possibility exists, determining whether it is correct for an individual is beyond the scope of this article. Here we are only interested in the fact that, for people in general, confusions do arise between semantically overlapping tasks that require distinct task-action mappings.

Display-Item Consistency

A task-action mapping must not only be consistent with the task semantics but also with properties of the display. For example, in MS Word all menu-bar items, whether labeled "file," "edit," or whatever else are selected by pressing the mouse button. In this case the consistency is relative to the visual properties of the item, for example, that it is a word written in black letters on a white background at the top of the display. We saw with chunk C5 that TAL assumes that if in one task-action mapping a menu bar is selected by pressing the mouse button then in all other task-action mappings menu-bar items will be selected in the same way.

MS Word supports the assumption made in chunk C5. However, if the device design conflicted with this assumption, then TAL would make errors. An error would consist of pressing the mouse button in the wrong circumstance.

Abbreviation Consistency

Evidence provided by Payne and Green (1986) suggests that users find consistent abbreviations easier to learn. An interface has consistent abbreviations if each task feature is always communicated using the same abbreviation function.

When learning Unix, we saw that the set of task-action mappings examined used abbreviations consistently: All took the first two consonants of the command Effect ("rm," "ls," "mv," and "pr") and the whole of the file Name. However, not all commands in Unix obey this rule. In particular, as

we saw earlier, the command for displaying the content of a file is "more," but the abbreviation used is not the first two consonants, "mr," but instead the whole word, *more*. As a consequence, TAL will make an error on this task-action mapping. We refer to this as abbreviation inconsistency. TAL makes errors if different abbreviation functions are used to map different values of the same task feature.

Syntactic Consistency

Payne and Green (1989) tested participants on a toy property office device using three categories of task: accessing a database, playing games, and mailing colleagues. There were three groups of participants on three different interface languages to this device. Language L1 used just one syntax for all three categories of task, language L2 used a different syntax for each category of task, and language L3 used two syntactic forms that were randomly assigned to tasks of whatever category. Payne and Green found a significant effect of the language design on the number of syntax errors that were made. Users on language L1 made fewer syntax errors than users of language L2 who made fewer errors than users of language L3.

Consider the task to get information about a particular boxer dog from the database. The required action sequence is to type the sequence of words: "DOG BOXER HIGH BROWN" The task can be described as:

> Task [Effect = use-database, Object = dog-animal, Type = boxer-dog, Specifier = tan, Priority = maximum]

Learning task-action mappings for this device is similar to learning task-action mappings for Unix except that there is no abbreviation function. In essence, there are two things to learn: (a) the word to be used to communicate a particular feature and value (e.g., For Specifier = tan use the word "brown"); (b) the order in which the features are to be communicated.

Learning interface L1 is straightforward. There is only one syntax for all tasks, which means that when the syntax of one task is learned it will successfully transfer to all subsequent tasks. The words used to communicate task features also transfer (e.g., Priority = maximum → "high" is used in a number of tasks.)

Learning interface L2 is not so easy. For this language, TAL has to learn three separate syntactic forms: one for database tasks, one for message tasks, and another for tasks involving playing games. The first thing to observe is that if TAL is given the same instruction as it was for language L1 then it will make overgeneralization errors by transferring the instructions to situations where they are not appropriate, and where in fact another syntax should be used. For example, if it is given the syntax for database tasks, then it will assume (incorrectly for L2) that the same syntax should be used for games. The reason for this is that the feature specifying whether the task is a database, message, or game is not used by TAL in constructing the

action sequence. This is because this aspect of the task is not explicitly referred to in the action sequence. There is therefore no reason why the task-action mapping that is learned should be sensitive to this feature. The feature is not part of the interpretation of the instruction and therefore does not appear in the chunk.

Now consider how we might get TAL to learn the three syntactic forms and to correctly discriminate the circumstances in which they are to be used. In order to achieve this, TAL is given the instruction, "context X" that it interprets as meaning "set up a subtask called X." For example, if it is given the task to get information about a boxer dog then it will be given the instruction, "context get-information." To interpret this instruction, TAL will form the chunk:

C15: IF the task involves the feature Effect = use-database
THEN make the subtask to get-information

Subsequent instructions will be interpreted in a way that results in chunks being conditional on the task to "get-information." These chunks will therefore transfer to all tasks that share the feature Effect = use-database but no further.

Using this technique, language L2 can be instructed with three separate syntactic forms: one for use-database tasks, one for message tasks, and one for game tasks. This requires more instruction than for language L1.

Learning interface L3 is harder still. There are only two syntactic forms in this language but they are assigned randomly across the task categories. Therefore to instruct TAL how to achieve the set of tasks, the instructor must specify that each set of instructions applies only to a particular feature such as whether or not the mapping task involves a "dog." Again, this is achieved using the instruction "for X" (e.g., "for dog"). However, where for language L2 only three categories of task-action mapping needed to be learned (one for games, one for database tasks, and one for message tasks), for language L3 a number of categories need to be learned: one for dogs, one for cats, one for cars, and so on. As a consequence, far more instruction must be given than was required for language L1 or L2. In fact there will be no transfer of instruction from one task to another.

Figure 10 (right panel) illustrates the amount of instruction required for each language. It indicates that TAL finds language L1 easier to learn than language L2, and language L2 easier to learn than language L3. This is predictive of Payne et al.'s experimental result indicating that people make fewer errors (Figure 10, left panel) learning language L1 than they do learning L2, and fewer errors learning L2 than they do learning L3.[3]

[3] Although the number of requests for instruction predicts Payne and Green's data, the number of errors made by TAL correctly distinguishes only L1 from the other two languages. This is because the human data reflect learning from errors over a number of trials, whereas TAL learns only from instruction and not from experience.

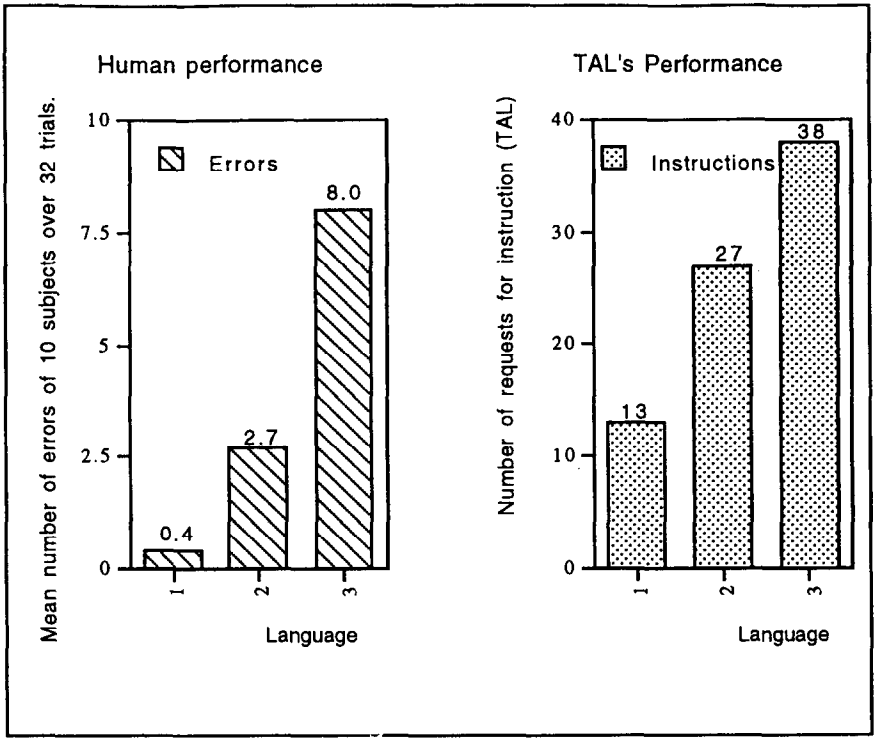


Figure 10. Number of requests for instruction made by TAL and mean number of errors made by human subjects: Transfer between tasks for the lost property device.

Meaningfulness

In the introduction to this article we made the point that to find a task-action mapping meaningful, people must be able to understand the role that individual actions have in achieving the overall task (Lewis, 1988; Payne et al., 1990). Understanding the meaning of the individual actions is critical to understanding the whole mapping. TAL captures this with a mechanism that analyzes each instruction with lexical-match in the hope of finding a link between the words of the instruction and the semantic features of the task. As we have seen, this has a significant effect on TAL's subsequent behavior. If instructions are meaningful then they can be related to the task features and will transfer to other similar tasks (see the earlier description of lexical consistency).

In addition to examining the transfer effects associated with meaningful task-action mappings, we have also reviewed empirical evidence suggesting that task-action mappings that do not use meaningful tokens are less memorable and more confusable than those that do. People confuse meaningless

mappings by using the actions instructed for one task-action mapping for other tasks. There behaviors are captured by TAL in two ways.

First, if lexical-match fails to find a semantic link between an instruction and a task then TAL cannot determine which of the task features the instructed action is intended to communicate. For example, the task-action mapping for storing the content of a file in the Emacs word processor involves the user typing "control-x," followed by "control-s." For this task the best that TAL can do is learn a decontextualized chunk that will recognize that "control-x" has been seen in an instruction. The chunk that is formed will not refer to the task at all. The use of "control-s" can be explained in terms of the task: It is the first letter of the word *store* and of *save*. However, there is no obvious link between "control-x" and the task description. When TAL is instructed with the actions for this mapping then it is not able to interpret the instruction "type control-x," using the task features and as a result it learns the chunk:

C16: IF there is an instruction to type control-x
THEN type control-x

This recognition chunk is overgeneral. During reconstruction it will fire whenever TAL generates the action "control-x." This generation may not occur very often but it is likely to occur for the wrong task as it is for the right task. For example, imagine that given another task with a meaningless mapping, for example, the task to load a Lisp function being edited in Emacs into an interpreter, TAL had learned the chunk:

C17: IF there is an instruction to type control-c
THEN type control-c

Now while reconstructing the task to save the contents of a file, whether chunk C16 or C17 fires first will be dependent on in which order the two actions are generated. As this order is random, it may well be that TAL will generate "control-c" first, and C17 will fire and recognize it. This recognition will lead to an erroneous action: "control-c" will be typed for the save task. Similarly, in reconstructing the task to load a Lisp function, it may be the case that "control-x" is generated and chunk C16 fires. In this case, "control-x" will be typed erroneously.

The second way in which TAL models the difficulty that people have in learning task-action mappings with meaningless tokens is that it may fail to generate actions during the reconstruction phase. This is because the required action may not be derivable from either the task or the display. Consider, for example, the Unix task to find lines in a file that contain a specified text string. This task is communicated to the device with the action sequence: "grep Text-string File-name," where Text-string and File-name are specified by the user. When TAL is given this task and instructed with this action

sequence, it will learn a set of instruction-based chunks. These instruction-based chunks will include a chunk that can be used to recognize the word *grep*. However, unless the function lexical-recall is preprogrammed with an association that generates the word *grep* given the task feature Effect = find, TAL will not be able to reconstruct the action. (It is hard to imagine that the word *grep* would be in the lexicon of a person who did not already know its Unix-specific meaning.) In this circumstance, TAL will again request instruction as to the word to use to communicate the Effect feature of the task. Although TAL can learn to recognize the word *grep*, it cannot learn to recall it. In contrast, with display-based devices, if there is no association between the task semantics and the required item, meaningless items can still be generated by virtue of their presence on the display. In this circumstance, the choice set is less constrained than if the item were meaningful, but at least it still contains the required item.

The most important point to take from this section is that TAL can fail to recall task-action mappings that do not use meaningful actions, where meaningful is defined in terms of the relationship of the tokens used in the actions to the semantic features of the task. If there is no relationship, as determined by lexical-match, then TAL's behavior is marked by the fact that actions become confused (actions learned for one task-action mapping are erroneously used for other tasks), and some actions that were instructed cannot be recalled.

DISCUSSION: INTERPRETING AND RECONSTRUCTING TASK-ACTION MAPPINGS

In this section we consider the two key mechanisms employed by TAL: interpreting and reconstructing instructions. We consider the relationship of these mechanisms to other models in the cognitive science literature and then consider their potential role in a model of instructionless learning.

Interpreting Instruction

The process by which TAL interprets instruction to determine which semantic feature of the task an instructed action is intended to communicate to the device is an example of an Analysis-Based Learning technique (ABL; Lewis, 1988). This class of learning techniques includes explanation-based learning (DeJong & Mooney, 1986; Mitchell, 1986). ABL involves determining why an example is a member of a class by constructing an understanding of the example (e.g., determining that the role of columns in an arch is to support the cross-beam) and then using the understanding to aid generalization. In this respect ABL techniques capture the observation that students who explain examples to themselves rather than learning by rote perform better on posttraining tests (Chi et al., 1989). Further, ABL can often learn useful

generalizations from just one or a handful of examples. In contrast, similarity-based learning mechanisms (Dietterich & Michalski, 1983) require many examples.

For the acquisition of task-action mappings, ABL generalizes by explaining why a sequence of actions achieves a particular task. ABL models of task-action mapping acquisition include EXPL (Lewis, 1988), Explor (Howes & Payne, 1990), and PUPS (Anderson, 1987). EXPL employs a set of domain-independent causal heuristics to explain instructions given in the form of a demonstration of a whole task-action mapping. Each action of the demonstration must be assigned a role in causing the final state change in the device; that is, the outcome of what the task achieves. EXPL's causal heuristics are the previous-action heuristic, the loose-ends heuristic, and the identity heuristic. The previous-action heuristic hypothesizes that if an event follows immediately after an action, then that action was one of the causes of the event. The loose-ends heuristic assumes that all user actions contribute to the task and therefore if there is an action and a feature of the task that have not been associated by another means it will associate them. The identity heuristic is characterized as, "if something appears in a user action, and in a later system response, the user action is probably a cause of the system response."

Where TAL uses lexical-match to understand instruction, EXPL uses causal heuristics. Although these analytic techniques perform the same function in their respective models, they do so very differently. Neither the loose-ends nor the previous-action heuristic are used by TAL. The greatest similarity between EXPL's and TAL's analysis mechanisms is between the identity heuristic and the lexical-match function. Where the causal identity heuristic finds matches between actions and task features that have surface similarities in their visual representation, the lexical-match function finds matches between items according to their semantic similarity.

In describing EXPL, Lewis (1988) raised the possibility that users can construct either rationalistic or superstitious accounts of task-action mappings. A rationalistic account is one in which the role of every action in achieving the task is determined. A superstitious account is one in which there are actions that it is known must be carried out, but it is not known what their individual roles are in achieving the overall task. This distinction is similar to the figurative–operational distinction made by Payne (1989). A figurative account of a task-action mapping is one in which all of the roles of the individual actions are elaborated. An operational account is one in which the constituents of a sequence of actions, learned by rote, have no individual meaning, instead only the sequence as a whole has a meaning. Payne observed that users construct both figurative and operational accounts and that which kind was constructed could be affected by the lexical semantics used in the action sequence (Payne et al., 1990). TAL constructs rationalistic, or figurative accounts, of task-action mappings.

Chunking is itself a learning mechanism for implementing ABL. It constructs a chunk that encodes the problem solver's analysis of a particular situation. The fact that the use of external sources of action (external generators), such as instruction, the device display, and the physical input devices, results in rules that are dependent on the presence of that external source, is an emergent property of chunking and of ABL.

One historical criticism of chunking and of explanation-based learning (Mitchell, 1986), in particular, was that it could not be used to learn new knowledge, only operationalize knowledge already in the system. However, Lewis (1988) observed that the domain theory may itself be speculative and not guaranteed to produce the correct analysis. For example, the causal heuristics of EXPL are only a domain theory of how the world might be, not how it necessarily is. Indeed for a badly designed device, the identity, previous-action, and loose-ends heuristic might well not apply. In this respect the analysis of an example is only correct if the domain theory happens to be correct for that example. In agreement with Lewis (1988), Rosenbloom and Aasman (1992) described how chunking can be described as acquiring new knowledge if there is only initially a "low belief" in the domain theory. The existence of the example or instruction is used to confirm, or make "high belief," the problem solver's confidence in that aspect of the domain theory. This is exactly what happens when TAL generates candidate actions in the hope of cueing an instruction-based chunk. Each candidate action is a "low belief," possible action. the firing of the instruction- based chunk serves to confirm that this action is in fact correct. When used in this way, the chunking is acquiring new knowledge: The new knowledge is inductively acquired by assuming that because an action is recognized (e.g., with instuction-based chunks), it must be correct.

A particularly interesting Soar model to contrast to TAL is Instructo-Soar (Huffman, 1994). Instructo-Soar operates in a simulated blocks-world domain and is given goals such as picking up blocks and building towers. It learns by taking a sequence of instructions given in a flexible natural-language-like form. In fact, instructions are interpreted by a model of human natural language processing (Lewis, 1993). The language processing component outputs recognition chunks similar to TAL's instruction-based chunks and later uses them to guide problem solving.

Despite the similarity in the acquisition of recognition chunks from instruction, the way in which Instructo-Soar problem solves is very different to TAL. Where TAL learns by constructing an analysis of instruction using lexical-match to relate actions to the task, Instructo-Soar constructs an analysis of instruction using a model of how the external world works. Instructo-Soar simulates the operation of the external world in order to determine the consequences of actions without actually performing the actions in the real world. It can thereby predict what the consequences of

following an instruction will be. Instructo-Soar thus employs a powerful problem-solving mechanism for determining exactly the right conditions in which a particular instruction should be employed. In this respect, Instructo-Soar captures an important aspect of human problem solving that is ignored by TAL. People do learn the effects of their actions, and they do use this knowledge to determine the actions required to achieve a task.

However, we believe that such a mechanism is not appropriate for modeling the learner of a new device. Novice users of computer applications do not have the knowledge to be able to predict exactly how the device will respond to their actions. If they did, and if they employed the knowledge as it is employed in Instructo-Soar, then learning would be trivial and no errors would occur in performance; after all, the user would know all there is to know about the device.

To summarize, TAL is a model that uses ABL to construct interpretations of instructions it is given. The interpretation serves as an understanding of the instruction in terms of the task. In this respect TAL is similar to a number of other models of human cognition in task-action mapping. Each model varies in the knowledge that is brought to bear in order to construct analysis.

Searching for Task-Action Mappings

In the absence of instruction, TAL's behavior can be described as a search of the space of candidate task-action mappings. The version of TAL described in this article searched six different spaces in order to find a mapping:

1. Similar tasks. By default, the first source of candidate actions is the task-action mapping of a similar task. Similar tasks share a semantic feature. Task-action mappings for individual task features transfer freely from one task context to another.
2. The device display. If there is no similar task, then TAL examines the device display in order to generate candidate actions. Either all of the actions generated from the display can be considered as candidates, or the set can be constrained by the application of lexical-match to determine which actions match the task description.
3. The physical affordances of the input devices. Once a candidate display item has been identified, the physical affordances of the input devices can be examined in order to determine an action for selecting the display item.
4. Task features. If the device display does not provide interactive support for determining the next action, then an action must be derived from the task. The set of task features must be searched in order to determine which is to be communicated next and thereby establish a syntax for the mapping.

5. Lexical knowledge. Once a task feature has been identified as "the next to be communicated," then TAL can generate words, using the lexical-recall function, to determine a word that can be used to communicate the feature.
6. Abbreviation functions. Once a word has been identified as suitable for communicating one of the task features, the word may need abbreviating. TAL can generate, from long-term memory, a number of abbreviation functions, for example, take-first-two-consonants and take-first-two-letters.

Both display-based and keyboard-based devices support the search of similar tasks, generator (1). In addition, display-based devices tend to encourage the search of the display (2) and the affordances (3), and keyboard-based devices tend to encourage the search of task features (4), lexical knowledge (5), and abbreviation knowledge (6). However, in principle, any type of device can require any type of generator to be searched: What might typically be called a display-based device (e.g., MS Word) may in some circumstances rely on a user's ability to generate semantically appropriate words (e.g., to specify the units of a margin width) and/or abbreviations (e.g., in order to use keyboard "accelerators": control sequences that obviate the need to use the menus). Similarly, the display of a keyboard-based device can be used as a generator, albeit a rather impoverished one. For example, in Unix, the files of the current directory may be listed on the display by typing the string "ls." This list can be used as an external generator of file names.

These six sources of candidate actions can be used flexibly by TAL to construct a range of task-action mappings, and not just those mappings that conform to the structures that we have examined for the MS Word and Unix file manipulation tasks. The generators can be seen as a set of resources TAL can search and utilize with its underlying learning mechanism: If available, TAL uses knowledge of similar tasks; failing this it uses the display and affordances, and if there is no display then it uses lexical-recall and abbreviation knowledge.

Further, as with most searches, the problem solver may choose to examine the candidates that are most likely to succeed first. The ordering of search spaces from (1) to (6) is not accidental. As we described earlier, some generators are not as efficient as others. Most significantly, in a population of users, the generation of lexical knowledge is known to be far less reliable than generation from a set of externally presented items (e.g., Furnas et al, 1987). In general, the reliability of a generator is determined by the medium on which it is based, small external generators being more reliable than internal generators (see also Jordan, Draper, MacFarlane, & McNulty, 1991).

With the range of generators that TAL can bring to bear to construct task-action mappings, the model is effective for both display-based and key-

board-based tasks. If the external device display is a rich source of candidate actions, then it will be used by TAL to constrain task-action mapping; if not then TAL will fall back on internal generators for hierarchically constructing task-action mappings. For this reason we observe that TAL's behavior on display-based tasks can be characterized as situated (Agre & Chapman, 1990): The content of the display drives behavior. In contrast, TAL's behavior on keyboard-based tasks can be characterized as plan-based (Young & Simon, 1987): It decomposes task-action mappings hierarchically and top-down from the task description. Further, whether TAL's behavior is situated or plan-based is not preprogrammed into the model; rather it emerges with the demands of the task and the state, or more specifically from the generators that need to be searched in order to construct task-action mappings.

Comparing TAL to Models of Instructionless Learning

Taking instruction is just one of a number of modes of learning that people engage in when learning to use complex devices. Others include learning from manuals and learning by exploration (Carroll, 1982; Franzke, 1994, 1995; Polson & Lewis, 1990; Rieman, 1994; Rieman, Young, & Howes, in press; Shrager & Klahr, 1986). Consider users who engage in exploration when they have a well-specified task but do not know which actions are required to achieve this task (what Rieman [1994] called "problem-oriented exploration"). With luck, these instructionless learners will learn the required action sequence without instruction. The appropriateness of actions is determined by trying them out to see what their effects are. If an action leads the problem solver closer to the task, then it will be deemed successful, otherwise a failure.

During instructionless learning, an action must be generated before it is tried. The user must know what the space of available actions is, and must search this space. We believe that the exploratory learner uses the same sources to generate actions as the instruction-based learner modeled by TAL. Indeed, Franzke (1994) described exploratory learning of display-based devices in similar terms to our earlier description, as a "search through a problem space of legal interactions," and described the use of the display as a generator of such actions during exploration. Other authors have observed that constraining the size of the search space improves the success of student exploration (Carroll & Carrithers, 1984). This is consistent with the observation that display-based devices provide physical constraints on the expansion of task-action mappings (Duff, 1989). Further, it is not surprising, given the analysis here, that keyboard-based devices are rarely learned by exploration. The search space of a keyboard-based device is simply not constructable by most users. The generators required by keyboard-based devices, based as they are on long-term memory, fail to generate all of the required

items (Furnas et al., 1987). Display-based devices, in contrast, provide reliable, if large (Carroll & Carrithers, 1984) search spaces for exploration.

During instructionless learning, as well as being generated, actions must be interpreted and analyzed. Polson and Lewis (1990), in particular, discussed how an exploratory learner must analyze interactions with the device in order to determine exactly which actions cause which events. Also, Shrager and Klahr (1986) described how different learners of the BigTrak toy make different assumptions about the meanings of particular buttons. To this degree, TAL's process of interpreting instructions is similar to the process of analyzing exploration. However, much of the analysis that exploratory learners do is to determine why particular actions had particular effects on the device: why they succeed and why they fail. Where TAL constructs simple analyses in terms of the task features, the exploratory learner must then revise these initial analyses depending on how the device actually responds to the actions that are performed. For these reasons, we believe that the predictions of device learnability made from TAL's instruction-taking mechanism should generalize to instructionless learning situations. This claim is made on the basis of TAL's mechanism for analyzing the task-action mappings to determine the roles of individual actions and the mechanism for searching the space of candidate task-action mappings.

DISCUSSION: AN INTEGRATED COGNITIVE MODEL

In the introduction to this article we stated the aim of describing an integrated model of the psychological phenomena of task-action mapping in human–computer interaction. TAL is integrated in two respects. First, it covers a range of previously unrelated empirical phenomena indicating that consistent, interactive, and meaningful devices are easier to learn and use. This is in contrast to previous models that address just consistency (e.g., TAG) or just interactivity (e.g., Dibs; Larkin, 1989). Second, TAL is both a learning and a performance model. It does not just learn task-action mappings as is the case in Howes and Payne (1990), neither does it just perform task-action mappings as is the case in Kitajima and Polson (1992, 1995). Instead it learns knowledge encoded in rules that are in the form required to perform the task-action mapping. The same architectural mechanisms, derived from Soar, that are used during learning are also used during performance.

In accordance with Newell (1972, 1990) we believe that the time is right for psychologists to pursue more integrated, or unified, accounts of human behavior. TAL is an example of such an account. We have shown how the use of the meaning, consistency, and interactivity of task-action mappings are mutually supporting. Along these dimensions there is not a part of the model that could be pulled out to "stand alone." For example a number of the predictions of consistency are based on the meaningfulness of the words

used in the task-action mapping, and the model could not make predictions about the advantage of interactivity if it were not dependent on lexical-recall and lexical-match. Similarly, some aspects of consistency are relevant to display-based devices (e.g., the consistency of the mouse actions used to select display items) and some are relevant to keyboard-based devices (e.g., the consistency of abbreviations). Further, the model is computationally implemented and based on a few simple assumptions about the nature of the underlying cognitive architecture.

In human–computer interaction, there have been other attempts to construct integrated models that both learn and perform task-action mappings. They include two earlier Soar models of computer users: a model of the use of MS Word (Howes & Young, 1991), and a model of the use of an ATM machine (Vera, Lewis, & Lerch, 1993), and in addition a model of the exploratory acquisition of task-action mappings, CE+ (Polson & Lewis, 1990).

Howes and Young's model of MS Word use was designed to predict a number of the consistency and interactivity effects described in the introduction. In particular it generated candidate actions from the display of the device (affordances) and it learned generalized mapping rules for predicting consistency effects. It used the device display as a source of candidate actions and formed chunks that remained dependent on the presence of that display. However, although Howes and Young's model generated candidate actions from the device display and learned display-based chunks, it did not first learn instruction-based chunks. It was not therefore aligned with Soar's chunking hypothesis. In contrast, Vera et al.'s model of the use of an ATM machine used display-based generation of candidate actions (as used by Howes and Young) but chunked instruction in a way that was aligned with the chunking hypothesis. The model reported in this article combines some of the HCI phenomena captured by Howes and Young's MS Word model with an instruction-taking mechanism similar to that used in Vera et al.'s ATM model.

In contrast to the instruction-taking models, CE+ (Polson & Lewis, 1990) is a proposed model of learning by exploration. It incorporates assumptions from a production system model of the encoding of procedural knowledge called CCT (Kieras & Polson, 1985), from EXPL (Lewis, 1988), and from research on problem solving. Like the present work, CE+ is motivated by the desire to understand the cognition involved when people interact with walk-up and-use interfaces. In outline, CE+ consists of (a) a problem-solving component for choosing actions; (b) an EXPL-like learning component, which analyzes the effects of choices and caches the results as CCT rules; and (c) a component capable of coordinating execution of the rules with the problem-solving component.[4]

[4] Lewis and Polson reported that although the components of CE+ exist they have not been integrated.

If CE+ has the knowledge required to achieve its goal, then it uses it; otherwise it problem solves. The problem-solving component of CE+ is responsible for generating exploratory behavior. It uses a label-following heuristic (Engelbeck, 1986) to select between candidate options. This heuristic says pick the label that shares the most words with the task description. CE+ uses its EXPL component to learn from what it has done during problem solving and construct new CCT-like productions.

CE+ has similarities to TAL. For example, both CE+ and TAL find it difficult to learn keyboard-based interfaces. Neither CE+ nor TAL can learn in situations where there are insufficient external cues to the available actions. Unfortunately a detailed comparison of CE+ to TAL is difficult because CE+ is not computationally implemented. This is reflected in the lack of detail about CE+'s behavior in Polson and Lewis (1990). However it is possible to make some substantive comparisons. For example, CE+ is committed to a simple mechanism for matching task descriptions to displayed labels, that is, the label-following heuristic. This heuristic is empirically supported (Polson & Lewis, 1990). In contrast, TAL uses a semantic match model that is implemented using a database of semantic links.

A relative weakness of CE+ is that it does not assume that the selection of display objects will be treated consistently (e.g., menus) and that methods for using them will generalize across the task semantics. Also, it does not create a hierarchy of mapping knowledge, the components of which can be reused for different tasks. This is because the problem solving occurs at only one level rather than at the multiple levels to be found in TAL (e.g., the hierarchical decomposition used to learn keyboard-based devices).

The major difference between TAL and CE+ is that CE+ uses EXPL to analyze which actions of the method contributed to the outcome. In contrast, TAL is oblivious to the outcome of the method. All of the causal associations that it learns are between the task description and the actions of the method. This limitation in TAL is deliberate and reflects the limited complexity of the devices it is intended to learn. Clearly, it could, in principle, gain much leverage by reflecting on what its behavior actually achieves, rather than assuming that the outcome is correct.

CONCLUSION

To summarize, TAL is a model of the human learning of computer interfaces that are relatively easy to learn. TAL does not model human learning on more difficult interfaces, nor does it model how people recover from errors. We have shown how the model is consonant with the fact that people learn consistent interfaces more rapidly and with fewer errors than inconsistent interfaces, that they learn display-based or interactive interfaces more rapidly than keyboard-based interfaces, and that they learn devices using meaningful words more efficiently than those using meaningless words.

Further, we have claimed that these phenomena derive from regularities in human cognition that are independent of the mode of learning. Thus, although TAL models a process of learning by taking instruction, we have discussed how the constraints on which it is based apply to other modes of learning such as learning by exploration. In essence, the idea is that the need to search for candidate actions and relate them to the task description is common to all modes of learning. The search space is constrained by the interactivity, consistency, and meaningfulness of the task-action mappings.

TAL has many limitations that restrict its utilization of the constraints in a device. As regards interactivity, TAL assumes much about the way in which the world is perceived, including how displays are parsed and how action affordances are detected. In addition, it assumes a mechanism for packaging the continuous events that happen in the world into discrete episodes. That is, TAL iterates through steps of performing an action and then perceiving the world. It does not consider that the world may change continuously after an action has been done, nor that an action itself may take time to do. Further, TAL does not capture the advantages of a display-based device in error recovery and it does not learn where items are on the display; that is, locational knowledge. Nygren, Lind, Johnson, and Sandblad (1992) and Lansdale (1991) showed that the use of locational knowledge is an essential part of expert skill for task-action mapping.

There are many other aspects of consistency that affect human behavior that are not captured by TAL. For example, the congruence of name sets (Carroll, 1982). A set of commands are congruent if they normally occur in opposition to each other. For example, "on" is congruent to "off" but not to "stop." In addition, TAL fails to capture the consistency of location: Obviously, items that are not always in the same location on the display reduce the effectiveness of locational knowledge (Teitelbaum & Granda, 1983).

Lastly, there are examples of interfaces that require users to learn a large number of arbitrary mappings between tasks and actions. For example, learning the Emacs word processor requires learning a number of arbitrary alphanumeric key sequences such as "control-x" for a file manipulation task. People find such devices very difficult to learn and indeed evidence from cognitive psychology suggests that people find such mappings highly confusing. Although TAL captures the fact that nonmeaningful task-action mappings are difficult to transfer and to recall, it does not capture the way that meanings are acquired through learning. The Unix command *grep* may be meaningless to the novice user, but it soon becomes meaningful. For example, more experienced users use phrases such as, "you need to grep for the word *apple*."

Our current work is aimed at addressing these weaknesses and also at reducing TAL's dependence on external instruction. To this end, we are building a model that explores without the help of an instructor. This will

involve the model recovering from explorations that lead to failure, a process that demands a more flexible control structure than is used in TAL. Howes (1993, 1994) described a technique called Recognition-based Problem Solving (RPS), which provides such a control structure. Our current work incorporates the constraints investigated in TAL with the RPS mechanism.

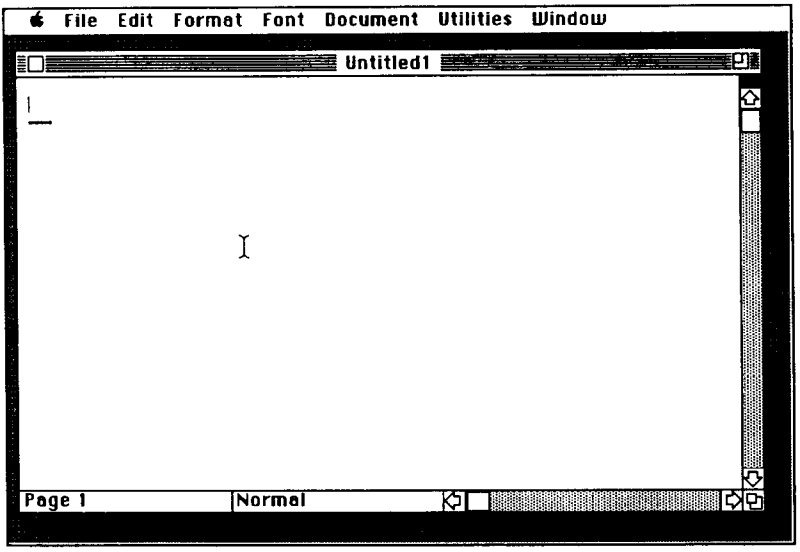
Lastly, we return to the central implication of the TAL model for cognitive science. Where researchers have previously reported limited-scope models that address just consistency (TAG; Payne & Green, 1986), or just interactivity (Dibs; Larkin, 1989), TAL is a single model that produces behaviors consonant with human performance on a wide range of disparate phenomena. Moreover, a number of the phenomena emerge in TAL's behavior from a few simple assumptions about the underlying cognitive architecture.

Further, where previous work on situated cognition has focused on the situated aspects of cognition at the expense of the more plan-based aspects (Agre & Chapman, 1990; Howes & Payne, 1990), we have shown that within the field of HCI the chunking mechanism on which TAL is based is effective for both situated and plan-based tasks. If the external device display is a rich source of candidate actions, then it will be used by TAL to constrain task-action mapping; if not then TAL will fall back on other internal generators for constructing task-action mappings.

APPENDIX A: MS WORD DISPLAY STATES

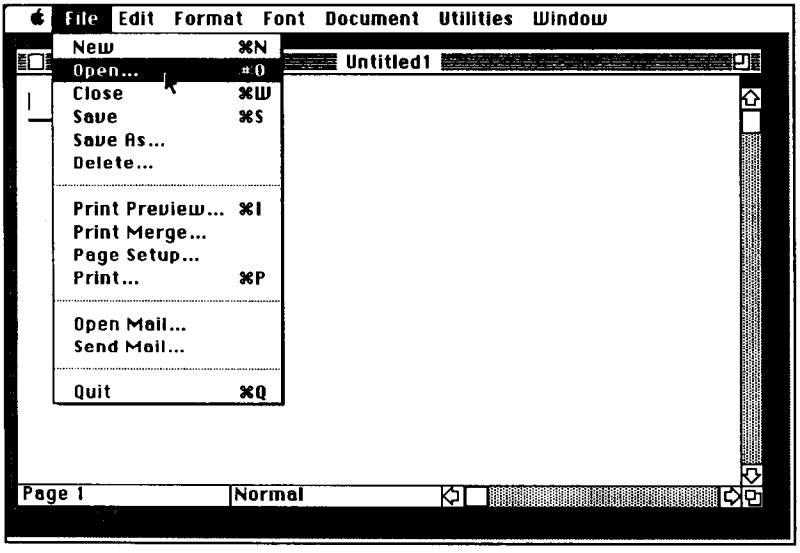
This scenario is taken from version 4.00B of Microsoft Word (MS Word), a word processor application for the Apple Macintosh. The task is to open a file called "invitation." It is representative of a set of tasks for dealing with file manipulation. These tasks include creating new files, opening files, closing files, and saving files.

The Intitial Display



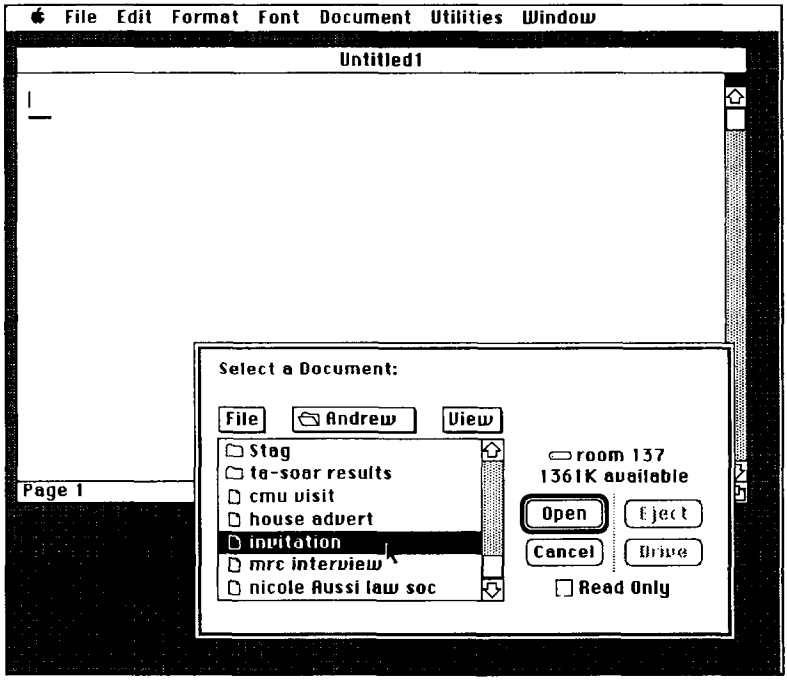
The File Menu

The file menu has been selected by moving the mouse pointer over the File option on the menu bar and pressing down the mouse button. The user has then moved the mouse over the Open option on the pull-down.



The Document Box: File Selection

The user has released the mouse button on the Open option and the device has displayed a document box. The user has then moved the mouse over the "invitation" document and clicked the mouse button.



APPENDIX B: EXAMPLE CHUNKS LEARNED BY TAL ON THE MS WORD TASKS

This appendix contains example chunks learned by TAL when given four Apple Macintosh MS Word tasks: open file, change font, new file, and close file.

- Each chunk consists of a left-hand side of conditions, followed by an arrow "→" and then a right-hand side. When reading a chunk, the reader should start by examining the right-hand side to determine the effect of the chunk.
- The chunks are listed in reverse order. Chunk-1 was learned first and chunk-44 last.

- Each effect has both a recall chunk and a recognition chunk, for example, for the effect "move-to close" chunk-44 is the recall form and chunk-42 the recognition form. The recognition form refers to the instruction and is therefore instruction based.
- Chunk-44 is a recall chunk for the action to move-to the close option. It reads:

```
IF there is a goal <g1> and a state <s1>
and state <s1> has a task <n1> and a display-item <d1>)
and task <n1> has the name select and the previous-task <p1> and a
feature <f2>
and previous-task <p1> has the name init and the feature <f1>
and display-item <d1> has label close
and feature <f2> has name effect and value make-unseen
and feature <f1> has name init
THEN propose an operator with name move-to value close and feature
<f2>
```

```
(sp chunk-44
  :chunk
  (goal <g1> ^object nil ^state <s1>)
  (<s1> ^task <n1> ^display-item <d1>)
  (<n1> ^name select ^previous <p1> ^feature <f2>)
  (<p1> ^name init ^feature <f1>)
  (<d1> ^label close)
  (<f2> ^name effect ^value make-unseen)
  (<f1> ^name init)
-->
  (<g1> ^operator <n2> + ^operator <n2> =)
  (<n2> ^feature <f2> + ^value close + ^name move-to +))
```

- Chunk-42 is an instruction-based recognition chunk for the action to move-to the close option. The instruction is Soar object <i1>.

```
(sp chunk-42
  :chunk
  (goal <g1> ^state <s1>)
  (<s1> ^task <n1> ^instruction <i1>)
  (<n1> ^name select ^previous <p1> ^feature <f2>)
  (<i1> ^value close ^name move-to ^feature <f2>)
  (<p1> ^name init ^feature <f1>)
  (<f2> ^name effect ^value make-unseen)
  (<f1> ^name init)
-->
  (<g1> ^operator <n2> + ^operator <n2> =)
  (<n2> ^name move-to + ^value close + ^feature <f2> +))
```

- Chunk-16 is a recall chunk for the action to press the mouse button. It is generalized for all selection tasks involving the menu bar.

```
(sp chunk-16
   :chunk
   (goal <g1> ^object nil ^state <s1>)
   (<s1> ^display-item <n3> ^task <n2> ^display-item <d2> ^display-
item <d1>)
   (<n3> ^name pointer ^location <d2>)
   (<n2> ^name select ^previous <n1>)
   (<n1> ^name move-to ^feature <f1>)
   (<d2> ^name menu-bar)
   (<d1> ^name mouse)
   (<f1> ^name class)
-->
   (<g1> ^operator <n4> + ^operator <n4> =)
   (<n4> ^feature <d2> + ^value mouse + ^name press +))
```

- Chunk-3 is an instruction-based recognition chunk for the action to press the mouse button. It is generalized for all selection tasks involving the menu bar. The instruction is <i1>.

```
(sp chunk-3
   :chunk
   (goal <g1> ^state <s1>)
   (<s1> ^task <n3> ^display-item <n2> ^instruction <i1> ^display-item
<d1>)
   (<n3> ^name select ^previous <n1>)
   (<n2> ^name pointer ^location <d1>)
   (<n1> ^name move-to ^feature <f1>)
   (<i1> ^name press ^value mouse ^feature <d1>)
   (<d1> ^name menu-bar)
   (<f1> ^name class)
-->
   (<g1> ^operator <n4> + ^operator <n4> =)
   (<n4> ^name press + ^value mouse + ^feature <d1> +))
```

REFERENCES

Agre, P.E., & Chapman, D. (1990). What are plans for? In P. Maes (Ed.), *New architectures for autonomous agents: Task-level decomposition and emergent functionality*. Cambridge, MA: MIT Press.

Anderson, J.R. (1993). *Rules of mind*. Hillsdale, NJ: Erlbaum.

Barnard, P.J., Hammond, N.V., Morton, J., Long, J., & Clark, I.A. (1981). Consistency and compatibility in human-computer dialogue. *International Journal of Man-Machine Studies, 15,* 87-134.

Briggs, P. (1990). Do they know what they're doing? An evaluation of word-processor users' implicit and explicit task-relevant knowledge, and its role in self-directed learning. *International Journal of Man-Machine Studies 32,* 385-398.

Card, S.K., Moran, T.P., & Newell, A. (1983). *The psychology of human-computer interaction*. Hillsdale, NJ: Erlbaum.

Carroll, J.M. (1982). Learning, using and designing command paradigms. *Human Learning, 1,* 31-62.

Carroll, J.M., & Carrithers, C. (1984). Training wheels in a user interface. *Communications of the ACM, 27,* 800–806.

Chapman, D. (1990). *Vision, instruction and action*. Unpublished doctoral dissertation, MIT, Cambridge, MA.

Chi, M.T.H., Bassok, M., Lewis, M.W., Reimann, P., & Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science, 13,* 145–182.

Collins, A., Brown, J.S., & Newman, S.E. (1986). Cognitive apprenticeship: Teaching the craft of reading, writing, and mathematics. In L.B. Resnick (Ed.), *Cognition and instruction: Issues and agendas.* Hillsdale, NJ: Erlbaum.

DeJong, G., & Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning, 1,* 145–176.

Dietterich, T.G., & Michalski, R.S. (1983). A comparative review of selected methods for learning from examples. In *Machine learning: An artificial intelligence approach.* Palo Alto, CA: Toga Press.

Draper, S.W. (1986). Display managers as the basis for user machine communication. In D.A. Norman & S.W. Draper (Eds.), *User centred system design.* Hillsdale, NJ: Erlbaum.

Duff, S.C. (1989). Reduction of action uncertainty in process control systems: The role of device knowledge. In E. Megaw (Ed.), *Contemporary ergonomics.* London: Taylor & Francis.

Engelbeck, G.E. (1986). *Exceptions to generalisations: Implications for formal models of human–computer interaction*. Unpublished master's thesis, University of Colorado, Department of Psychology, Boulder, CO.

Franzke, M. (1994). Exploration and experienced performance with display-based systems (Tech. Rep. No. 94-05). University of Colorado at Boulder, Institute of Cognitive Science.

Franzke, M. (1995). Turning research into practice: Characteristics of display-based interaction. In I.R. Katz, R. Mack, & L. Marks (Eds.), *Human factors in computing systems: CHI'95 Conference Proceedings.* Reading, MA: Addison-Wesley.

Furnas, G.W., Landauer, T.K., Gomez, L.W., & Dumais, S.T. (1987). The vocabulary problem in human-system communication. *Communications of the ACM, 30,* 964–971.

Gaver, W. (1991). Technology affordances. In S.P. Robertson, G.M. Olsonn, & J.S. Olson (Eds.), *Reaching through technology: CHI'91 Conference Proceedings.* Reading, MA: Addison-Wesley.

Gentner, D.R., & Grudin, J. (1990). Why good engineers (sometimes) create bad interfaces. In J.C. Chew & J. Whiteside (Eds.), *Empowering People: CHI'90 Conference Proceedings.* Reading, MA: Addison-Wesley.

Gibson, J.J. (1979). *The ecological approach to visual perception.* Boston: Houghton Mifflin.

Green, T.R.G., & Payne, S.J. (1984). Organisation and learnability in computer languages. *International Journal of Man–Machine Studies, 21,* 7–18.

Green, T.R.G., Schiele F.G., & Payne, S.J. (1988). Formalisable models of user knowledge in human–computer interaction. In G.C. van der Veer, T.R.G. Green, J-M. Hoc, & D.M. Murray (Eds.), *Working with computers: Theory versus outcome.* London: Academic.

Grudin, J. (1989). The case against user interface consistency. *Communications of the ACM, 32,* 1164–1173.

Howes, A. (1992). *Learning task-action mappings by exploration.* Unpublished doctoral thesis, University of Lancaster, Department of Computer Science.

Howes, A. (1993). Recognition-based problem solving. In J.W. Kintsch (Ed.), *Proceedings of the 15th Annual Meeting of the Cognitive Science Society.* Hillsdale, NJ: Erlbaum.

Howes, A. (1994). A model of the acquisition of menu knowledge by exploration. In B. Adeslon, S. Dumais, J. Olson (Eds.), *Celebrating interdependence: Proceedings of Human Factors in Computing Systems CHI'94.* Reading, MA: Addison-Wesley.

Howes, A., & Payne, S.J. (1990). Display-based competence: Towards user models for menu-driveb interfaces. *International Journal of Man-Machine Studies, 33,* 637-655.

Howes, A., & Young, R.M. (1991). Predicting the learnability of task-action mappings. In S.P. Robertson, G.M. Olson, & J.S. Olson (Eds.), *Reaching through technology—CHI'91 Conference Proceedings.* Reading: MA: Addison-Wesley.

Huffman, S.B. (1994). *Instructable autonomous agents.* Doctoral thesis, University of Michigan, Department of Electrical Engineering and Computer Science, Ann Arbor.

John, B.E., & Vera, A.H. (1992). A GOMS analysis of a graphic-display-based, machine-paced, highly interactive task. In P. Bauersfeld, J. Bennet, & G. Lynch (Eds.), *Proceedings of Human Factors in Computing Systems, CHI'92.* Reading, MA: Addison-Wesley.

Jordan, P.W., Draper, S.W., MacFarlane, K.K., & McNulty, S.A. (1991). Guessability, learnability, and experienced user performance. In D. Diaper & N. Hammond (Eds.), *People and computers VI.* New York: Cambridge University Press.

Kellogg, W.A. (1987). Conceptual consistency in the user interface: Effects on user performance. In H.J. Bullinger & B. Shackel (Eds.), *Proceedings of Human-Computer Interaction—INTERACT'87.* Amsterdam: Elsevier.

Kieras, D.E., & Polson, P.G. (1985). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies, 22,* 365-394.

Kitajima, M. (1989). A formal representation system for the human-computer interaction process. *International Journal of Man-Machine Studies, 30,* 669-696.

Kitajima, M., & Polson, P.G. (1992). A computational model of skilled use of a graphical user interface. In P. Bauersfeld, J. Bennet, & G. Lynch (Eds.), *Proceedings of Human Factors in Computing Systems, CHI'92.* Reading, MA: Addison-Wesley.

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

Laird, J.E., Newell, A., & Rosenbloom, P.S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence, 33,* 1-64.

Lansdale, M.W. (199). Remembering about documents: Memory for appearance, format, and location. *Ergonomics, 34,* 1161-1178.

Larkin, J.H. (1989). Display-based problem solving. In D. Klahr, & K. Kotovsky (Eds.), *Complex information processing: The impact of Herbert A. Simon.* Hillsdale, NJ: Erlbaum.

Lave, J., Murtaugh, M., & de la Rocha, O. (1988). The dialectic of arithmetic in grocery shopping. In J. Lave (Ed.), *Cognition in practice.* New York: Cambridge University Press.

Lee, A.Y., Foltz, P.W., & Polson, P.G. (1994). Memory for task-action mappings: Mnemonics, regularity and consistency. *International Journal of Human-Computer Studies, 40,* 771-794.

Lewis, C.H. (1988). Why and how to learn why: Analysis-based generalization of procedures. *Cognitive Science, 12,* 211-256.

Lewis, R.L. (1993). *An architecturally-based theory of human sentence comprehension.* (Tech. Rep. No. CMU-CS-93-226). Department of Computer Science. Pittsburgh, PA: Carnegie-Mellon University.

Lewis, R.L., Newell, A., & Polk, T.A. (1989). Toward a Soar theory of taking instructions for immediate reasoning tasks. *Proceedings of the Cognitive Science Annual Conference* Hillsdale, NJ: Erlbaum.

Marr, D. (1982). *Vision.* San Francisco.

Mayes, J.T., Draper, S.W., McGregor, M.A., & Oatley, K. (1988). Information flow in a user interface: The effect of experience and context on the recall of MacWrite screens. In D.M. Jones & R. Winder (Eds.), *People and computers IV.* New York: Cambridge University Press.

Miller, G.A., Galanter, E., & Pribram, K. (1960). *Plans and the structure of behaviour.* New York: Holt, Rinehart & Winston.

Miller, G.A., & Gildea, P.M. (1987). How children learn words. *Scientific American, 257,* 86–91.

Mitchell, T.M. (1986). Explanation-based generalisation: A unifying view. In R.S. Michalski, J.G. Carbonell, & T.M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Los Altos, CA: Morgan Kaufmann.

Newell, A. (1972). You can't play 20 questions with nature and win. In W.C. Chase (Ed.), *Visual information processing.* London: Academic.

Newell, A. (1982). The knowledge level. *Artificial Intelligence, 18,* 87–127.

Newell, A. (1990). *Unified theories of cognition.* Cambridge, MA: Harvard University Press.

Newell, A., & Simon, H.A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall.

Norman, D.A. (1988). *The psychology of everyday things.* New York: Basic Books.

Nygren, E., Lind, M., Johnson, M., & Sandblad, B. (1992). The art of the obvious. In P. Bauersfeld, J. Bennett, & G. Lynch (Eds.), *Proceedings of Human Factors in Computing Systems, CHI'92* (pp. 235–239). Reading, MA: Addison-Wesley.

Payne, S.J. (1985). *Task-action grammars: The mental representation of task languages in human-computer interaction.* Unpublished doctoral dissertation, Department of Psychology, University of Sheffield, Sheffield, England.

Payne, S.J. (1989). A notation for reasoning about learning. In J. Long & A. Whitefield (Eds.), *Cognitive ergonomics and human-computer interaction.* Cambridge: Cambridge University Press.

Payne, S.J. (1990). Looking HCI in the I: A simple notation for describing interactions at the user interface. In D. Diaper, D. Gilmore, G. Cockton, & B. Shackel (Eds.), *Human-computer interaction—INTERACT'90.* Amsterdam: Elsevier.

Payne, S.J. (1991). Display-based action at the user interface. *International Journal of Man-Machine Studies, 35,* 275–289.

Payne, S.J., & Green, T.R.G. (1986). Task-action grammars. A model of the mental representation of task languages. *Human-Computer Interaction, 2,* 93–133.

Payne, S.J., & Green, T.R.G. (1989). The structure of command languages: An experiment on task-action grammar. *International Journal of Man-Machine Studies, 30,* 213–234.

Payne, S.J., Squibb, H., & Howes, A. (1990). The nature of device models: The yoked state space and some experiments with text editors. *Human-Computer Interaction, 5,* 415–444.

Polson, P.G., & Lewis, C.H. (1990). Theory-based design for easily learned interfaces. *Human-Computer Interaction, 5,* 191–220.

Reisner, P. (1981). Formal grammar and design of an interactive system. *IEEE Transactions on Software Engineering, 5,* 229–240.

Rieman, J.F. (1994). *Learning strategies and exploratory behavior of interactive computer users.* (Tech. Rep. No. CU-CS-723-94). University of Colorado at Boulder, Department of Computer Science.

Rieman, J.F., Young, R.M., & Howes, A. (1996). A dual-space model of iteratively deepening exploratory learning. *International Journal of Human-Computer Studies.*

Rosenbloom, P., & Aasman, J. (1992). Knowledge level and inductive uses of chunking. In J.A. Michon & A. Akyurek (Eds.), *Soar: A cognitive architecture in perspective.* Dordrecht: Kluwer.

Rosenbloom, P.S., Laird, J.E., & Newell, A. (1987). Knowledge-level learning in Soar. *Proceedings of AAAI'87* (pp. 499–504). Los Altos, CA: Morgan Kaufman.

Rosenbloom, P.S., Laird, J.E., & Newell, A. (1989). The chunking of skill and knowledge. In H. Bouma & B.A.G. Elsendoorn (Eds.), *Working models of human perception.* London: Academic.

Shrager, J., & Klahr, D. (1986). Instructionless learning about a complex device: The paradigm and observations. *International Journal of Man-Machine Studies, 25,* 153–190.

Teitelbaum, R.C., & Granda, R.E. (1983). The effects of positional constancy on searching menus for information. In A. Janda (Ed.), *Proceedings of Human Factors in Computing Systems, CHI'83* (pp. 150–153). Reading, MA: Addison-Wesley.

Vera, A.H., Lewis, R.L., & Lerch, F.J. (1993). Situated decision-making and recognition-based learning: Applying symbolic theories to interactive tasks. In W. Kintsch (Ed.), *Proceedings of the 15th Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: Erlbaum.

Young, R.M., Howes, A., & Whittington, J. (1990). A knowledge analysis of interactivity. In D. Diaper, D. Gilmore, G. Cockton, & B. Shackel (Eds.), *Human-computer interaction—INTERACT'90.* Amsterdam: Elsevier.

Young, R.M., & Simon, T. (1987). Planning in the context of human-computer interaction. In D. Diaper & R. Winder (Eds.), *People and computers III.* Cambridge: Cambridge University Press.