COGNITIVE SCIENCE Vol 23 (2) 1999, pp. 117–156 ISSN 0364-0213

Episodic Indexing: A Model of Memory for Attention Events

ERIK M. ALTMANN
George Mason University
BONNIE E. JOHN
Carnegie Mellon University

People often remember what they attend to in the world. Such memory can be cast as a kind of mental catalog or index of attended objects. To investigate how such an index is acquired and used, protocol data were collected from a programmer who scrolled to off-screen objects from time to time as she worked. These protocol data were modeled using Soar, which constrains how the index is constructed. In the model, an index entry is an episodic trace encoded during attention. The trace associates the attention event with a time symbol denoting the event's occurrence. Later, the model can ask itself whether it saw that object by calling to mind an image of the attention event. If this image retrieves a time symbol, then the model infers that the object exists and can reason about bringing the object back into view. Episodic indexing is a theory of these encoding and retrieval processes. It posits that information about attention events is encoded automatically, but that retrieval requires effort and knowledge. Episodic indexing is congruent with a range of results on episodic and temporal codes and recognition and recall processes. It incorporates source monitoring (Johnson, Hashtroudi, & Lindsay, 1993) and is a simple and pervasive form of long-term working memory (Ericsson & Kintsch, 1995).

I. INTRODUCTION

Our surroundings are filled with information. Most of this is hidden to us at any given time, being out of our field of view, yet we manage to gain access to it when we need to. For example, we might recall seeing a figure in a book, or a key phrase, and then return to that area in the book to refresh our memory, or to examine the context more carefully. This article investigates how and why people remember the existence of hidden information. To obtain data on this kind of memory phenomenon, we observed an experienced programmer doing her own work at her own computer. The programmer's interaction with the computer generates much more information than fits on the screen at once. Most of this information is hidden, scrolled out of the way by the programming environment to make

Direct all correspondence to: Erik M. Altmann, George Mason University, Human Factors & Applied Cognition, Mailstop 2E5, Fairfax, VA 22030 USA; E-Mail: altmann@gmu.edu.

room for new information. However, old information remains accessible, and the programmer occasionally scrolls some hidden information back into view. She appears to have some internal representation that allows her to index hidden objects in the environment.

We set out to answer two specific questions about this internal representation, in terms precise enough to implement as a computational cognitive model. First, what does the programmer learn about displayed information when it first appears that lets her remember later that this information exists? We would like to know what she encodes in memory, and under what circumstances she encodes it. Second, what causes the retrieval of these memories? She scrolls not randomly but purposefully, when the target information is somehow relevant to the task at hand. Her recollection for the target appears to be prompted by task-related activity. We would like to understand the role of knowledge in making the connection between task-related activity and a person's index of the environment.

Previous computational models of knowledge about the environment have focused on higher-level spatial knowledge (Kuipers, 1978) or the complexities of mental imagery (e.g., Glasgow & Papadias, 1992; Kosslyn, 1981) or visual attention (e.g., Logan, 1996; Pylyshyn, 1989; Wolfe, 1994). Models of interactive and diagrammatic problem solving (Green, Bellamy, & Parker, 1987; Kitajima & Polson, 1995; Larkin, 1989; Lohse, 1993; Vera, Lewis, & Lerch, 1993) either do not address learning, or are concerned with the acquisition or transformation of skills (Bauer & John, 1995; Howes, 1994; Howes & Young, 1996; Mannes & Kintsch, 1991; Pearson, 1996; Polson & Lewis, 1990; Rieman, Young, & Howes, 1996; Ritter & Bibby, 1997; Vera, Lewis, & Lerch, 1993). The work discussed here is an attempt to understand one kind of simple learning across the boundary between environment and problem solver, in which new objects in the environment are mapped to a new internal representation that is stored permanently and interpreted as a basis for action. Such learning appears to be a key part of our ability to navigate the environment in a purposeful manner. People need to be able to encode and retrieve knowledge describing what objects exist in the environment—a form of basic landmark knowledge (Golledge, 1991; Thorndyke, 1981) that seems an essential part of the evolving human environmental response function (Neisser, 1976; Newell, 1990).

Our approach to understanding this learning uses rich data consisting of keystroke and verbal protocols (Ericsson & Simon, 1992; Ritter & Larkin, 1994; Newell & Simon, 1972; Sanderson et al., 1994; van Someren, Barnard, & Sandberg, 1994), which together trace the programmer's problem-solving steps and her actions that change the display. We then use a cognitive architecture, Soar (Lehman, Laird, & Rosenbloom, 1998; Newell, 1990; Rosenbloom, Laird, & Newell, 1992), to implement a computational model of the observed behavior. The Soar architecture includes a theory of learning, which constrains the information that can be encoded by the model, the circumstances of attention and problem-solving that cause such learning to occur, and the cues necessary to retrieve the acquired traces. From the dual constraints of human process data and cognitive architecture, our approach reveals general mechanisms for maintaining access to information in a rich and dynamic environment.

These mechanisms lead us to a theory of *episodic indexing*, in which we make two main claims. The first claim is that people automatically encode in long-term memory the event of attending to an object. (The resulting memory trace is episodic in that it represents an event.) The second claim is that when an object is hidden, people may imagine attending to it. The purpose of this imagery is to trigger any recollections for having actually attended to the object, which would indicate the object's existence and would be a basis for reasoning about whether to bring it back into view. (This is analogous to checking an index to see if an item is listed, and if so reasoning about whether to follow the pointer.) The theory predicts that successful access to external information requires the interaction of episodic and semantic knowledge. Episodic knowledge represents the comings and goings of objects in the world, and semantic knowledge is used to generate the images that make this episodic knowledge available when it is relevant. Thus effective use of external information, which is often highly dynamic, is mediated by relatively static domain knowledge.

The article is organized as follows. We first describe the task environment and the data we studied. We then describe the model, using simple hypothetical examples to introduce its encoding and retrieval processes. We then step through the model's simulation of two episodes from our data, one in which there is encoding and one in which there is retrieval. We next describe the model's fit to the protocol data at a higher level, and illustrate the effects of pervasive learning with data on how many productions are acquired and fired during the simulation. We then examine episodic indexing in the context of related theories and findings, including the source monitoring framework (Johnson, Hashtroudi, & Lindsay, 1993) and the construct of long-term working memory (Ericsson & Kintsch, 1995). We conclude by summarizing the elements of the theory.

II. THE DATA

The programming session we studied was part of a long-term, multi-person project to create a large natural-language comprehension system (we refer to this system as the program). The programmer contributed regularly to implementation, and was thoroughly familiar with the theory behind the program and the production-system language in which the program was implemented. Her high-level goal for the session we observed was to make a specific change to the implementation. The session lasted 80 minutes, during which the programmer stepped through a single run of the program interactively at a fine grain. The interaction took place in a GNU Emacs process buffer, a virtual teletype with a prompt at the bottom at which a user can enter commands to a running process. The process with which the programmer communicated was a production-system interpreter into which the program had been loaded. In addition to issuing run commands to advance the program, the programmer issued query commands to request various kinds of state information from the interpreter, including the contents of data structures, the contents of the runtime stack, and elements of code (productions).

All information printed by the language interpreter appeared at the bottom of the process buffer, with Emacs automatically scrolling old output off the top of the screen when more room was needed at the bottom. The old output remained accessible, but had to be scrolled back into view using keyboard commands. Thus the screen was functionally a window onto the process buffer, with the default window location being over the command prompt at the bottom, but with window location adjustable through scrolling commands. We instrumented Emacs to record the contents of the process buffer, as well as a time-stamped keystroke protocol. The programmer thought aloud, and we videotaped the screen, her utterances, and her gestures.

At various times the programmer scrolled the window across large amounts of old output in the process buffer. (She also used string searching, but only three times, and only one search was successful.) She used two kinds of scrolling commands, one that scrolled a screenful of text and one that scrolled a line of text. Both kinds were issued using keystrokes (as opposed to a mouse). We coded a string of consecutive scrolling commands as one scrolling *incident* that redisplayed a particular target object. There were 26 incidents in the 80-minute session, for a mean rate of one incident every three minutes. The total amount of text scrolled was 2482 lines under a window of 60 lines, or roughly 41 screens, underscoring the importance of hidden external information in real tasks. Understanding the memory processes that manage access to external information will contribute to improved human factors of interfaces and information systems.

In roughly two-thirds (17/26) of our scrolling incidents, the target had been hidden for the preceding 30 seconds or longer. Given the programmer's cognitive activity during the intervals since each target was last visible, it is implausible that information about the target could have persisted in short-term or working memory (WM) alone. Thus, the programmer seems to have made use of long-term memory (LTM) to store information about target objects she returned to later.[1]

To understand the nature of information stored in memory, it helps to have as much data as possible on the circumstances under which storage occurred. In the session we observed, the target in several scrolling incidents was an object that had been generated dynamically in the course of the session, as opposed to existing beforehand. In particular, for eight of the scrolling incidents in which LTM was implicated, the target object had been generated in the process buffer during the session. (For the remaining nine events, the target was in a file that existed prior to the session and which the programmer had looked at many times in the past.) Thus for these eight events, we were able to determine when the programmer's memory for the target must have been formed, and were able to study the protocol data and screen contents from that period.

From the 80-minute session we selected a 10.5 minute interval to model. This interval contains five scrolling incidents, providing a relatively high concentration of navigation activity. The five events are shown in Figure 1, which miniaturizes all the program output generated during the 10.5 minute interval. Rectangles represent windows on the buffer. Windows at the head of an arrow contain targets. Windows at the tail of an arrow contain the information on the screen when the programmer decided to scroll to a target. LTM is implicated in incidents 1, 3, and 5, in which the target was hidden for at least 30 seconds. LTM is not necessarily implicated in incidents 2 and 4, in which the target is hidden for

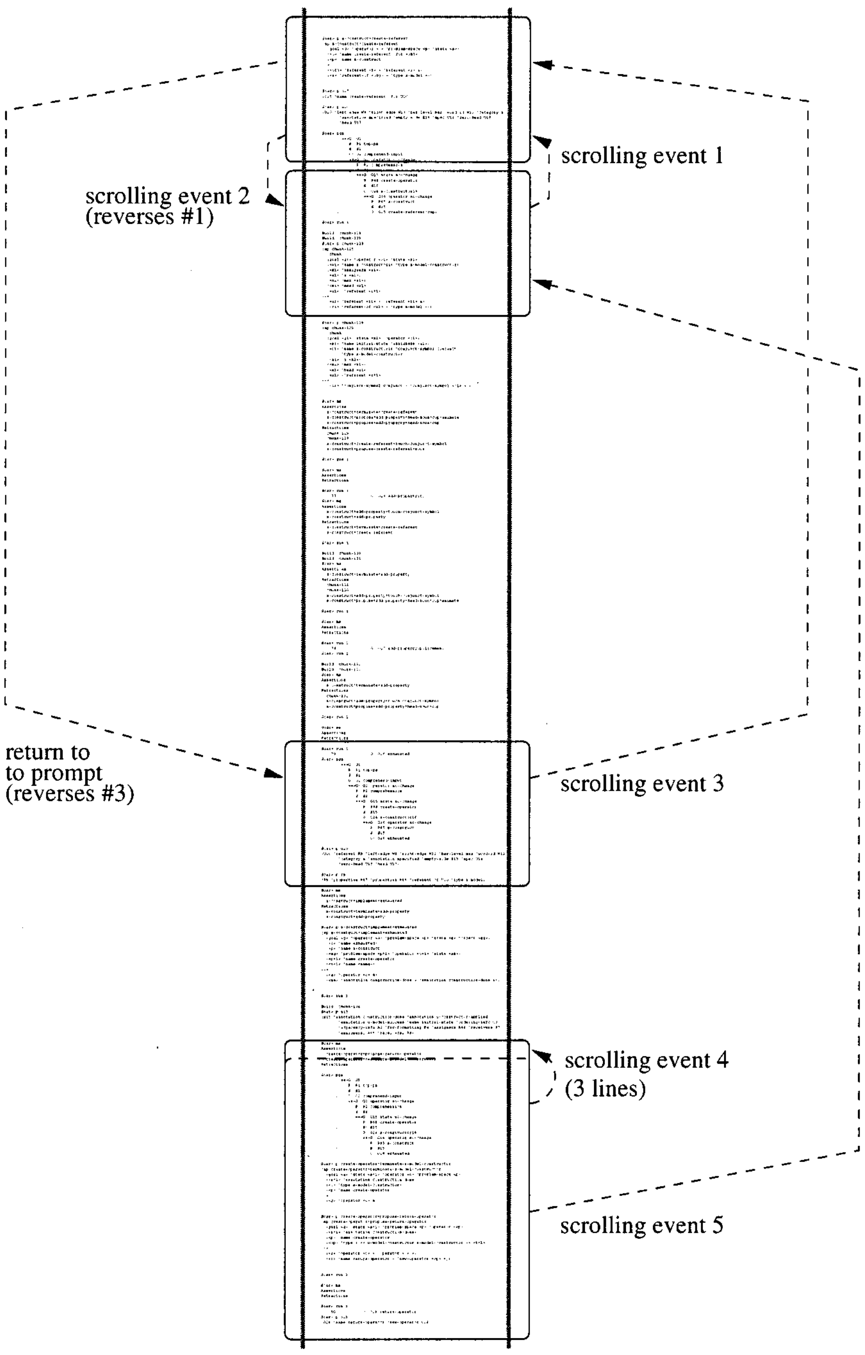


Figure 1. The five scrolling incidents accounted for by the model (Altmann, 1996). The contents of the programmer's buffer, generated interactively during the session, run down the center. Windows on this buffer (shown as squares) are connected by scrolling incidents (shown as dashed lines). The simulation ends after incident 5. This article examines incident 3.

less than 15 seconds. In these latter incidents, a trace of the target might have remained active in WM while the target was hidden.

Altmann (1996) provides a comprehensive description of the model, the data, and the correspondence between the two. Altmann, Larkin, and John (1995) and Altmann and John (1995) discuss scrolling incident 5 in the abstract and in detail, respectively. In this paper we focus on incident 3, but first turn to a discussion of the model itself.

III. THE MODEL

In the sections below we first introduce the model's performance, by describing its goals, subgoals, and actions at a high level. We then describe how the model constructs and uses an episodic index of the environment. The model brings to its task static knowledge representing expertise in the domain of programming. It uses this static knowledge to identify dynamic objects as they appear on the screen. As the model attends to these objects, Soar's learning mechanism encodes each attention event in LTM, constructing the episodic index. Later, if the object is hidden from view, the model can look it up in the index by imagining the attention event, which may trigger a memory for the event having occurred. This kind of imagery requires additional static knowledge of at least two kinds. First, the model must be capable of recalling the image of an attention event from memory. Second, the model must have some sense of the object's relevance to the task at hand, which implies a semantic understanding of how the object relates to other objects in the domain.

Characterizing the Model's Performance

The model's main mode of performance is a kind of comprehension—it tries to gather information about objects in its environment. This is a simplified representation of the programmer's primary activity during the modeled interval, in which she is reminding herself how the program works by stepping through it in detail. The model does not construct the complex mental structures associated with comprehension of text in general (e.g., Kintsch, 1998; Lewis, 1993) and programs in particular (Boehm-Davis, Fox, & Philips, 1996; Brooks, 1983; Davies, 1994; Detienne & Soloway, 1990; Gray & Anderson, 1987; Green, Bellamy, & Parker, 1987; Holt, Boehm-Davis, & Schultz, 1987; Von Mayrhauser & Vans, 1995; Pennington, 1987a; Rist, 1995). The purpose of the model is less to represent the problem-solving processes of programming than to investigate the acquisition of knowledge that indexes the environment.

The model selects goals to comprehend program objects, for example a code fragment or a data structure. The model also issues commands to change the display. Some commands generate new information, and some scroll to old information. The model uses this external information as it tries to comprehend objects.

To comprehend an object, the model selects subgoals that retrieve information about the object. Information can come either from the screen (an external source) or from LTM (an internal source). For example, suppose the model is comprehending a data structure that represents a student record. The student record contains a field for the student's Social Security Number (SSN), which is displayed on the screen.[2] To retrieve information about this field, the model selects an *attend* subgoal. Suppose (for simplicity) that the model attends only to the field and not to the actual number stored there. This act of attention would add the following attribute-value pair to WM.

```
(^field ssn)              From attending to SSN field.
```

Alternatively, if this information is not available externally but the model has the appropriate domain knowledge, the same information can be recalled from LTM. To do this, the model selects a *probe* subgoal. For example, the model might probe with the SSN field, perhaps to see if this activates any other information relevant to the student record. Probing and attention are symmetrical in that a probe can look exactly like the output of attention.

```
(^field ssn)              From probing with SSN field.
```

Under episodic indexing, attention and probing generate another kind of element, one which represents the actual event of attending to an object. Attention automatically adds this element to WM as a side effect of attending to an object. Thus the full outcome of attending to an SSN field would be the following.

```
(^field ssn)                      From attending to SSN field.
(^attended-to ssn)                From attending to SSN field.
```

The same representation could also be produced by a probe, consistent with the attention-probing symmetry noted above. The probe below represents the model asking itself, "What do I know about the event of attending to an SSN field?"

```
(^field ssn)                      From probing with SSN field.
(^attended-to ssn)                From probing with SSN field.
```

We refer to an attribute-value pair like attended-to ssn, when generated by a probe, as an *image* of attending to an object. The term image is meant to suggest a code like that produced by attention, namely more like a percept than an abstraction or a concept. Beyond this, we do not attempt to interpret the model's images phenomenologically, or psychologically in terms other than how they function in the model. For example, their symbolic nature reflects Soar's representation language and is not intended as a statement in the debate over propositional vs. analog spatial codes (reviewed, for example, in Humphreys & Bruce, 1993). In general, LTM contains many kinds of codes (Bower, 1975), and in particular expert programmers often use vivid imagery to understand programs, including color, sound, and dancing symbols (Petre & Blackwell, 1997). Amidst this diversity it seems reasonable to posit a code representing the event of attending to an object.

Thus the model can imagine attending to an object, providing it has the knowledge to do so. Such imagining, and hence the requisite store of images, is the basis of the retrieval

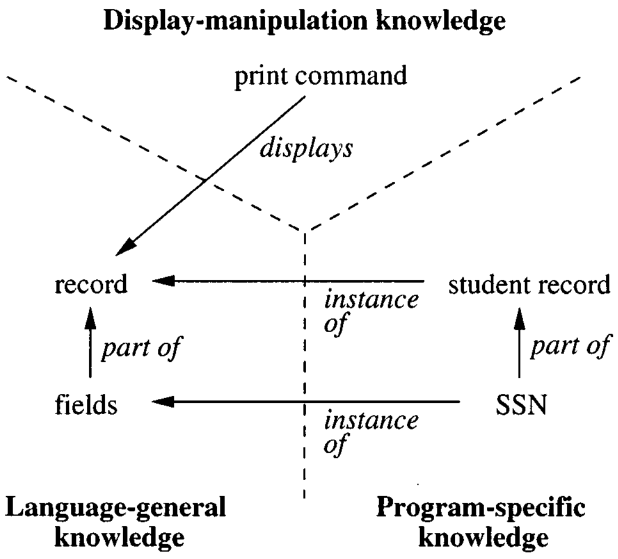


Figure 2. Kinds of static domain knowledge in the model.

processes of episodic indexing. The static knowledge of which imagining knowledge is a component is discussed in the next section.

Static Knowledge Brought to the Task

The model contains domain knowledge of the kind we would expect a skilled programmer to bring to a programming task. This knowledge is static in that it is loaded before the simulation begins and persists in the same state throughout. It consists of knowledge about the particular program to be modified, including its structures and processes; about the implementation language, including its central concepts and idioms; and about computer science fundamentals, like data structures and algorithms. Such knowledge is typically found in expert systems and other symbolic AI programs. The model also contains knowledge about the programming environment, including procedures for manipulating the display. This is the kind of expertise represented in the operators, methods, and selection rules of GOMS models (Card, Moran, & Newell, 1983; John & Kieras, 1996).

Figure 2 shows examples of each kind of static knowledge, drawing from the illustrative domain introduced in the previous section. The model might know about language-general constructs like records and fields (on the left), about instances of these constructs specific to a particular program (on the right), and about how to print such instances (on top).

Static knowledge serves several purposes in the model. It indicates which program objects are important to comprehend, letting the model propose these objects as comprehension goals. It identifies program objects in the environment, letting the model attend to them. For example, the model would have to know that student records are important to comprehend, and would have to be able to identify parts of a student record like the SSN field. The model also has to know what domain objects look like, in order to generate images, and how they are related to each other, to generate images at the right time. For example, if the model were asked whether a particular database contained confidential information, it might ask itself whether it had seen an SSN field in any of the records it had examined. Knowledge that such a field was present would indicate that the database did in fact contain confidential information.

Two other categories of static knowledge need to be introduced (though neither are detailed in Figure 2). First, there is program-specific knowledge that draws the model's attention to features missing from the screen. For example, the model might know to look for an SSN field in a record it prints out, and might attend to the fact that the field is absent. This absence might indicate that the record is not yet fully constructed, which might in turn provide useful debugging information about the program's execution status. This absence-detecting knowledge plays a role in the sample simulation discussed later.

A second category of static knowledge supplies generic mechanisms that appear to be very general, but which have not yet been proven so general as to be architectural. For example, one such mechanism implements internal attention, allowing the model to move from one comprehension goal to the next in a coherent but flexible train of thought (Altmann, 1996). A second mechanism implements a heuristic preference for attending to the output of the most recent query to the interpreter, as compared to older regions of output. Generic mechanisms are worth noting because they emphasize the range of generality at which knowledge contributes to performance. Performance depends on knowledge ranging from generic mechanisms that may stay constant over time and apply in many kinds of problem solving, to dynamic information about a specific and changing external situation. This dynamic information is discussed in the next section.

Dynamic Information Arising During the Task

Dynamic information arises as the model performs its comprehension task. It arises both outside and inside the model. External dynamic information consists of objects appearing on the screen, for example in response to queries to the language interpreter.

Internal dynamic information consists of new symbols (gensyms, in Lisp terms) generated regularly over the course of time. A new time symbol is generated when the model selects a new comprehension goal (meaning that the model's sense of time is keyed to its train of thought). All objects attended during the current goal are encoded in LTM with the current time symbol. Later, if the model can recall a time symbol from LTM, it can compare the recalled symbol to the current time, which is always available. If the two are different, then the recalled symbol designates a past attention event.

This identity comparison is the only computation on time symbols that the model supports. Thus time is categorical, rather than ordinal or interval, and the only categories are present (the current comprehension goal) and past (any previous goal). The model cannot compute, for example, the interval between two events. This information-leanness is consistent with qualitative aspects of the rapid decay of unelaborated temporal codes in people (Underwood, 1977).

Some form of temporal coding or temporal inferencing seems necessary to be able to index a dynamic environment, where new objects appear continually and where any one of them may be worth remembering after disappearing from view. Our model indexes such objects by mapping attention events to new symbols, and by encoding this mapping in LTM. How this encoding occurs is discussed in the next section.

Learning in Soar: Acquiring New Productions

All the model's knowledge is represented as productions. These are condition-action rules like the one below. If the condition part (above the arrow) matches a structure in WM, then the action part (below the arrow) adds new elements to WM. The production below acts as a declarative memory, because it associates an object (a student record) with facts about that object (that it has an SSN field). In general, all the model's operations, like attending to objects, generating probes, and recalling facts, depend on knowledge represented as productions.

`(^structure student-record)`	Condition: Student record in WM.
$\rightarrow$	
`(^field ssn)`	Action: Put SSN field in WM.

The model learns by acquiring new productions. The learning mechanism is part of Soar, the cognitive architecture in which the model is implemented. Soar's learning mechanism is unified with its knowledge-representation language (productions) and control structure (goals), in that Soar acquires new productions in response to achieving goals (Laird, Rosenbloom, & Newell, 1986). A new production, or *chunk*, represents an inference that may have taken several steps to make.[3] The chunk is added to LTM, making the inference available in a single step from then on.

Figure 3 shows an example of chunking (after Howes & Young, 1998). The goal (shown at the top left) is to add two numbers (2 and 3). Two subgoals (smaller circles to the right) each add one of the addends to a running sum. The first subgoal initializes the running sum to 0, then adds 2. The second subgoal acts on the fact that there are no more addends, inferring that the current sum achieves the goal of adding the addends. Specifically, the subgoal designates 5 as the result of adding 2 and 3. This causes Soar to build a chunk, which in the future will compute $2 + 3 = 5$ directly, without subgoals.

In general terms, a chunk encodes an association between an inferred result (e.g., the sum) and the WM elements on which the inference is based, which we refer to as premises (e.g., the addends). Problem-solving productions infer the result from the premises, often in multiple steps. When the result is generated, the chunking mechanism identifies the premises by recreating the problem-solving trace backwards, starting from the result (Altmann & Yost, 1992; Laird, Rosenbloom, & Newell, 1986; Howes & Young, 1998). The backtracing algorithm stops when it reaches elements that have already been designated as contributions to the current goal. These elements are taken to be the premises of

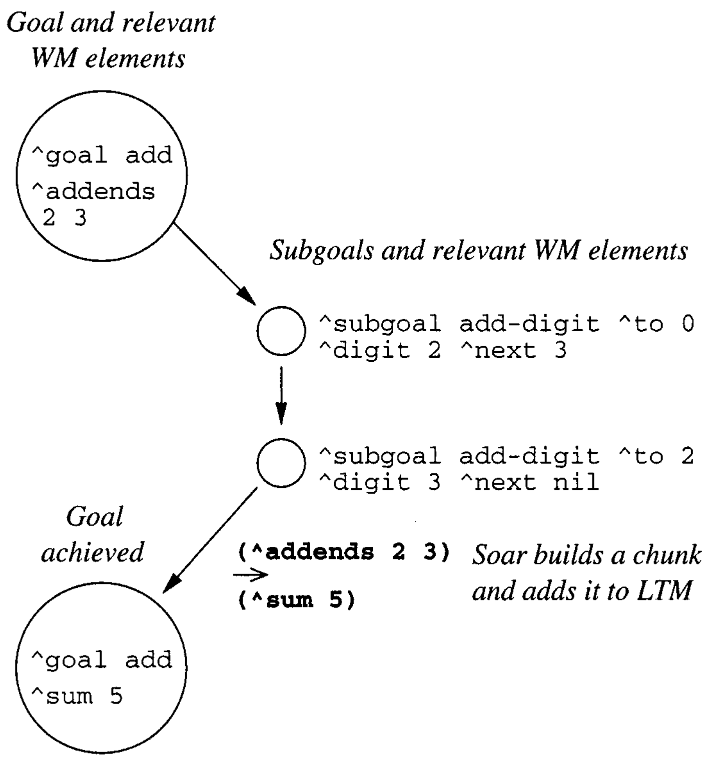


Figure 3. Soar adding two numbers and capturing the process in a chunk. Soar begins with a goal to add 2 and 3 (top left), selects two subgoals that each add a number to a running sum (initially 0), and finally achieves the goal by returning 5 as the result (bottom left). Upon achieving the goal, Soar encodes a chunk that will add 2 and 3 directly (without subgoals), should this goal arise in the future.

the new chunk. In the future, should the same goal and premises appear in WM, the new chunk will fire immediately.

The chunking process does very little induction or generalization. The result essentially becomes the chunk's action and the premises become the chunk's conditions, though there is some variabilization (Laird, Congdon, Altmann, & Doorenbos, 1993; Laird, Rosenbloom, & Newell, 1986; Newell, 1990). This makes a chunk specific to its encoding context, consistent with the encoding specificity principle (Tulving, 1983). This specificity acts as a hard constraint on the processes of acquiring and retrieving knowledge (Howes & Young, 1998). In the next two sections, we describe these processes as they occur in our model.

Constructing the Episodic Index

The model contains two key assumptions about attentional processing. Both assumptions are related to the event of deliberately attending to an object. The first assumption is that the event itself is worth representing in WM, apart from the object of attention. The second assumption is that attention events are goal-directed, in that they are initiated in service of comprehension goals. This second assumption says that the model is always looking for new information about the object it is trying to comprehend, and therefore automatically takes any deliberate attention event to contribute to the current goal. The two assumptions together operationalize what we might think of as "paying attention to" or "concentrating on" what we are doing. The important implication is that if the model "pays attention" to an event, this enables remembering the event because it causes chunks to be acquired.

The first assumption (that attention events are noteworthy) is implemented using the dynamic internal information described earlier. When the model attends to an object, it records the event using its internal clock. Specifically, the model associates the WM code for the attention event with the current time symbol. For example, when the model attends to the SSN field of a student record, the complete representation created in WM is something like the following.

```
(^attended-to ssn)             From attending to SSN field.
(^event ssn ^time t42)         From attending to SSN field.
```

The second assumption (that episodic processing contributes to the current goal) is implemented by designating the event/time-symbol element a contribution to the current goal. This causes Soar to build a chunk, as described in the previous section. The premise of the chunk is an element representing attention to the SSN field, and the result is an element representing the event/time-symbol association (though we typically refer only to the time symbol returned by an episodic chunk). We have named the chunk attended-ssn, for reference later.

```
chunk: attended-ssn            Chunk capturing an attention event.
 (^attended-to ssn)
 →
 (^event ssn ^time t42)
```

Attended-ssn encodes an attention event. This makes it an episodic trace, as distinct from a semantic trace with no temporal content (Tulving, 1983). It functions as one entry in a large index of attention events. In the future, if no SSN field is visible, the model can look up the SSN field in this index by attempting to cause this chunk to fire. If the lookup is successful (i.e., if the chunk fires) then the model can infer that it attended to an SSN field in the past, even though no such field is currently visible.[4] The lookup and inference processes are described in the next section.

Using the Episodic Index

Suppose that a particular attention event occurred long enough in the past that it is no longer active in WM and that the corresponding object is no longer in view. The model can use its episodic index to see if the object exists somewhere in the environment. This requires two steps. The first step is to generate the cue necessary to get an episodic chunk to fire. We can think of this as “looking up” the object. The second step is to make the appropriate inferences based on any recalled time symbols. We can think of this as acting on the information retrieved from the lookup.

To look up an object, the model must add the appropriate image to WM as a cue for triggering episodic chunks. As discussed previously, an image can appear in WM either through attention, which generates the image from an external stimulus, or through probing, which generates the image from memory. In either case, an image appearing in WM will activate all episodic chunks acquired whenever the corresponding object was attended in the past. Production imagine-ssn, below, generates the necessary image for the SSN field. (The processing described below is also summarized in the Appendix.)

```
production: imagine-ssn
 (Conditions testing that it's relevant to know that an SSN field was seen)
 →
 (^attended-to ssn)                                  A1
 (^imagined ssn)                                     A2
```

Imagine-ssn will fire in a situation in which it would be useful to remember seeing an SSN field. For instance, suppose (as we did earlier) that the model is asked whether a database contains confidential information. The model might try firing imagine-ssn to trigger an episodic chunk like attended-ssn. Should attended-ssn fire, A1 would generate an image and A2 would tag this image as generated from memory rather than from a stimulus. There could in general be many situations in which it might be useful to imagine an SSN field. Each would be represented in a production like imagine-ssn (with different conditions).

Should imagine-ssn fire and attended-ssn fire in response, then WM would contain the following elements.

```
(^attended-to ssn)              From imagine-ssn.
(^imagined ssn)                 From imagine-ssn.
(^event ssn ^time t42)          From attended-ssn.
```

From these elements the model can infer that an SSN field exists in the environment. The production that makes this inference is recall-seeing-object, below.[5] This production belongs to the generic mechanisms that are part of the model’s static knowledge.

```
production: recall-seeing-object
 (^attended-to <o>)                                  C1
 (^imagined <o>)                                     C2
 (^event <o> ^time <then>)                           C3
 (^time <now> ≠ <then>)                              C4
 →
 (^recall-seeing  <o>)
```

Recall-seeing-object's conditions, numbered on the right, are as follows. Conditions C1 and C2 test that there is an image in WM that was generated internally rather than from an external stimulus.[6] C3 and C4 test that the image was attended in the past. The single action summarizes what is expressed by the conditions. It adds to WM the recollection of having seen the object.

The nature and use of the episodic index is shaped by Soar's constraints on learning. Because Soar makes a chunk specific to its encoding context, the attended-ssn chunk is specific to the object code that appeared in WM during the attention event. This specificity implies that recalling the existence of an object must be preceded by imagery involving the object.

Summary of Assumptions and Implications

There are four theoretical assumptions that shape how the model acquires and retrieves memories for attention events. The first two concern details of human attention, representing a model increment (Howes & Young, 1998) that fills gaps in the Soar theory. The first assumption is that attention events themselves are worth symbolizing in WM, in addition to the attended objects. The second assumption is that deliberate attention is an integral part of comprehension, and thus that attention events automatically contribute to the current comprehension goal.

The third and fourth assumptions are part of the Soar theory. The third is that knowledge that contributes to achieving a goal is stored permanently in chunks. The fourth is that chunks are specific to their encoding context.

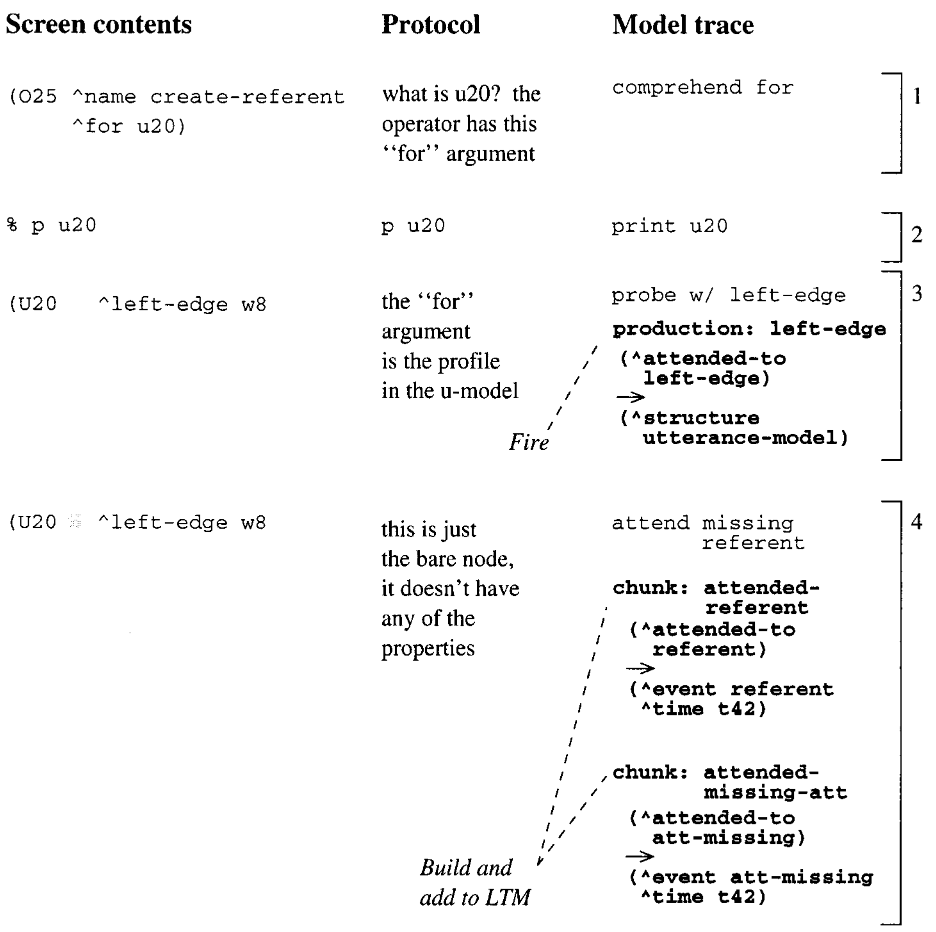
Together, these assumptions imply that chunk acquisition in the model will be pervasive and automatic, and that retrieval will be effortful. Learning will be pervasive because the model will encode a new episodic chunk for every object it attends to. This learning is automatic, in that the model exercises no choice over whether or not to learn, and in that learning is a side effect of attentional processing rather than an end in itself. Retrieval will be effortful because learning involves little induction; to get chunks to fire, the model must recreate the encoding context from memory.

Having described the basic processes of episodic indexing, we turn to an example of how they are applied in simulating our data.

III. A SIMULATION OF ONE SCROLLING INCIDENT

This section describes and interprets scrolling incident 3, one of the five incidents covered by the simulation. The incident really consists of two episodes. The first episode is when the programmer prints an object and attends to it. We refer to this object as the target, and to this episode as the acquisition episode because this is when the programmer must acquire a memory for having seen the target. In the minutes that follow the acquisition episode, the programmer's work produces output that displaces the target from the screen.

The second episode contains the actual scrolling action. It occurs four minutes (and two screenfuls) after the target is displaced from the screen. The programmer becomes



Screen contents	Protocol	Model trace	
`(O25 ^name create-referent ^for u20)`	what is u20? the operator has this "for" argument	`comprehend for`	1
`% p u20`	`p u20`	`print u20`	2
`(U20 ^left-edge w8`	the "for" argument is the profile in the u-model	`probe w/ left-edge` **`production: left-edge (^attended-to left-edge) --> (^structure utterance-model)`**	3
`(U20 ^left-edge w8`	this is just the bare node, it doesn't have any of the properties	`attend missing referent` **`chunk: attended-referent (^attended-to referent) --> (^event referent ^time t42)`** **`chunk: attended-missing-att (^attended-to att-missing) --> (^event att-missing ^time t42)`**	4

Figure 4. The acquisition episode, beginning with a goal to comprehend the object linked to the for attribute (step 1 in the rightmost column), followed by printing the object (step 2), probing with attributes of the printed object (step 3), and attending to an attribute that is missing (step 4). The model learns episodic chunks for the attention event in step 4.

reminded of the target, scrolls back to it quickly, attends to it briefly, then returns to the prompt at the bottom of the buffer, where she issues a command to print a fresh copy of the target. We refer to this sequence of events as the retrieval episode because it encompasses the retrieval of the memory acquired during the acquisition episode.

Acquisition Episode

Figure 4 shows the acquisition episode. The screen contents (what the programmer sees) appear in the left column, abridged to show only the objects referred to in the verbal protocol. The programmer's verbal and keystroke protocols appear in the middle column.

The corresponding model trace, consisting of goals, subgoals, and commands, appears in the right column.

The acquisition episode begins with the programmer wanting to know about a data structure. This structure is denoted by the symbol u20, which is a unique identifier generated at run-time by the program. The structure itself is not displayed on the screen, but u20 is, so the programmer prints u20. (To be precise, the programmer prints the data structure identified by u20, but for simplicity we often refer to the data structure by its identifier.) The system responds, and the programmer recognizes the printed object as a u-model (standing for utterance model, a structure specific to the program's domain of natural language comprehension). The programmer then notices that a key field of the u-model is missing because the program has not constructed it yet. The missing field is called the referent, which is a substructure that will hold the "properties" that the programmer refers to in the verbal protocol. In simulating this behavior, the model also notices the missing referent. This creates the episodic chunk used later to infer that a u-model is hidden off-screen.

Below we step through the events of this episode in greater detail. The numbered points in the text correspond to the numbered steps at the far right of Figure 4. In the text we will identify the kinds of static knowledge (as shown in Figure 2) used by the model in its simulation. Note that for clarity the model trace in Figure 4 is highly abstracted, showing only highlights of the model's activity; detailed traces appear in Altmann (1996). We pick up the episode at the point where the programmer wants to know more about u20, which she has not yet printed nor recognized as a u-model.

1. Goal to comprehend an object. The programmer refers to the for attribute and to its value u20, which identifies a structure that the programmer wants to know more about.
   The model uses program-specific knowledge to recall that for is an important attribute (this recollection is not shown in Figure 4). This recollection prompts the model's internal attention mechanism to select a goal to learn more about the attribute and its value. Hence the model selects a goal to comprehend for.
2. Command to print the object. The programmer enters a query to print u20 (p u20, where p denotes print).
   The model also issues a print command (print u20), based on its display-manipulation knowledge of what commands help to achieve which goals.
3. Recognize the printed object. The language interpreter responds by printing u20. The programmer recognizes it as a u-model (saying, "The 'for' argument is the profile in the u-model").
   The model's print command is interpreted by a display emulator, which responds by adding a representation of u20 to the emulated screen. The model identifies u20 by its attributes. The model attends to left-edge, then probes with left-edge to see what it can recall. The probe retrieves program-specific, semantic knowledge that recognizes left-edge as part of a u-model. This is represented in Figure 4 as the firing of production: left-edge.
4. Attend to missing attribute and acquire memories for the event. The programmer sees that the u-model is missing something, namely a referent field. The referent is itself a

> data structure that will contain the properties attribute that she refers to in the protocol. She knows that the referent will be constructed at some point during program execution. The referent appears later, in front of the left-edge attribute (in Figure 4 the position is marked by a grey dot, which does not actually appear on the programmer's screen).
>
> The model also knows to look for a missing referent. From its program-specific knowledge, it knows that the referent will be constructed at some point during program execution, so it looks to see if the construction has occurred yet. The chunks acquired by this act of attention are described below.

The protocol data by themselves do not say what kind of information the programmer encoded about the u-model during this episode. Some information must have been stored in memory, but the direct evidence for this is the scrolling action in the retrieval episode. There is no similarly direct evidence in the acquisition episode. In particular, there is no evidence that the programmer intends to remember the u-model prospectively for future reference (Brandimonte, Einstein, & McDaniel, 1996; Patalano & Seifert, 1997).

The model provides an operational hypothesis about what information was stored in memory. In the fourth step above, attending to the missing referent adds two elements of information to WM.[7]

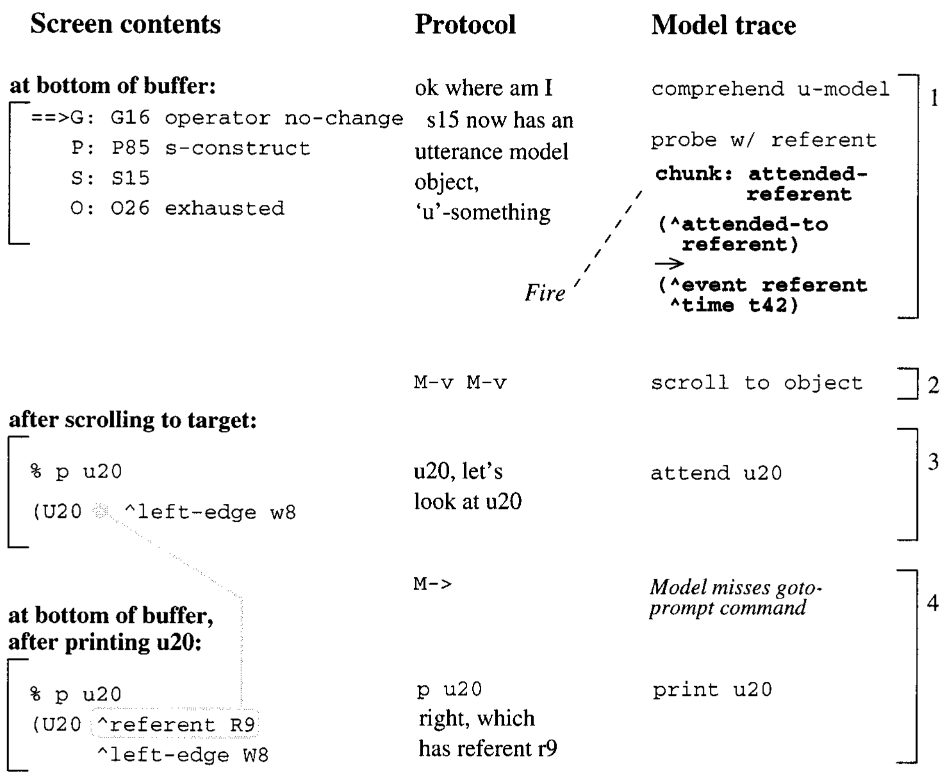
```
(^attended-to referent)            Looked for a referent,
(^attended-to att-missing)         but it was missing.
```

At this point the theoretical assumptions outlined in the previous section come into play: (1) that attention events themselves are represented in WM; (2) that they contribute to the current goal; (3) that anything that contributes to the current goal is chunked in LTM; and (4) that chunks are specific to the encoding context. The outcome is that the model tags the WM elements above with the current time symbol and creates two new chunks.

```
chunk: attended-referent          Chunk capturing an attention event.
 (^attended-to referent)
 →
 (^event referent ^time t42)

chunk: attended-missing-att       Chunk capturing an attention event.
 (^attended-to att-missing)
 →
 (^event att-missing ^time t42)
```

These chunks (also shown in Figure 4, step 4) permanently store the event of attending to a missing referent. They are two of several chunks encoded when the model attends to the various parts of the target. Another of these attended parts is u20. However, because identifiers like u20 are generated dynamically by the program, the model cannot predict which symbol will identify a given object. Thus it cannot use its static domain knowledge to generate identifiers as cues, which means that it cannot use imagery to trigger the chunk


Screen contents
Protocol
Model trace
at bottom of buffer:
after scrolling to target:
at bottom of buffer, after printing u20:
Fire
1
2
3
4

```
==>G: G16 operator no-change
    P: P85 s-construct
    S: S15
    O: O26 exhausted
```

ok where am I s15 now has an utterance model object, ‘u’-something

```
comprehend u-model

probe w/ referent
chunk: attended-
        referent
(^attended-to
  referent)
-->
(^event referent
  ^time t42)
```

M-v M-v

```
scroll to object
```

```
% p u20
(U20 ... ^left-edge w8
```

u20, let’s look at u20

```
attend u20
```

M->

Model misses goto-prompt command

```
% p u20
(U20 ^referent R9
     ^left-edge W8
```

p u20 right, which has referent r9

```
print u20
```

Figure 5. The retrieval episode, beginning with a goal to comprehend the u-model (step 1 in the rightmost column), followed by scrolling (step 2), attending to the scrolled-to object (step 3), and returning to the command prompt at the bottom of the buffer to print a fresh copy of the object (step 4). In the protocol data, M-v (meta followed by v) scrolls one screen toward older output, and M-> (meta followed by >) returns the cursor to the command prompt. The model's scrolling decision is based on the attended-referent chunk firing in step 1.

for u20. Instead, it triggers one of the chunks above (attended-referent), as described in the next section.

Retrieval Episode

Figure 5 shows how the attended-referent chunk is retrieved. As before, screen contents are on the left, protocol in the middle, and model trace on the right. The u-model printed during the acquisition episode has been off-screen for four minutes, and is two screenfuls away from the information currently visible. To summarize the episode, the programmer is reminded about the utterance model under construction by the program, scrolls quickly back to the hidden copy of the utterance model that she printed during the acquisition episode, attends to the target briefly, then returns to the command prompt, where she prints a fresh copy of the utterance model.

1. Comprehend u-model. Looking at the current screen, the programmer makes a comment about the u-model (using the unabbreviated term “utterance model”). The basis for this comment appears to be the s15 identifier that is visible on the screen. (The u-model is indirectly attached to s15, but neither the u-model nor the attachment to s15 are visible.) The programmer cannot recall the u-model identifier, beyond its prefix (“ ‘u’-something”).

   The model selects a goal to comprehend the u-model (comprehend u-model). It selects this goal based on domain knowledge about the program that associates cues on the screen with the existence of a u-model.[8] It then tries to elaborate its knowledge about the u-model. Specifically, it tries to recall seeing attributes of the u-model. Knowing which attributes are present would provide information about how far the program has run.

   One u-model attribute is the referent. Based on its knowledge of the program, the model knows that the referent will be constructed at some point during execution. Therefore, the model probes with an image of attending to a referent (probe w/ referent). If this triggers an episodic chunk, then the referent might exist already, providing a clue about how complete the u-model is.

   Probing with a referent requires two kinds of knowledge. First, the model must have program-specific knowledge that lets it call an image to mind in the absence of the external stimulus. Second, it must have program-specific knowledge about the relationship between the referent and the completeness of the u-model. This knowledge makes it relevant to probe with a referent when comprehending a u-model.

   Once in WM, the referent image triggers one of the chunks encoded during the acquisition episode (attended-referent). The chunk adds the time symbol t42 to WM. This time symbol in turn triggers the recall-seeing-object production (discussed in the section, Using the Episodic Index), which adds to WM the knowledge that the model looked for a referent in the past. Using its domain knowledge about the structure of program objects, which says that a referent is part of a u-model, the model infers (not shown) that there must be a u-model off-screen.
2. Scroll to hidden u-model. The programmer scrolls back to the u-model that she printed during the acquisition episode. She issues two keyboard commands in quick succession (less than one second apart), each of which scrolls one screen (M-v, M-v).

   The model knows to scroll to a hidden target when it is trying to comprehend an object of that kind. This is a generic mechanism, representing heuristic knowledge that hidden objects can be useful sources of information. The model is still trying to comprehend the u-model, so it scrolls to the hidden copy.
3. Attend to u20. Back at the acquisition screen, the programmer sees the u20 identifier. The model also attends to u20, based on a generic mechanism that says to look at the parts of a target that has been effortfully revisited.
4. Print fresh copy of u-model. The programmer issues an editor command to return to the prompt at the bottom of the buffer (M->), where she types a command to print u20 (p u20). The utterance model now has the referent that was missing during the acquisition episode.

   The model also prints u20, based on display-manipulation knowledge.[9] The print command is contingent on two aspects of the situation. First, the identifier, necessary for print commands, is now known. Second, the u-model has likely been modified since the scrolled-to copy was printed, meaning that a fresh copy would be useful.

The programmer may employ spatial knowledge in this episode that is not represented in our model. The two keystrokes that redisplay the target are separated by less than a second, leaving little time to scan for the target on the intermediate screen. This suggests that the programmer not only has a robust memory for the target, but also has an accurate notion of how far away the target is.

In contrast, the model has little spatial knowledge about the display. It has no knowledge of "screens" as such, nor of "scrolling up" vs. "scrolling down". It knows only the difference between visible objects and previously-seen hidden objects. It also knows that it can bring a hidden target into view by issuing an abstract (directionless) scroll command.[10] This simplified world knowledge means that the model has no representation of how far away the u-model is, nor of the direction in which it is located.

These simplifications leave open the possibility that the programmer acquires a richer episodic index than the model does. This would be consistent with findings, for example, on the incidental acquisition of spatial knowledge (Andrade & Meudell, 1993; Lansdale, 1991; Lovelace & Southall, 1983; Nygren, Lind, Johnson, & Sandblad, 1992). However, our objective is to demonstrate simple encodings and memory processes that are sufficient to enable purposeful access to hidden objects. The simpler we can make the representation, the closer it approximates a core that could be the basis for many variations of episodic indexing.

An Alternative Model

In many respects a correct cognitive model is impossible to identify (Anderson, 1990; Newell, 1990). In particular, in the example above the programmer might have scrolled to the u-model for a different reason than we attribute to her in our model. Here we discuss a plausible alternative to our model, and argue that it also requires episodic indexing.

Our model's account of the retrieval episode is represented as a pseudo-protocol on the right of Figure 6, aligned with the programmer's protocol on the left. To summarize, the model remembers seeing a u-model and revisits it for general information-gathering purposes.

A plausible alternative account is shown in Figure 7, with an alternative model pseudo-protocol on the right. This account attributes a different goal to the programmer, which is to retrieve the u20 identifier specifically to include it in the ensuing print command.

Although these two accounts propose different reasons for scrolling, they both propose that scrolling was purposefully directed toward a target. This implies a distinct memory for the target, which is functionally what is represented in an episodic chunk. Moreover, both accounts imply failure to recall the u20 identifier itself. This is consistent with the programmer apparently recalling only "'u'-something" and not the numeric suffix that makes u20 unique. Thus the model's time symbols qualitatively reflect the level of detail in the programmer's memory for the u-model.

Programmer protocol	Model pseudo-protocol
1. Ok, where am I ... s15 now has an utterance model object, 'u'-something	1. Ok, the program is building a u-model. What do I know about the u-model? Does it have a referent yet? I remember looking for a referent earlier ...
2. `M-v M-v`	2. `scroll`
3. u20, let's look at u20	3. There's the u-model, but I've run the program a lot since I printed it, so let's print a fresh copy. The identifier is u20 ...
4. `M-> p u20` Right, which has referent r9	4. `print u20` The u-model now has a referent.

Figure 6. The programmer's protocol data from the retrieval episode (left column), aligned with a pseudo-protocol representing the model's account of the data (right column).

Knowledge Distinguished by Operation

The process by which the model gains access to hidden information breaks down into three component operations. Below we summarize these operations, and the knowledge upon which each operation is contingent.

1. Attention. During the acquisition episode, the model must know to attend to the u-model in the first place, and in particular to its attributes. This means that the u-model must be identifiable to the model, and must be understood as relevant to the task at hand.
2. Retrieval. During the retrieval episode, the model applies two kinds of knowledge: (a) the ability to generate a u-model attribute as cue, and (b) the ability to do this when the

Programmer protocol	Alternative pseudo-protocol
1. Ok, where am I ... s15 now has an utterance model object, 'u'-something	1. Ok, the program is building a u-model. I want to print out the u-model, but I can't recall its identifier. I do remember seeing an old u-model, so I'll get the identifier from it ...
2. `M-v M-v`	2. `scroll`
3. u20, let's look at u20	3. There's the identifier. It's u20 ...
4. `M-> p u20` Right, which has referent r9	4. `print u20` The u-model now has a referent.

Figure 7. The programmer's protocol data from the retrieval episode (left column), aligned with a pseudo-protocol representing an alternative to the model's account (right column).

results of a successful probe would be useful. Thus retrieval depends on both structural and semantic knowledge specific to the particular domain.

3. Scrolling. The decision to scroll is distinct from retrieval of the episodic chunk. The decision is based on knowledge that evaluates whether a hidden target is worth scrolling to. In general there might be other methods for generating the information that the target could provide, and scrolling to the target might be a relatively poor choice.

The distinction above between retrieval of an episodic chunk, which we as analysts have inferred, and scrolling, for which there is explicit evidence in the data, means that retrieval may occur more often than we can tell from the data. The programmer may have recalled other attention events and not acted on the recollections in a manner that we could observe.

Thus the model points to several operations by which domain knowledge mediates access to hidden information. Later we revisit this connection as it plays out in other theories of memory. First we step back from our sample scrolling incident and discuss the model at a higher and more quantitative level.

IV. BROAD MEASURES OF THE MODEL

The previous section examined one scrolling incident—two short segments of protocol data—and our model's account of it. In terms of elapsed time, this behavior spans roughly 20 seconds. The model's lifetime spans five such incidents and all the intervening behavior, accounting for 10.5 continuous minutes of problem solving (Altmann, 1996). This extended lifetime, spanning the halves of each incident, acted as a form of control in our analysis of the data. A sufficiently close examination of the data to construct a model was the best way to avoid missing events between acquisition and retrieval that might have recoded or otherwise affected the nature of the programmer's episodic index.

In this section we illustrate the overall fit of the model to the data. We then present data on productions acquired and fired to illustrate the implications of pervasive episodic learning.

Fit to Keystroke Data

We rely on the keystroke protocol to characterize the model's fit. The keystroke protocol was relatively unambiguous to code, and also contained explicit evidence of purposeful access to hidden information. The verbal protocol provided critical guidance in building the model, but was incomplete and difficult to code formally and hence difficult to use as a quantitative measure of fit.

We segmented the keystroke protocol first into commands and then into command sequences. There were 50 commands (e.g., M-v, which scrolls one screen). Consecutive scrolling commands (like M-v M-v in our scrolling example) were treated as one sequence. There were three such sequences (one with two commands and two with three commands). Similarly, we treated as a sequence consecutive interpreter commands that seemed to achieve one identifiable goal. For example, the programmer once issued three quick commands to run the program to a specific checkpoint, and we treated this as one sequence. There were four such sequences (one with two commands, and three with three commands). The target data to which we fit the model consisted of these seven sequences plus the 31 remaining commands that were not part of any sequence. Thus the target data contained 38 items, which for simplicity we refer to as commands.

The model issues 34 commands. These map, in correct sequential order, to 34 of the 38 commands in the target data. Of the four commands in the target data for which the model fails to account, two brought the command prompt back into view in preparation for entering a new interpreter command. The model has no representation of the prompt, and hence no knowledge of these commands. The two other missed commands each ran the program one cycle. The model accounts for 10 run commands, but lacks the knowledge to issue these two. The display emulator compensates for all four misses by, in effect, issuing the missed commands to itself at the appropriate time.

Production and Firing Counts

An important way to characterize a model is to examine the kinds of knowledge it contains. This is especially true for models like ours that are constructed primarily to fit data rather than to make predictions, because their explanatory power rests heavily on qualitative aspects of their representations. One way to categorize knowledge is in terms of generality. In particular, one would expect that complex problem solving involves knowledge ranging from general, because some thoughts and actions recur often, to specific, because circumstances are often unique.

Figure 8 tabulates productions and firing counts according to four categories of generality (arrayed horizontally). The top bar indicates the number of productions in the model at the end of the simulation, including all preloaded productions and all chunks acquired as the model runs. The bottom bar indicates the total number of production firings that occur during the simulation.

The right half of the picture (shaded) shows that most of the model's productions (87%) are acquired by learning, suggesting that learning in the model is in fact pervasive. Soar makes these chunks specific to their encoding context, but some transfer despite this. In particular, chunks encoding information from or about the display account for 8% of firings. Thus the model's behavior depends in part on memories for specific external situations that arise during task performance.

The left half of the picture (unshaded) shows that the model begins with a small number of preloaded productions that account for most of its processing. Preloaded productions number 194 (13%), of which 126 (8%) represent domain knowledge that we attribute to the programmer. This preloaded knowledge lets the model attend to external objects, generate cues, and recall facts about objects. It also tells the model what commands it can issue and what objects are important to comprehend and therefore select as comprehension goals.

A question concerning any representation of expert knowledge is whether it can be used flexibly, in the way we suppose people make flexible use of expertise. In our model,

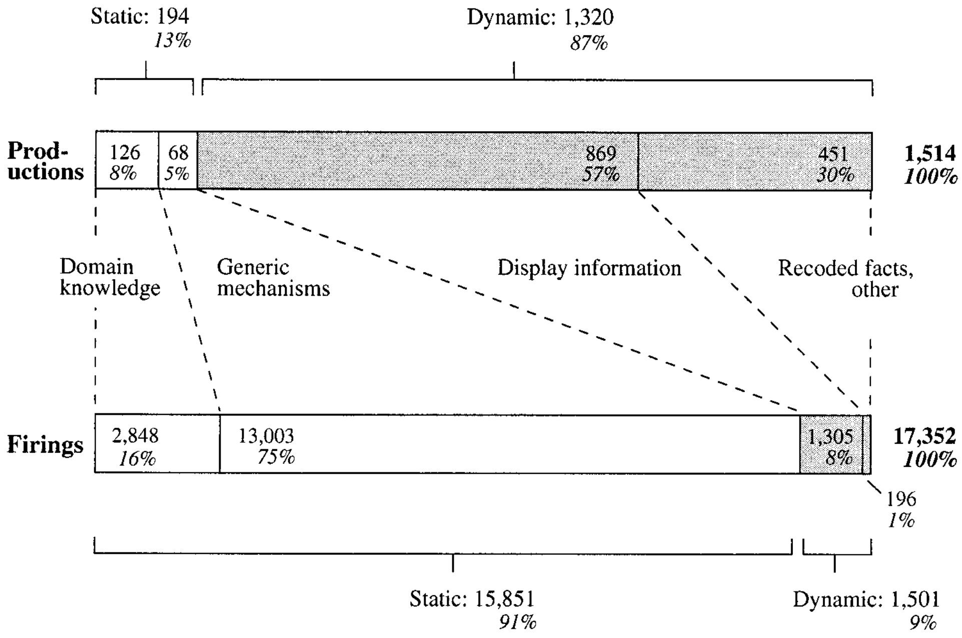


Figure 8. Number of productions and production firings at the end of the simulation. Most productions were acquired as the model ran (the shaded segments of the upper bar). Most firings are by general productions loaded before the run starts (the unshaded segments of the lower bar). Static productions represent domain knowledge that the model brings to the task. Dynamic productions are specific to the context in which they were encoded, which limits the extent to which they transfer.

a large number of static domain-knowledge productions (93 out of the 126 indicated in Figure 8) represent either comprehension goals or attend or probe subgoals. These 93 productions account for all 499 goal and subgoal selections that occur as the model runs, for a mean of 5.4 goals per production. They also account for 2,518 out of the 2,848 firings of domain-knowledge productions. These measures indicate that to a large extent the model's goal and subgoal productions transfer among situations rather than being hard-wired to a particular one.

The category of preloaded productions labeled "generic mechanisms" accounts for 75% of total production firings, despite being only 5% of the total production count. These are domain-independent productions like recall-seeing-object, which infers the existence of a hidden target from an episodic trace. The high firing rate of mechanistic productions is consistent with their being the most general productions in the model and potentially architectural.

The production and firing counts over the lifetime of the simulation illustrate the implications of pervasive episodic learning. In a few minutes of simulated time, the model acquires a great deal of dynamic information about its environment and stores it permanently in LTM. Some of these chunks transfer in the near term, firing seconds to minutes (of simulated time) after being created. The fast learning rate–1320 chunks in 10.5

minutes, or roughly two chunks per second—suggests that Brooks's (1977) estimate of tens or hundreds of thousands of rules making up a programmer's static domain knowledge may account for only part of what generates expert performance. There may in addition be a vast and constantly growing store of simple dynamic knowledge that mediates expert access to external information.

V. GENERAL DISCUSSION: EPISODIC INDEXING IN CONTEXT

The scrolling example discussed in this article is an instance of what we propose as a more general theory of episodic indexing. The theory is that people continually acquire pointers to external objects as a side effect of attention, and that these pointers are retrieved from LTM by a deliberate cue-generation process. In the following sections we relate the theory to previous research on episodic and temporal codes, recognition and recall processes, and source monitoring (Johnson, Hashtroudi, & Lindsay, 1993). We then examine how the assumptions and constraints we have adopted play out as requirements on the representation of episodic memory. Finally, we interpret episodic indexing as an instance of long-term working memory (Ericsson & Kintsch, 1995), finding that it occupies one end of a spectrum of skilled memory.

Temporal and Episodic Codes

The association of episodic information with perceived objects seems to play a role at other points in attentional processing, further out from the center of cognition. For example, feature integration theory (e.g., Treisman, 1993) proposes that an object file is created or updated in the process of attending to an external object. The object file contains episodic information about the object in context, describing the object more broadly or more narrowly as attention is less or more focussed, and keeping track of such dynamic aspects as position. This representation implies that objects are associated with a sense of the present—that there is some way to distinguish current from previous values of attributes like position or orientation. This is converging evidence for the possibility that episodic processing is an integral part of attention.

Integrating episodic encoding with attention implies a pervasive storage of episodic codes at a very fine temporal grain. This is congruent with the obligatory encoding assumption of instance-based memory theory (Logan, 1988). It also agrees with evidence that events are encoded in memory at a fine grain even when such encoding is not logically necessary for task performance (Altmann & Gray, 1998).

In characterizing the temporal coding of perceptual events, Underwood (1977) proposed a recency principle: "Immediately after the perception of an item, temporal information for that item relative to other items coming before it is perfect, but as time passes the information available becomes less and less reliable". Underwood also proposed that temporal codes richer than recency are established only when a known temporal-ordering scheme is intentionally involved at encoding time. The overall picture, including related findings on incidental temporal codes (Hasher & Zacks, 1979), is that recency encoding is automatic, but that more complex temporal encodings are not. Our model reflects this with an automatic awareness of the present and a reliable ability to distinguish present from past. Despite its discrete (and hence somewhat degenerate) nature, this sense of recency is sufficiently powerful to index hidden objects in at least one dynamic task environment.

The episodic index records other events than attention (as we discuss in the upcoming section, Representational Requirements on Episodic Memory), and thus may account for various expert-novice differences in problem solving. Two possibilities are the event of making a decision and the event of making notes. For example, in one study novice and expert software designers were observed as they designed a text-processing system (Jeffries, Turner, Polson, & Atwood, 1981). One novice failed to remember an initial design decision, whereas no experienced designer showed this kind of failure. The novice wrote notes about his initial design decision, but then remade the same decision again from scratch later in the session. Apparently he failed to remember not only the initial decision, but also the existence of his notes. Jeffries et al. characterize this as a failure of episodic memory. It might also be interpreted as a failure of the retrieval structures that point to external memory (Schönpflug & Esser, 1995); had the participant recalled simply the event of writing down the initial decision, he might have retrieved his notes instead of solving the problem again. In either case, episodic indexing suggests that the participant was deficient in domain knowledge for generating cues. A more expert designer might have generated an image at the appropriate time as a reminder of the previous design decision or of the event of writing it down. This is consistent with experts in the study showing no similarly clear failures of episodic memory.

Recognition and Recall

Soar in general and our model in particular predict that recognition knowledge (in the form of episodic chunks) is acquired automatically, whereas recall knowledge is not. This is consistent with the heavy dependence on environmental cues shown even by skilled computer users (Mayes, Draper, McGregor, & Oatley, 1988; Payne, 1991). It also agrees with Larkin's computational theory of display-based problem solving (Larkin, 1989), which derives flexibility and efficiency from productions that directly recognize functional relationships in the environment.

Whereas attention events are encoded automatically, they can be recalled deliberately with controlled processing and the requisite knowledge. This agrees with generate-recognize theories of recall (Anderson & Bower, 1972; Kintsch, 1970; Watkins & Gardiner, 1979), in which recall consists of one process that generates candidates, linked to a second process that recognizes the target. The generator in our model is the imagining process, which places candidate attention events into WM as cues. This generator depends on different kinds of domain knowledge, including knowledge from which to generate images as well as a semantic understanding of when imagery might be productive. The recognizer is the Soar production-firing cycle, which finds matches between imagined events and conditions of episodic chunks. The actions of successfully-matched productions add symbols to WM that represent recall of past events.

Episodic indexing suggests that the bottleneck in gaining access to external information is a function of deficits in knowledge for generating cues. In the context of a learning architecture like Soar, this begs an important question: How do people acquire the ability to generate an image from memory in the first place?

Soar models can learn to generate images by the process of data chunking (Lehman, Laird, & Rosenbloom, 1998; Newell, 1990; Rosenbloom, Laird, & Newell, 1987; Rosenbloom, Newell, & Laird, 1991; Vera, Lewis, & Lerch, 1993). Data chunking is a reconstructive process that creates the target image (or other data to be made recallable) in WM, using pre-existing, typically finer-grain knowledge that is already available in LTM. The target image is then encoded as the result of a new chunk. For example, a model might learn to recall a word by constructing it from letters (Rosenbloom, Newell, & Laird, 1991), and a letter by constructing it from features like lines and curves. This regress ultimately implicates the symbol-grounding problem (Harnad, 1990; Touretzky & Pomerleau, 1994), which is the problem of how to develop a reliable mapping between the symbols of a cognitive representation and the real-valued, physical signals that impinge on the senses. Although the symbol-grounding problem has not been solved, it is likely that, whatever the grounding, Soar will use data-chunking ubiquitously at all levels to build up the symbolic memory structures necessary for problem solving.

In sum, episodic indexing maps to standard conceptions of recognition and recall. Retrieval maps to coupled generate and recognize processes with plausibly asymmetrical levels of difficulty. Encoding maps to the low-cost acquisition of recognition knowledge. No component of the model maps to the acquisition of recall knowledge, but were such a component implemented in Soar it would be as costly as recall itself. This mapping suggests that people know much more about the environment than they are aware of, and that becoming aware of this knowledge requires either the reappearance of the external stimulus (which may then be recognized) or the effort and ability to call an image of the stimulus to mind.

Source Monitoring

Source monitoring is the process of determining the origins of a remembered item or event (Johnson, Hashtroudi, & Lindsay, 1993). For example, suppose one recalls a particular passage of text, but not its source (that is, where it was seen, or whether it was imagined). According to the source monitoring framework, source judgments are sometimes made quickly (or heuristically) based on qualitative aspects (or characteristics) of a memory. For example, a memory that has the characteristics of vividness and dense detail might be judged a product of perception rather than a product of imagination. At other times, source judgments are made more slowly (or systematically) based on reasoning and explicit knowledge.

Heuristic source judgments occur in our model when it generates a probe, if an episodic chunk fires in response to the probe. The probe itself represents activation of a memory, and the episodic chunk firing represents activation of an associated temporal characteristic. The temporal characteristic triggers a heuristic judgment (made by production recall-seeing-object, discussed earlier) about whether the corresponding object exists in the environment. This in turn may trigger a systematic process that scrolls a target back into view. We interpret such an action on the environment as source identification involving external rather than internal memory.

The model's time symbols are qualitatively as basic and lean as the temporal characteristics described by the source monitoring framework. For example, by themselves they do not indicate that an object exists; this must be inferred, perhaps erroneously if the environment has changed. The elementary nature of these symbols is consistent with what we propose is the ubiquity of episodic indexing. Because encoding is automatic, the encoding rate is continuously high (as illustrated in the section, Production and Firing Counts). This limits the amount of information that can be stored in each chunk, given that increasing the amount of declarative information in a chunk entails cognitive opportunity cost (Lehman, Laird, & Rosenbloom, 1998; Newell, 1990). Thus a processing account of why memory characteristics are as lean as they are is that this is the cost of the coverage provided by pervasive encoding.

In sum, our model suggests that people encode lean source information automatically and continuously, hedging their bets with a minimal but constant cognitive investment in episodic chunking. This implies that source identification is possible in principle for any deliberately attended object. It also implies that for many targets, specific source judgements (concerning exact location, for example) will often require systematic processing of external memory, because by default so little context is actually stored with the item.

Representational Requirements on Episodic Memory

The memory representation that implements episodic indexing is constrained both by Soar and by the requirements of reality monitoring (Johnson, Kounios, & Reeder, 1994; Johnson & Raye, 1981), a specific form of source monitoring that determines whether a recollection is a figment of the imagination. In our model, reality monitoring supports the accuracy of heuristic judgments about the existence of an object in the environment. We describe this function below, and argue that the combined constraints of episodic indexing, chunking, and reality monitoring place a lower bound on how lean the model's time symbols can be.

Episodic memory is a natural construct to study in Soar, because chunks have an inherently episodic quality. A chunk firing means that an event occurred in the past (namely the event in which the chunk was learned), and the event's context is captured in the chunk's conditions (Rosenbloom, Newell, & Laird, 1991). A model can determine whether an event occurred by generating the corresponding context in WM, then by monitoring WM for the result of the corresponding chunk. Several Soar models have addressed episodic memory phenomena in these terms (e.g., Huffman, 1994; Lehman, Laird, & Rosenbloom, 1998; Mertz, 1995; Rieman, Young, & Howes, 1996; Rosenbloom, Newell, & Laird, 1991). However, the memory processes supported in our model impose new constraints on what information must be represented in the results of episodic chunks.

An assumption implemented in our model is that it should store probe events in memory as well as attention events. This reflects the general symmetry of probing and attention as information-gathering operations, extended specifically to our assumptions about attention. A probe event, like an attention event, is worth remembering, both because it contributes to comprehension by retrieving knowledge, and because simply knowing that something was imagined sometime in the past could be important.

This symmetry introduces the need to distinguish attention events from probe events at retrieval time. If the model probes repeatedly with the same image, a later probe will trigger episodic chunks from earlier probes. This could lead the model to mistake these earlier probes for attention events, which could in turn lead it to scroll to a non-existent target. To let the model avoid this, episodic memory must contain some source information that allows the discrimination of products of perception from products of imagination.

However, storing source information in chunks can interfere with episodic indexing, which requires transfer of episodic chunks from attention context to probe context. This transfer can occur only if the chunk omits most source information, a constraint that follows from Soar's implementation of encoding specificity. When building a chunk, Soar traces from the result back to premises on which the result logically depends. When source information is a result, it depends directly on the actual source of the target object, making source information a premise as well. Thus if a result identifies an object as real, that chunk can never be triggered by an image, and hence fails to support episodic indexing.

These interfering constraints are satisfied by a representation in which reality monitoring falls partly to memory and partly to retrieval-time processing. Source information is processed as a result in probe events but not in attention events, meaning that source information is stored in chunks only when the source is internal. When the source is external, this fact must be inferred, because it cannot be retrieved. The model makes this inference by examining the episodic chunks triggered at retrieval time. Chunks from past probes are identified by their results and subtracted from the full set of triggered chunks.[11] If any chunks remain, the model infers that they represent attention events and that the target is therefore real. Thus to answer the question, "Did an attention event occur?", the model asks instead, "Did any events occur that were not probes?"

Answering this question requires subtracting one set of triggered chunks from another, which in turn requires well-formed sets of chunks in which each member can be distinguished. However, Soar cannot inspect LTM directly (Newell, 1990), meaning that chunks can be differentiated only by the results they add to WM when they fire. Thus, set operations on chunks, like set subtraction, require that each chunk add a distinct result to WM. This requirement cannot be met by a single symbol (e.g., old-event), because at most one instance of a given symbol is represented in WM at a given time; if multiple chunks fire at the same time and each adds only that symbol to WM, the distinction between them is lost. A dynamic declarative representation is required, in which each new chunk is identified by a new result. This requirement is met by the model's time symbols (as illustrated in the Appendix), because the current time symbol, which is the one encoded with current attention events, is replaced in WM at fine-grain intervals (once per comprehension goal) by a new one guaranteed to be unique.

In sum, episodic indexing and reality monitoring interact with each other and with chunking to require a richer representation of episodic memory in Soar than identified in previous analyses (e.g., Rosenbloom, Newell, & Laird, 1991). Source information encoded in chunks can inhibit transfer between contexts and thereby inhibit episodic indexing. This conflict is overcome by encoding only internal source information, and by supplementing this with declarative episodic codes that support inferences about external source at retrieval time. This representation is consistent with the view of episodic memory as declarative (e.g., Tulving, 1983), and agrees qualitatively with findings that correct source attribution takes longer when source was external as opposed to internal (Johnson, Kounios, & Reeder, 1994). The representation is primitive, for example in that it over-predicts the reliability of reality monitoring. Nonetheless, the manner in which it was shaped by mutually constraining processes and mechanisms is paradigmatic of an integrated and computational approach to the development of cognitive theory (Newell, 1973).

Long-Term Working Memory

We claim that effective access to external information depends in part on semantic, relatively static domain knowledge that one brings to the task. This claim is congruent in key respects with the construct of long-term working memory (Ericsson & Kintsch, 1995), in which long-term knowledge (as opposed to inherent WM capacity; e.g., Just, Carpenter, & Hemphill, 1996) accounts for functionally expanded WM. Episodic indexing and LT-WM both propose that people store information rapidly in LTM, and use well-learned semantic knowledge to gain access to this information when it becomes relevant.

Below we examine the similarities and differences between the two proposals, and argue that the behavior we modeled marks the lean and ubiquitous end of a continuum of skilled memory. We organize this comparison around the three top-level components of LT-WM: (1) the ability to encode traces in LTM online (in real time), in such a way as to (2) allow rapid and flexible retrieval, without (3) interference effects from associating too many elements with the same cue too quickly.

Online Encoding. Ericsson and Kintsch cite incidental memory as evidence of online encoding of information. In laboratory tests of incidental memory, the experimenter unexpectedly asks the participant to recall information related to task performance, once task performance is complete. If the participant can recall information after it has fallen out of WM, without knowing ahead of time that this information would have to be recalled, then the information must have been stored in LTM online, as part of the participant's regular cognitive activity during task performance. In our data, evidence for online encoding comes from scrolling incidents that implicate LTM. This online encoding resembles incidental encoding in that the programmer was not explicitly asked to remember particular objects, and although she might have anticipated the need to scroll back to certain objects and thus made a special effort to remember them when they first appeared, this is not evident from her verbal protocol. Our working hypothesis is that the programmer could have recalled attending to any number of objects, not just the targets that she did scroll to.

Retrieval Structures. In LT-WM, rapid and flexible retrieval is made possible by retrieval structures. These are LTM structures that maintain close and reliable associations between easily-generated cues and important dynamic information. For example, chess experts can encode meaningful board configurations in a retrieval structure that links squares to their contents. A more common example is the situation model constructed in the mind of the skilled reader comprehending text (Ericsson & Kintsch, 1995). The situation model represents the objects and events described in a text (van Dijk & Kintsch, 1983). One of its primary ingredients is domain knowledge, which integrates and relates objects and events as they arise during comprehension. The situation model allows the comprehender to maintain ready access to information necessary for such processes as referent resolution.

We found evidence of retrieval structures similar to those implicated in text comprehension. In the scrolling incident we have described, the programmer is comprehending a u-model and recalls the existence of a hidden copy, rapidly and reliably enough that she acts on this recollection with confidence in the existence of the target. This suggests an efficient retrieval structure organized around objects that are salient in the domain over which the program computes. (The u-model, or utterance model, is an object in the domain of language comprehension.)

Retrieval structures organized around semantic nodes are consistent with comprehension studies involving external memory, as well as with comprehension studies in programming. In one study, text comprehenders were allowed to externalize some (but only some) information in a paragraph at their discretion, in preparation for a comprehension test (Schönpflug & Esser, 1995). When the paragraph was organized hierarchically, participants memorized the higher-level propositions, externalized lower-level propositions, and associated the former with addresses of the latter. Thus, given a choice, participants remembered propositions as a function of their semantics and used those propositions to index the environment. There is also evidence that as programmers become more familiar with a program, their mental representation shifts from a control-flow representation to a situation model involving the objects that the program is about (Pennington, 1987b). When participants were asked to modify the programs they were studying rather than simply study them, accuracy declined on questions about control flow but increased on questions about data structures representing domain objects. In sum, evidence from diverse sources is consistent with the proposal that experts organize retrieval structures around domain knowledge and use these structures to gain access to working information stored externally.

However, there is an important distinction between the retrieval structures of text comprehension and the episodic index. This concerns the degree of semantic integration achieved online. As proposed in theories (e.g., Gernsbacher, 1990; van Dijk & Kintsch, 1983) and as made explicit in simulations (e.g., Kintsch, 1998; Lewis, 1993), comprehenders create densely integrated memory structures as they process text, allowing thematic and other inferences based on meaning. Thus in comprehension it is semantic information that is being integrated online. This implies, for example, that a difference between skilled and unskilled readers is the degree to which semantic structures can be constructed quickly and completely (Ericsson & Kintsch, 1995). Although our programmer was clearly engaged in comprehension, and was presumably building the appropriate memory structures, we chose not to address this phenomenon. The model only elaborates goals, rather than building them into organized structures. The main product of its online structure-building is the episodic index, which is not marked by any particular degree of semantic integration. Episodic chunks are not cross referenced, and their content consists of lean temporal codes rather than pointers to related propositions or objects. Moreover, episodic chunks are only functional in combination with spatial and navigation knowledge indicating where and how to look for the target object. (As we have outlined, this spatial and navigation knowledge is simplified in our model, but more complete representations appear to be compatible with the mechanisms of episodic indexing; Altmann, 1996.) In sum, whereas comprehension constructs a semantic network with links across many dimensions, episodic indexing constructs only a one way mapping from semantic to episodic codes.

Interference-Resistant Encoding. Keeping events distinct in memory requires interference-resistant encoding. However, the need for such encoding varies with the task and with the item to be stored (Ericsson & Kintsch, 1995). For example, mental-calculation experts need to maintain access to intermediate products, but have poor memory for all but the most recent one. Older ones are typically unnecessary, so there is no need to prevent them from interfering with each other. Thus interference-resistant encoding varies adaptively.

From this point of view, interference resistance might be unnecessary in task environments where knowledge is recorded externally. If potentially-interfering objects can be distinguished with reference to the environment, then it may not pay to invest the extra cognitive effort in distinguishing them internally as well. Our programmer's task environment may be one example. For instance, the u-model changes over time, and the most recent copy is most likely to be current. Thus there is no need to keep different instances distinct in memory; scrolling can stop with the first instance encountered.[12]

A more general example is the way people use search commands (as opposed to scrolling commands) in a text editor or browsing environment. When a search turns up a hit, the searcher will often evaluate the context of the hit to see if the hit is the desired one (e.g., Peck & John, 1992). This method is supported by many interfaces (ranging from Emacs to the Macintosh) with special short-cut methods for finding the next instance of a search string. Again, it may not matter if the agent internally confuses multiple copies of an object, because the external environment allows for disambiguation.

Thus in common tasks involving external memory, the external record allows the problem solver in effect to release interference between multiple copies of an object. In such tasks it may be appropriate to implicate LT-WM without requiring evidence of interference-resistant internal encodings.

A Spectrum of Skilled Memory. To summarize, we find evidence in our data of pervasive online encoding, a process that Soar's learning mechanism supports naturally. We also find evidence of domain knowledge providing access to simple existential information about objects in the world. Finally, because these objects are distinct externally, there is less need to encode them distinctly in memory. Thus episodic indexing has the form of LT-WM, but it may be more common than the forms analyzed by Ericsson and Kintsch (1995).

The spectrum of domains in which LT-WM is a factor is marked by differences in the extent to which memory skill is an end in itself. At one extreme is the exceptional digit span (Chase & Ericsson, 1981; Ericsson & Staszewski, 1989), which requires extensive training tied specifically to the memory task. More common are domains like medical diagnosis and chess, where exceptional memory is still a product of training, but where it is not the primary measure of performance. More common still is text comprehension, which is practiced by most adults, but this continues to implicate densely-connected semantic structures encoded online. In episodic indexing, online encoding is reduced to constructing a low-cost, sparsely-connected mapping from semantic to episodic codes, a structure sufficient for remembering and retrieving external objects.

Having identified a spectrum of skilled memory, we can ask what is common about its various manifestations and what this commonality tells us about underlying mechanisms. All the manifestations described above feature a massive volume of small units of symbolic knowledge, acquired and organized with substantial effort over time and accessed by means of specialized cues. We have focused primarily on the acquisition and access processes, but Soar has been used to address knowledge organization and restructuring (Miller & Laird, 1996; Newell, 1990). Also, as we argued in our discussion of recognition and recall, Soar predicts which memory processes will be easy and which will be hard for a wide variety of tasks. Finally, because it couples learning and problem solving (Lehman, Laird, & Rosenbloom, 1998; Newell, 1990), Soar should be able to explain the problem solving processes integral to skilled memory phenomena. Thus Soar appears to be a viable starting point from which to undertake a unified theory of skilled memory.

VI. CONCLUSIONS

We propose that people store large amounts of simple information in LTM as they pay attention to what they are doing. This information represents a database of attention events and serves as a mental index in which objects can be looked up to see if they exist in the environment. The index is constructed automatically and used deliberately, with its usefulness depending on the quality of the domain knowledge available for generating cues during the lookup process.

More specifically, episodic indexing posits:

1. Automatic and pervasive encoding. People acquire large amounts of episodic information about attention events, as a side effect of attention.
2. Controlled, knowledge-based retrieval. People gain access to this information by generating cues from memory. Each cue represents an attention event, and the result of access is an indication that the event occurred. The effectiveness of retrieval depends

on domain knowledge, including familiarity with domain objects (the source of cues) and an understanding of their relevance to the task at hand (the source of guidance for when to generate them).

These processes are shaped by the dual constraints of empirical data and cognitive architecture. They are implemented in a computational cognitive model, demonstrating sufficiency in one domain, but derive generality from the domain independence of the underlying generic and architectural mechanisms. We thus offer episodic indexing as a general hypothesis about how people maintain access to large and dynamic information spaces.

Acknowledgements: We would like to thank Walter Kintsch and Paul S. Rosenbloom for their insightful reviews of this article, and Wayne D. Gray, Jill H. Larkin, Jill Fain Lehman, Carol L. Raye, Peter G. Polson, Robert S. Rist, Frank E. Ritter, Wendy A. Rogers, Marilyn C. Salzman, and Richard M. Young for their comments on this work. The first author is particularly grateful to John R. Anderson, Bonnie E. John, Clayton H. Lewis, and James H. Morris in their capacity as his dissertation committee (Bonnie E. John, chair).

This work was supported in part by a Krasnow Postdoctoral Fellowship, in part by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and ARPA under grant number F33615-93-1-1330, in part by the Office of Naval Research, Cognitive Science Program, Contract Number N00014-89-J-1975N158, and in part by the Advanced Research Projects Agency, DoD, monitored by the Office of Naval Research under contract N00014-93-1-0934. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of Wright Laboratory, the Advanced Research Projects Agency, the Office of Naval Research, or the U. S. Government.

APPENDIX

Here we present a complete picture of the processing that occurs when the model probes with an image and retrieves episodic chunks. (This elaborates on the process described in the section, Using the Episodic Index.) In the general case, the episodic chunks retrieved by a probe will be of two kinds: those encoded during attention events and those encoded during past probe events. Only those acquired during past probe events will contain source information in their actions (as discussed in the section, Representational Requirements on Episodic Memory). To determine whether an object of interest was actually attended in the past, the model computes the difference between the total set of episodic chunks triggered by the current probe event and those representing past probe events. If the difference is non-empty, the model infers that at least one past event was an attention event.

The scenario below supposes that the model first probes for information about the SSN field, then actually attends to the field, then probes again. The first probe occurs at time t42, when the model places an image in WM (A1) together with source information identifying the image as such (A2).

```
production: imagine-ssn
(Conditions testing that it's relevant to know that an SSN field was seen)
 →
 (^attended-to ssn)              A1
 (^imagined ssn)                 A2
```

An episodic chunk is encoded during this probe event, under the assumption of pervasive episodic encoding. The source information described above is included as a chunk action (A2) and hence also as a chunk condition (C2).

```
chunk: imagined-ssn              Chunk capturing a probe event.
 (^attended-to ssn)              C1
 (^imagined ssn)                 C2
 →
 (^event ssn ^time t42)          A1
 (^probe t42)                    A2
```

At time t43, the model actually attends to the SSN object, resulting in another episodic chunk.

```
chunk: attended-ssn              Chunk capturing an attention event.
 (^attended-to ssn)
 →
 (^event ssn ^time t43)
```

Finally, at time t44, the model probes a second time (by firing imagine-ssn). This triggers the two episodic chunks described above, causing the following elements to enter WM.

```
(^attended-to ssn)              From imagine-ssn.
(^imagined ssn)                 From imagine-ssn.
(^event ssn ^time t42)          From imagined-ssn.
(^probe t42)                    From imagined-ssn.
(^event ssn ^time t43)          From attended-ssn.
```

From these elements the model can infer that an SSN field exists in the environment. The production that makes this inference is recall-seeing-object, below. Condition C5 (not reported in the section, Using the Episodic Index) effectively subtracts the set of probe events (containing t42) from the set of probe plus attention events (containing t42 and t43). The leading minus sign ("−") negates the subsequent condition, meaning that WM cannot contain an element matching that condition. In our scenario, this negated condition holds for at least one past event (t43). Thus the production matches, inferring that SSN was attended at some point in the past (A1).

```
production: recall-seeing-object
 (^attended-to <o>)            C1, <o> = ssn
 (^imagined <o>)               C2, <o> = ssn
 (^event <o> ^time <then>)     C3, <o> = ssn, <then> = t43
 (^time <now> ≠ <then>)        C4, <now> = t44, <then> = t43
−(^probe <then>)               C5, <then> = t43
 →
 (^recall-seeing <o>)          A1, <o> = ssn
```

NOTES

1. The threshold of 30 seconds for implicating LTM agrees with the duration identified by Card, Moran, and Newell (1983) as the length of a unit task. It is also the span of interrupt used to test the contents of readers' LTM in text comprehension studies (reported in Ericsson & Kintsch, 1995).
2. This example, involving student records and Social Security Numbers, is not from the programmer's domain, but makes the model easier to introduce. Later we present a simulation of actual data.
3. In the context of Soar, use of the term chunk reflects the architectural assumption that all learning, and hence all chunking, occurs through the acquisition of new productions. In the context of the ACT-R theory, which also has architectural learning mechanisms, the same term has recently been adopted to refer to declarative structures (e.g., Anderson, Matessa, & Lebiere, 1998). In each other's terms, an ACT-R chunk is roughly a Soar WM element, and a Soar chunk is either a composed production or a proceduralized declarative structure (Anderson, 1983).
4. The model encodes every attention event, regardless of whether the target was attended previously. Thus if the model attends to the SSN field multiple times, it will acquire multiple chunks, with each chunk mapping the SSN field to a different time symbol. This is consistent with theories of memory in which repeated exposure to a stimulus strengthens memory for that item (e.g., Anderson, 1990; Logan, 1988). However, unlike such theories, our model makes no statements about retrieval latencies and retrieval order. The model simply fires all episodic chunks for a given object in parallel when the appropriate image enters WM. The model also lacks any way to discriminate among different time symbols for the same attended object. The absence of such discrimination mechanisms reflects the scrolling incidents that we modeled. In each incident, the target is always the only hidden copy of that structure, meaning there is no need to distinguish between hidden copies.
5. The production is simplified here for expository purposes. The complete production appears in the Appendix and is discussed in the section, Representational Requirements on Episodic Memory.
6. Angle brackets around a letter (e.g., "<o>") indicate a variable. If the same variable occurs in multiple conditions, it must have the same value in each condition for the production to fire. Thus, for example, C1 and C2 test that the object bound to <o> is both attended-to and imagined.
7. These two elements are actually linked by a relation indicating that the missing attribute was a referent. For brevity, and because this relation does not come into play again, we have omitted it from the representation used here.
8. The cues are the s-construct symbol above s15 and the exhausted symbol below. In terms of the program's execution, these cues imply that certain operations on the u-model are pending.
9. The model contains no representation for the prompt at the bottom of the buffer, so it skips the step of returning to the prompt. This is a simplification that we believe does not materially affect the model's hypotheses (Altmann, 1996). The display emulator is programmed to compensate for the model's lack of a command-prompt representation.
10. The display emulator interprets these abstract commands in accordance with the sequence of actual commands issued by the programmer. That is, it automatically scrolls in the direction that the programmer scrolled.
11. This set subtraction operation is implemented in production recall-seeing-object, as we discuss in the Appendix.

12. Our data are neutral with respect to this claim. In all the scrolling incidents we studied, the target object is the only instance of that object that arose in the session, so there is no opportunity to look for interference among multiple instances.

REFERENCES

Altmann, E. M. (1996). *Episodic Memory for External Information*. Doctoral dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh.

Altmann, E. M. & Gray, W. D. (1998). *Pervasive episodic memory: Evidence from a control-of-attention paradigm*. Proceedings of the Twentieth Annual Conference of the Cognitive Science Society (pp. 42–47). Hillsdale NJ: Lawrence Erlbaum.

Altmann, E. M. & John, B. E. (1995). *A Preliminary Model of Expert Programming.* Tech. rep. CMU-CS-95-172, School of Computer Science, Carnegie Mellon University, Pittsburgh.

Altmann, E. M. & Yost, G. R. (1992). *Expert-System Development in Soar: A tutorial*. Tech. rep. CMU-CS-92-151, School of Computer Science, Carnegie Mellon University, Pittsburgh.

Altmann, E. M., Larkin, J. H., & John, B. E. (1995). *Display navigation by an expert programmer: A preliminary model of memory*. Human Factors in Computing Systems: CHI 95 Conference Proceedings (pp. 3–10). New York: ACM Press.

Anderson, J. R. (1983). *The Architecture of Cognition.* Cambridge, MA: Harvard University Press.

Anderson, J. R. (1990). *The Adaptive Character of Thought*. Hillsdale NJ: Lawrence Erlbaum.

Anderson, J. R. & Bower, G. H. (1972). Recognition and retrieval processes in free recall. *Psychological Review, 79*, 97–123.

Anderson, J. R., Matessa, M., & Lebiere, C. (1998). ACT-R: A theory of higher-level cognition and its relation to visual attention. *Human-Computer Interaction, 12*, 439–462.

Andrade, J. & Meudell, P. (1993). Is spatial information encoded automatically in memory? *Quarterly Journal of Experimental Psychology, 46A*, 365–375.

Bauer, M. & John, B. E. (1995). *Modeling time-constrained learning in a highly interactive task.* Human Factors in Computing Systems: CHI 95 Conference Proceedings (pp. 19–26). New York: ACM Press.

Boehm-Davis, D. A, Fox, J. E., & Philips, B. H. (1996). *Techniques for exploring program comprehension.* Empirical Studies of Programmers: 6th workshop (pp. 3–37). Norwood NJ: Ablex.

Bower, G. H. (1975). Cognitive psychology: An introduction. In W. Estes (Ed.), *Handbook of Learning and Cognitive Processes* (Vol. 1). Hillsdale NJ: Lawrence Erlbaum.

Brandimonte, M., Einstein, G. O., & McDaniel, M. A., Eds. (1996). *Prospective Memory: Theory and applications*. Hillsdale NJ: Lawrence Erlbaum.

Brooks, R. E. (1977). Towards a theory of the cognitive processes in computer programming. International *Journal of Man-Machine Studies, 9*, 737–751.

Brooks, R. E. (1983). Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies, 18*, 543–554.

Card, S. K., Moran, T. P., & Newell, A. (1983). *The Psychology of Human-Computer Interaction.* Hillsdale NJ: Lawrence Erlbaum.

Chase, W. G. & Ericsson, K. A. (1981). Skilled memory. In J. R. Anderson (Ed.), *Cognitive Skills and Their Acquisition.* Hillsdale NJ: Lawrence Erlbaum.

Davies, S. P. (1994). Knowledge restructuring and the acquisition of programming expertise. *International Journal of Human-Computer Studies, 40*, 703–726.

Detienne, F. & Soloway, E. (1990). An empirically-driven control structure for the process of program understanding. *International Journal of Man-Machine Studies, 33*, 323–342.

Ericsson, K. A. & Kintsch, W. (1995). Long-term working memory. *Psychological Review, 102*, 211–245.

Ericsson, K. A. & Simon, H. A. (1992). Protocol Analysis: *Verbal reports as data*. Cambridge, MA: MIT Press.

Ericsson, K. A. & Staszewski, J. J. (1989). Skilled memory and expertise: Mechanisms of exceptional performance. In D. Klahr & K. Kotovsky (Eds.), *Complex Information Processing: The impact of Herbert A. Simon*. Hillsdale NJ: Lawrence Erlbaum.

Gernsbacher, M. A. (1990). *Language Comprehension as Structure Building.* Hillsdale NJ: Lawrence Erlbaum.

Glasgow, J. & Papadias, D. (1992). Computational imagery. *Cognitive Science, 16*, 355–394.

Golledge, R. G. (1991). Cognition of physical and built environments. In T. Garling & G. W. Evans (Eds.), *Environment, Cognition, and Action*. New York: Oxford University Press.

Gray, W. D. & Anderson, J. R. (1987). *Change-episodes in coding: When and how do programmers change their code?* Empirical Studies of Programmers: 2nd workshop (pp. 185–197). Norwood NJ: Ablex.

Green, T. R. G., Bellamy, R. K. E., & Parker, J. M. (1987). *Parsing and gnisrap: A model of device use.* Empirical Studies of Programmers: 2nd workshop (pp. 132–146). Norwood NJ: Ablex.

Harnad, S. (1990). The symbol grounding problem. *Physica D, 42*, 335–346.

Hasher, L. & Zacks, R. T (1979). Automatic and effortful processes in memory. *Journal of Experimental Psychology: General, 108,* 356–388.

Holt, R. W., Boehm-Davis, D. A., & Schultz, A. C. (1987). *Mental representations of programs for student and professional programmers.* Empirical Studies of Programmers: 2nd workshop (pp. 33–46). Norwood NJ: Ablex.

Howes, A. (1994). *A model of the acquisition of menu knowledge by exploration.* Human Factors in Computing Systems: CHI 94 Conference Proceedings (pp. 445–451). New York: ACM Press.

Howes, A. & Young, R. M. (1996). Learning consistent, interactive and meaningful task-action mappings: A computational model. *Cognitive Science, 20*, 301–356.

Howes, A. & Young, R. M. (1998). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human-Computer Interaction, 12*, 311–343.

Huffman, S. C. (1994). *Instructable Autonomous Agents.* Doctoral dissertation, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor.

Humphreys, G. W. & Bruce, V. (1989). *Visual Cognition: Computational, experimental, and neuropsychological perspectives.* Hillsdale NJ: Lawrence Erlbaum.

Jeffries, R., Turner, A. A., Polson, P. G., & Atwood, M. E. (1981). The processes involved in designing software. In J. R. Anderson (Ed.), *Cognitive Skills and Their Acquisition.* Hillsdale NJ: Lawrence Erlbaum.

John, B. E. & Kieras, D. E. (1996). The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Transactions on Computer-Human Interaction, 3,* 320–351.

Johnson, M. K. & Raye, C. L. (1981). Reality monitoring. *Psychological Review, 88,* 67–85.

Johnson, M. K., Hashtroudi, S., & Lindsay, D. S. (1993). Source monitoring. *Psychological Bulletin, 114*, 3–28.

Johnson, M. K., Kounios, J., & Reeder, J. A. (1994). Time-course studies of reality monitoring and recognition. *Journal of Experimental Psychology: Learning, memory and cognition, 20,* 1409–1419.

Kintsch, W. (1970). Models for free recall and recognition. In D. A. Norman (Ed.), *Models of Human Memory.* New York: Academic Press.

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

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

Kosslyn, S. M. (1981). The medium and the message in mental imagery: A theory. *Psychological Review, 88*, 46–66.

Kuipers, B. (1978). Modeling spatial knowledge. *Cognitive Science, 2*, 129–153.

Laird, J. E., Congdon, C. B., Altmann, E. M., & Doorenbos, R. (1993). *Soar User's Manual*, Version 6, Edition 1. http://www.isi.edu/soar/users-manual/html/soar6-manual.info.Top.html.

Laird, J. E. Rosenbloom, P. S., & Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning, 1*, 11–46.

Lansdale, M. W. (1991). 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: Lawrence Erlbaum.

Lehman, J. F., Laird, J. E., & Rosenbloom, P. S. (1998). A gentle introduction to Soar: An architecture for human cognition. In D. Scarborough & S. Sternberg (Eds.), *An Invitation to Cognitive Science*, Volume 4: Methods, models, and conceptual issues. New York: MIT Press.

Lewis, R. L. (1993). *An Architecturally-Based Theory of Human Sentence Comprehension*. Doctoral dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh.

Logan, G. D. (1988). Toward an instance theory of automatization. *Psychological Review, 95,* 492–527.

Logan, G. D. (1996). The CODE theory of visual attention: An integration of space-based and object-based attention. *Psychological Review, 103*, 603–649.

Lohse, G. E. (1993). A cognitive model for understanding graphical perception. *Human-Computer Interaction, 8*, 353–388.

Lovelace, E. A. & Southall, S. D. (1983). Memory for words in prose and their locations on the page. *Memory & Cognition, 11*, 429–434.

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

Mayes, J. T., Draper, S. W., McGregor, A. M., & 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.

Mertz, J. S. Jr. (1995). *Using a Cognitive Architecture to Design Instructions.* Doctoral dissertation, Department of Engineering and Public Policy, Carnegie Mellon University.

Miller, C. S. & Laird, J. E. (1996). Accounting for graded performance within a discrete search framework. *Cognitive Science, 20*, 499–537.

Neisser, U. (1976). *Cognition and Reality: Principles and implications of cognitive psychology*. San Francisco: W. H. Freeman and Co.

Newell, A. (1973). You can't play 20 questions with nature and win: Projective comments on the papers of this symposium. In W. G. Chase (Ed.), *Visual Information Processing.* New York: Academic Press.

Newell, A. (1990). *Unified Theories of Cognition*. Cambridge: Harvard University Press.

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

Nygren, E., Lind, M., Johnson, M., & Sandblad, B. (1992). *The art of the obvious.* Human Factors in Computing Systems: CHI 92 Conference Proceedings (pp. 235–239). New York: ACM Press.

Patalano, A. L. & Seifert, C. M. (1997). Opportunistic planning: Being reminded of pending goals. *Cognitive Psychology, 34*, 1–36.

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

Pearson, D. J. (1996). *Learning Procedural Planning Knowledge in Complex Environments*. Doctoral dissertation, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor.

Peck, V. A. & John, B. E. (1992). *Browser-Soar: A computational model of a highly interactive task.* Human Factors in Computing Systems: CHI 92 Conference Proceedings. New York: ACM Press.

Pennington, N. (1987a). *Comprehension strategies in programming.* Empirical Studies of Programmers: 2nd workshop (pp. 100–113). Norwood NJ: Ablex.

Pennington, N. (1987b). Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive Psychology, 19*, 295–341.

Petre, M. & Blackwell, A. F. (1997). *A glimpse of expert programmers' mental imagery*. Empirical Studies of Programmers: 7th workshop. New York: ACM Press.

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

Pylyshyn, Z. W. (1989). The role of location indexes in spatial perception: A sketch of the FINST spatial-index model. *Cognition, 32*, 65–97.

Rieman, J., Young, R. M., & Howes, A. (1996). A dual-space model of iteratively deepening exploratory learning. *International Journal of Human-Computer Studies, 44*, 743–775.

Rist, R. S. (1995). Program structure and design. *Cognitive Science, 19*, 507–562.

Ritter, F. E. & Bibby, P. A. (1997). *Modelling learning as it happens in a diagrammatic reasoning task.* Tech. rep. 45, Department of Psychology, University of Nottingham, UK.

Ritter, F. E. & Larkin, J. H. (1994). Using process models to summarize sequences of human actions. *Human-Computer Interaction, 9*, 345–383.

Rosenbloom, P. S., Laird, J. E., & Newell, A. (1987). *Knowledge level learning in Soar*. Proceedings of the Sixth National Conference on Artificial Intelligence (pp. 499–504). Menlo Park, CA: AAAI.

Rosenbloom, P. S., Laird, J. E., & Newell, A., Eds. (1992). *The Soar Papers: Research on integrated intelligence.* Cambridge: MIT Press.

Rosenbloom, P. S., Newell, A., & Laird, J. E. (1991). Towards the Knowledge Level in Soar: The role of the architecture in the use of knowledge. In K. VanLehn (Ed.), *Architectures for Intelligence*. Hillsdale NJ: Lawrence Erlbaum.

Sanderson, P., Scott, J., Johnston, T., Mainzer, J., Watanabe, L., & James, J. (1994). MacSHAPA and the enterprise of exploratory sequential data analysis. *International Journal of Human-Computer Studies, 41*, 633–681.

Schönpflug, W. & Esser, K. B. (1995). Memory and its Graeculi: Metamemory and control in extended memory systems. In C. A. Weaver III, S. Mannes, & C. R. Fletcher (Eds.), *Discourse Comprehension: Essays in honor of Walter Kintsch*. Hillsdale NJ: Lawrence Erlbaum.

Thorndyke, P. W. (1981). Spatial cognition and reasoning. In J. H. Harvey (Eds.), *Cognition, Social Behavior, and the Environment.* Hillsdale, NJ: Erlbaum.

Touretzky, D. S. & Pomerleau, D. A. (1994). Reconstructing physical symbol systems. *Cognitive Science, 18*, 345–353.

Treisman, A. (1993). The perception of features and objects. In A. Baddeley & L. Weiskrantz (Eds.), *Attention: Selection, awareness, and control—A tribute to Donald Broadbent*. New York: Oxford University Press.

Tulving, E. (1983). *Elements of Episodic Memory.* New York: Oxford University Press.

Underwood, B. J. (1977).*Temporal Codes for Memories: Issues and problems.* Hillsdale NJ: Lawrence Erlbaum.

van Dijk, T. & Kintsch, W. (1983). *Strategies of Discourse Comprehension*. New York: Academic Press.

van Someren, M. W., Barnard, Y. F., & Sandberg, J. A. C. (1994). *The Think Aloud Method.* New York: Academic Press.

Vera, A. H., Lewis, R. L., & Lerch, F. J. (1993). *Situated decision-making and recognition-based learning: Applying symbolic theories to interactive tasks.* Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society (pp. 84–95). Hillsdale NJ: Lawrence Erlbaum.

Von Mayrhauser, A. & Vans, A. M. (1995). Program comprehension during software maintenance and evolution. *Computer, 28,* 44–55.

Watkins, M. J. & Gardiner, J. M. (1979). An appreciation of generate-recognize theory of recall. *Journal of Verbal Learning and Verbal Behavior, 18*, 687–704.

Wolfe, J. M. (1994). Guided Search 2.0: A revised model of visual search. *Psychonomic Bulletin and Review, 1*, 202–238.