A Preliminary Model of Expert Programming

Erik M. Altmann and Bonnie E. John

July 1995
CMU-CS-95-172

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Also appears as Human-Computer Interaction Institute Technical Report
CMU-HCII-95-103

Abstract

Expert programming involves the manipulation of large amounts of memory, including external memory represented on a display, semantic memory for expert knowledge, and episodic memory about what has transpired during a particular programming session. We present a computational cognitive model that emulates several minutes of expert, naturalistic programming behavior. The model has three high-level components: knowledge, an underlying cognitive architecture, and mechanisms that allow the architecture to manipulate the knowledge. To illustrate these components we trace the model on two related episodes of behavior. In the first episode, the programmer acquires information from the display. Then, in the second episode, she recalls something and scrolls through previous screens until she reaches the display from the first episode. Emulating this behavior requires the interaction of external, expert, and episodic knowledge, the encoding of episodic knowledge as carried out by the architecture, and mechanisms for selecting goals, attending to the display, and probing long-term memory.

This work was supported 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.

Keywords: Psychology of programming, computational modeling, memory, display navigation, Soar.

1. Introduction

Expert programmers in natural task environments manipulate vast amounts of information. This information exists externally, represented on the display, and internally, in the long-term memory of the expert. This internal information includes knowledge about selected elements of external information. This knowledge is what expert programmers encode and then activate in order to navigate through large displays of information.

The episodic nature of this kind of knowledge was discussed by Jeffries *et al* (1981), in a study of software design. The authors identified the role of knowledge about what has transpired during a particular design session in allowing designers to return to previous states in their problem solving. More recently, Ericsson and Kintsch (1995) have presented an extensive treatment of a phenomenon they call *long-term working memory*. This is long-term memory that is rapidly and robustly encoded during skilled behavior and resembles short-term memory in how it informs problem-solving. Ericsson and Kintsch cite empirical evidence for long-term working memory in complex tasks including chess, a skilled waiter remembering orders, and medical diagnosis. We see this phenomenon as central to the performance of expert programmers.

We describe a preliminary computational model of expert programming that integrates external memory with internal expert (semantic) and episodic memory. Our model is based on observations of an experienced research programmer working on a large system in her own programming environment. The issue of episodic memory arises because the programmer often navigates through the output of her program to regions generated earlier during the session.

Below we describe the task environment and the programming behavior on which we trace our model's behavior. In Section 2 we give an overview of the high-level components of the model: the knowledge represented in it, the underlying architecture, and the mechanisms that allow the architecture to manipulate the knowledge. In Section 3 we trace the model's behavior on two episodes. In the first episode, the main phenomenon of interest is the encoding of episodic knowledge about a feature of the display. In the second episode, we show how the model manipulates the display, selects goals, attends to the display, and searches its memory. This problem-solving activity eventually parlays a cue on the display into a recollection of the episodic memory encoded in the first episode. This recollection is the basis for a decision to scroll in search of the display feature responsible for the memory.

1.1. Task environment and data

The programming session we observed was part of a long-term project to create a natural-language comprehension program in a production language. The task the programmer set herself for this session was to comprehend and modify a part of this program originally written by a colleague.

The programmer displayed and edited code files, and ran her program interpretively in a GNU Emacs process buffer. During most of the session she stepped through the program slowly, issuing commands to display program state. The interpreter output, including long traces, appeared at the bottom of the process buffer, with Emacs automatically scrolling the buffer when more room was needed to display the output.

We recorded a think-aloud protocol, and captured the display and the programmer's gestures on video. We instrumented Emacs to record a timed keystroke protocol, the contents of the process buffer, and code files examined and edited.

1.2. Modeled behavior

Our model currently emulates the programmer's behavior over seven continuous minutes of the 80-minute session. Here we focus on two separate but related episodes within this seven minutes, introduced in Figure 1. The display areas examined by the programmer are on the left, her utterances are on the right, and the time course for the utterances runs down the center. The task-specific references in the utterances will be defined as needed.

```
                 display                   time    utterances
                                           (sec)

(sp chunk-128                              t17  i don't know
 :chunk                                    t18  why it's testing for the
 (goal <g1> ^operator <o1>                 t19
            ^state <s1>)                   t20
 (<o1> ^name s-constructor16               t21  operator?
 ...

episode E1
____________________________________________________________
episode E2
(6 minutes, 3 screenfuls later)            t388 eeuuu
                                           t389
                                           t390
                                           t391 rats
                                           t392
                                           t393
                                           t394
                                           t395
                                           t396
                                           t397 i think all those chunks
                                           t398 i built
                         starts scrolling  t399
                                           t400 test for operator
                                           t401
                                           t402
                                           t403 six
                                           t404 uh
                                           t405 ess  whatever it is
                                           t406 i think
                          stops scrolling  t407 they test for the
                                           t408 s-constructor
E1 screen redisplayed:                     t409
                                           t410 let's
(sp chunk-128                              t411 see shall we
 :chunk                                    t412 yeah
 (goal <g1> ^operator <o1>                 t413 s-constructor16
            ^state <s1>)                   t414 there it is
 (<o1> ^name s-constructor16
 ...
```

Figure 1: Sample behavior involving episodic memory.

In E1, the programmer is looking at a *chunk*, or production. She expresses a lack of knowledge about its *tests*, or conditions (t18). She examines them, then notes an *operator test* (t21). Later, in E2, she recalls something relating to an operator test (t400). This prompts her to scroll back to the screen from E1.

The two episodes are separated by many intervening tasks, including 28 commands to the production-system interpreter and 2 other unrelated scrolling sequences. The time elapsed between the episodes is 6 minutes. The output from the intervening tasks adds 3 new pages to the bottom of the process buffer. Thus there is enough separation, in terms of tasks, time, and display changes, to indicate that the programmer's knowledge of the E1 display is encoded in her long-term memory. In the 80-minute session there were 17 instances where the target of a search through hidden regions of the display had been hidden for at least a minute (Altmann *et al*, 1995). This suggests that the programmer commonly relied on long-term memory to navigate.

The model we describe next offers one explanation of how episodic long-term memory is encoded, what it represents, and how it can be activated.

2. A Computational Cognitive Model

Our model has three components that interact to account for the behavior in the example above: knowledge, an underlying cognitive architecture, and mechanisms that allow the architecture to manipulate the knowledge. Knowledge is the primary ingredient in most performance models of human behavior. The architecture provides a language and a decision procedure for representing problem-solving. It also provides the critical function of learning as it performs, encoding information about the programming session into long-term memory. Mechanisms on top of the architecture decide the model's goals and mediate access to external and internal knowledge.

2.1. Overview of the model

In the behavior we are modeling, the programmer tries to comprehend a succession of specific objects in her program. She also periodically issues commands to the language interpreter to print new information.

The model reflects this behavior by setting itself a succession of *comprehension goals*. (As we use it, the term *goal* often implicitly refers to the object of the goal.) Comprehension goals arise from knowledge about what objects and relationships are important to understand. More comprehension goals are *proposed* (arise) than the model actually *selects*, meaning that the model has to decide what it will think about next, and when. Having selected a comprehension goal, the model proceeds by trying to *generate* knowledge relevant to that goal. It can bring new knowledge in from the display, or try to recall knowledge from its long-term memory (LTM). In the latter case, generation consists of bringing old knowledge into a new context.

Generated knowledge accumulates in working memory (WM). Reflecting limitations on human WM, the model's WM is ephemeral, holding only the knowledge generated during the current goal and the immediately previous goal.

Interleaved with comprehension goals are occasional *display goals*, which cause the model to issue interpreter or editor commands that change the information on the display. These goals represent a combination of knowledge about the programming environment and the language interpreter, and tactical knowledge about when to issue such commands. The model knows, for example, to display objects that it has set a goal to comprehend, either by printing or scrolling. Just as the programmer is skilled at manipulating her environment, the model does not need to deliberate to accomplish its display goals; the display emulator immediately responds with the requested information.

2.2. Knowledge: Expert, external, and episodic

Represented in the model are three distinct kinds of knowledge: *expert*, *external*, and *episodic*. Expert knowledge is what we would expect a skilled programmer to bring to a programming task. This comprises knowledge about the particular program to be modified, including specific data structures and functions; knowledge about the implementation language, including its central concepts and idioms; and knowledge of computer science fundamentals, like data structures and algorithms. Such knowledge is typically found in expert systems and other symbolic AI programs. Expertise also includes knowledge of the programming environment, including procedures for navigation. This is the kind of expertise represented in the operators, methods, and selection rules of GOMS models (Card *et al*, 1983).

To account for the behavior at hand, the model has expert knowledge about program objects and how they relate to each other. Figure 2 shows a lattice representing a subset of this knowledge. Some of what the model knows is specific to the program that the programmer is working on, and some of what it knows is general to the production-system language in which the program is written. The model also has tactical knowledge about how to interact with its programming environment. For example, it knows in general to display objects that it is trying to comprehend, whether by printing them out, or by finding them in hidden regions of the display if it can recall that they are there.

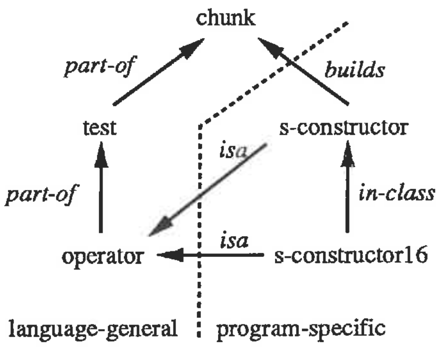


Figure 2: Objects and relations in the programmer's program.

External knowledge rests in the display, coordinating with internal knowledge to extend a problem-solver's effective memory (Davies, 1993, Green *et al*, 1987, Larkin, 1989). It has an immediately-available component, which is visible. It also has a hidden component, which can be made accessible by navigation. Computational models that address the display as external knowledge often treat only the immediately-available component (e.g., Bauer and John, 1995, Kitajima and Polson, 1992). But in real programming situations, and in our model, only a small fragment of external knowledge is immediately available at any given time.

Episodic knowledge describes a particular session with the interpreter, including knowledge about external knowledge that was visible once but is now hidden. In our protocol, for example, the programmer's navigation behavior shows that she knows she has seen something earlier in the session. Because episodic knowledge only arises in the course of a session, it must be learned on the fly. In contrast to computational models of the acquisition of expert procedural knowledge (e.g., Howes, 1994, Polson and Lewis, 1990, Rieman *et al*, 1994, Vera *et al*, 1993), our model accounts for the encoding and retrieval of episodic knowledge used as long-term working memory (Ericsson and Kintsch, 1995).

2.3. Soar as the underlying architecture

The model's underlying cognitive architecture is Soar (Newell, 1990). Soar provides a production-rule representation for knowledge, a decision-making process for selecting cognitive operators, and an inherent learning mechanism that encodes new long-term knowledge as it performs. Goals, attention acts, and LTM probes (described in Section 2.4) are all represented as Soar operators, proposed by rules in Soar's LTM.

The learning mechanism provides us with a natural way to account for the encoding of long-term working memory in expert behavior. A full discussion of the learning mechanism (Laird *et al*, 1986) is beyond the scope of this paper, but as it works in our model, learning occurs as a side-effect of generating new knowledge. The architecture encodes new rules in LTM to avoid having to generate this knowledge again.

The conditions of learned rules are highly specific to the WM context at the time the rule was encoded. Generally the conditions are sensitive to both display features and goals. Display conditions arise when the model looks to the display for new knowledge. Goal conditions arise because the object being comprehended typically helps determine what new knowledge is relevant. We refer to the new rules that Soar encodes as *episodic* because their conditions capture much of what defines a particular temporal context.

Because episodic rules are so specific, retrieving the knowledge they represent requires a process that can intelligently reconstruct WM contexts. This process, which we call *probing*, is described in the next section.

2.4. Mechanisms that manipulate knowledge

Our model augments the architecture with generic, mechanistic knowledge that carries out several important functions. The mechanisms discussed below terminate goals and select new ones (*goal selection*), bring information from the display into WM (*attention*), and search LTM by intelligently reconstructing WM contexts (*probing*).

2.4.1. Goal selection

The model contains knowledge about what program objects are important to comprehend. But how does it select which goal to work on? How does it know when it has thought enough about one object and that it is time to move on? Goal selection is a complex decision. It depends on the availability of other goals, whether those goals are relevant to the current train of thought, and whether the current goal is by some criterion achieved.

These factors bear on how well the model is able to be both purposeful and flexible in its thinking. There is a tension between purposefulness, where each new goal follows from conditions associated with the previous one, and flexibility, where to emulate human behavior properly the model must be able to leap to thinking about something only tenuously related. There is also a need to keep the model from being distracted by what could be floods of goals arising from knowledge it gathers. When there is too much to think about, there must be ways to pick and choose.

Until a new goal is proposed, there is no issue: the model continues working on the current one. When a new goal is proposed, the model uses the criterion of *salience* to decide whether to terminate the current goal and select the new one. Salience requires that the new goal is something that the model has decided is important to think about. Functionally, a goal becomes salient when the model attends to the actual goal object on the display, or generates this object as a cue for probing. In both cases, knowledge in the model has said that the object is worth knowing more about.

A third condition of salience governs the interaction of comprehension goals and display goals. Display goals arise in response to comprehension goals, for example when the object to be comprehended is newly-generated by the program and needs to be printed out. The model automatically re-selects the comprehension goal once the display goal is done; the model never forgets why it just did something.

2.4.2. Attention

The model interacts with an emulation of the display. The model issues display commands, and in response the emulator delivers a representation of the new display features. The emulator also removes the features that scroll off the programmer's display. The role of the model's attention mechanism is to bring elements from the emulated display into WM.

Attention occurs through operators. Each attention operator corresponds to a display feature. A display feature enters WM when the model selects an attention operator for that feature.

Attention operators are proposed by attention knowledge. The entire visible display is available for this knowledge to examine. When there are multiple attention operators to choose from, the model prefers to attend to the information region displayed by the most recent display goal. Within an information region, the model prefers to attend top-to-bottom.

Attending to a feature causes the model to encode that feature in a new rule. The new rule encodes both the content of what was attended to, and that it was attended to on the display (as opposed to having been recalled from LTM). This knowledge lets the model recall knowledge of displays that become hidden later in the session.

2.4.3. Probing

As described in Section 2.3, the model automatically learns rules that recognize specific situations. But how can we get the model to recall something later when the goal or the display has changed? Just as with people, recall is more difficult than recognition, and the model resorts to an effortful process that we call *probing*.

Probing also occurs through operators. Each probe operator corresponds to a cue that could satisfy a condition of some rule in LTM. The cue enters WM when the probe operator is selected. There are two kinds of cues: *semantic* and *image*. These correspond to the two kinds of conditions in episodic rules: goals and display features. Semantic cues are objects that may have been goals. Image cues are objects that may have been on the display.

Probe operators are proposed by expert knowledge about what cues may be useful. For instance, the model knows that operators (in the language it is working in) are a primary functional unit, and that recalled knowledge about a specific operator (in the program it is working on) is likely to be relevant to any given goal. When an operator (from the program) appears in WM, the model automatically proposes using it as a semantic cue (rule 12, Section 3.2.3). As a memory search mechanism, probing is similar to Rist's cue-based memory search (Rist, 1995).

In principle, probing can activate any of the model's knowledge, both expert and episodic. The extent to which it does in practice depends on its success at generating cues. With respect to Figure 2, the more objects and relations the model has access to, the better it will be able to search its LTM. Thus as a general problem-solving mechanism, probing has the important property that it can transform additional expertise into better performance. Ericsson and Kintsch (1995) observed this property in humans' use of long-term working memory.

3. Tracing the Model's Behavior

In this section we trace two separate but related episodes of the model's behavior, giving examples of the use and interaction of knowledge, architecture, and mechanism. In the first episode, E1, the model encodes episodic knowledge about a display feature. In the second episode, E2, the model begins by printing out an object on the display. This sets off probing, attention, and goal selection activity that eventually activates the episodic knowledge encoded during E1.

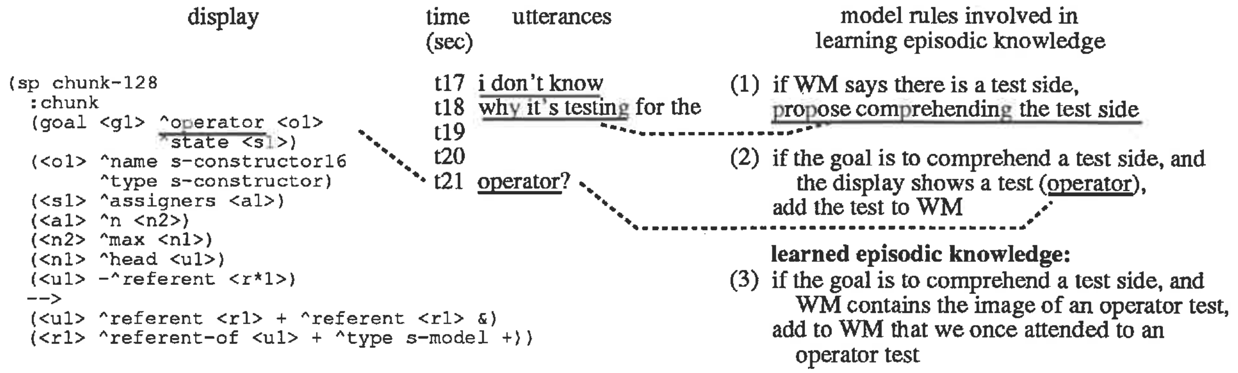
Figures 3 and 4 show sequences of rules in the model that fire in the course of emulating the programmer's behavior, copied from Figure 1. The dashed lines in the figures make connections where display features and utterances are explicit evidence for the model's knowledge.

3.1. Learning episodic knowledge in E1

The programmer's high-level task for this session is to modify her program so it will learn slightly different natural-language comprehension rules (or *chunks*). To do this she needs to understand the chunks that the program currently learns. In E1 she is trying to understand a chunk on the display (to the left, in Figure 3) and says, "I don't know why it [the chunk] is testing for the ... operator?" (t17). This indicates that she has attended to the operator test on the chunk's test side.

Proposing a goal. Reflecting what the programmer is trying to understand at this point, there is a chunk represented in the model's WM. The model knows that chunks are important to comprehend, when working on the program that generates them. The model also knows that syntactically chunks consist of test sides and action sides, and that comprehending the parts is a good way to comprehend the whole. This knowledge proposes a goal to understand the test side.

Goal proposal (language knowledge):
if WM says there is a test side, (1)
propose comprehending the test side



display

```
(sp chunk-128
  :chunk
  (goal <g1> ^operator <o1>
             ^state <s1>)
  (<o1> ^name s-constructor16
        ^type s-constructor)
  (<s1> ^assigners <a1>)
  (<a1> ^n <n2>)
  (<n2> ^max <n1>)
  (<n1> ^head <u1>)
  (<u1> -^referent <r*1>)
 -->
  (<u1> ^referent <r1> + ^referent <r1> &)
  (<r1> ^referent-of <u1> + ^type s-model +))
```

time (sec)	utterances
t17	i don't know
t18	why it's testing for the
t19	
t20	
t21	operator?

model rules involved in learning episodic knowledge

(1) if WM says there is a test side, propose comprehending the test side

(2) if the goal is to comprehend a test side, and the display shows a test (operator), add the test to WM

learned episodic knowledge:

(3) if the goal is to comprehend a test side, and WM contains the image of an operator test, add to WM that we once attended to an operator test

Figure 3: Display, protocol, and model rules for episode E1. Dashed lines make connections where display features and utterances are explicit evidence for the model's knowledge.

Attending to the display. The model selects this goal and continues with a sequence of attention and probe operators. (Goal selection, attention, and probing will be described in detail in the context of E2, below.) In the course of this sequence the model attends to an operator test on the display. The model knows that it is important to understand the individual tests on the test side of a chunk, because they are key to understanding the chunk's high-level functionality.

Displayed:

```
(sp chunk-128
    :chunk
    (goal <g1> ^operator <o1>
               ^state <s1>)
    (<o1> ^name s-constructor16
```

Attention (language knowledge):
if the goal is to comprehend a test side, and
  the display shows a test (operator),
add the test to WM (2)

Encoding episodic knowledge about the display. New knowledge appearing in WM causes the model to encode a new rule. When new knowledge is generated by attention, the new rule encodes both what was attended to, and that it was attended to. When the following new rule fires, it will remind the model that it saw an operator test on the display.

Learned episodic knowledge:
if the goal is to comprehend a test side, and (3)
  WM contains the image of an operator test,
add to WM that we once attended to an operator test

The model fails to attend to the name of the operator test, s-constructor16. This is consistent with the programmer's protocol, in which she says nothing about s-constructor16. Later on, in E2, the model is unable to recall seeing s-constructor16 on the display, despite s-constructor16 being present on the display during E2. This lack of specific knowledge about s-constructor16 plays a role in the model's explanation for why the programmer had to scroll back to the E1 display (Section 3.3).

3.2. Attention, probing, and episodic recall in E2

In the second episode, beginning 6 minutes later in the session, the programmer is trying to comprehend a program object called o29 (the information on the display during this episode is in the upper left of Figure 4). She says, "New operator o24, which is in fact s-constructor16" (t379). Then there is a pause, followed by an expletive: "Eeuuu rats" (t388). Then, "I think all those chunks I built test for operator six, uh, s-whatever-it-is; I think they test for the s-constructor" (t397). During the last utterance, she begins scrolling through previous screens, apparently with some hypothesis about chunks testing an s-constructor. Eventually the display from E1 scrolls into view, and she confirms her hypothesis: "Yeah, s-constructor16, there it is" (t412).

In the trace we describe below, the model is also trying to comprehend o29. It first attends to the subobject o24. Then it finds another instance of o24, where o24 identifies operator s-constructor16. Attending to s-constructor16 begins a sequence of probes that leads the model to recall that it saw an operator test as part of a chunk. Because this operator test is important to comprehend but hidden, the model scrolls to find it.

3.2.1. Comprehending o29 by attending to subobject o24

Printing an object. We pick up the trace where the model selects a goal to comprehend object o29. Because the object is not displayed, the model sets a display goal to print it.

Goal proposal (display-manipulation knowledge):

```
if the current goal is to comprehend an identifier (O29), and      (4)
  the identified object is not on the display,
print the object
```

The display emulator responds to the print command and puts a representation of object o29 onto the display. With object o29 displayed, the goal selection mechanism re-selects the goal to comprehend o29, based on the assumption that the goal will still be relevant.

Displayed:

```
(O29 ^name return-operator
     ^new-operator O24)
```

Goal selection (mechanism):

```
if the current goal is a display command, and                      (5)
  the desired display contents have been generated,
re-select the last comprehension goal (O29)
```

Attending to a subobject. At this point, the model attends to new-operator o24, the only subobject of object o29. The model knows that it should attend to subobjects of the object that it is trying to comprehend.[1]

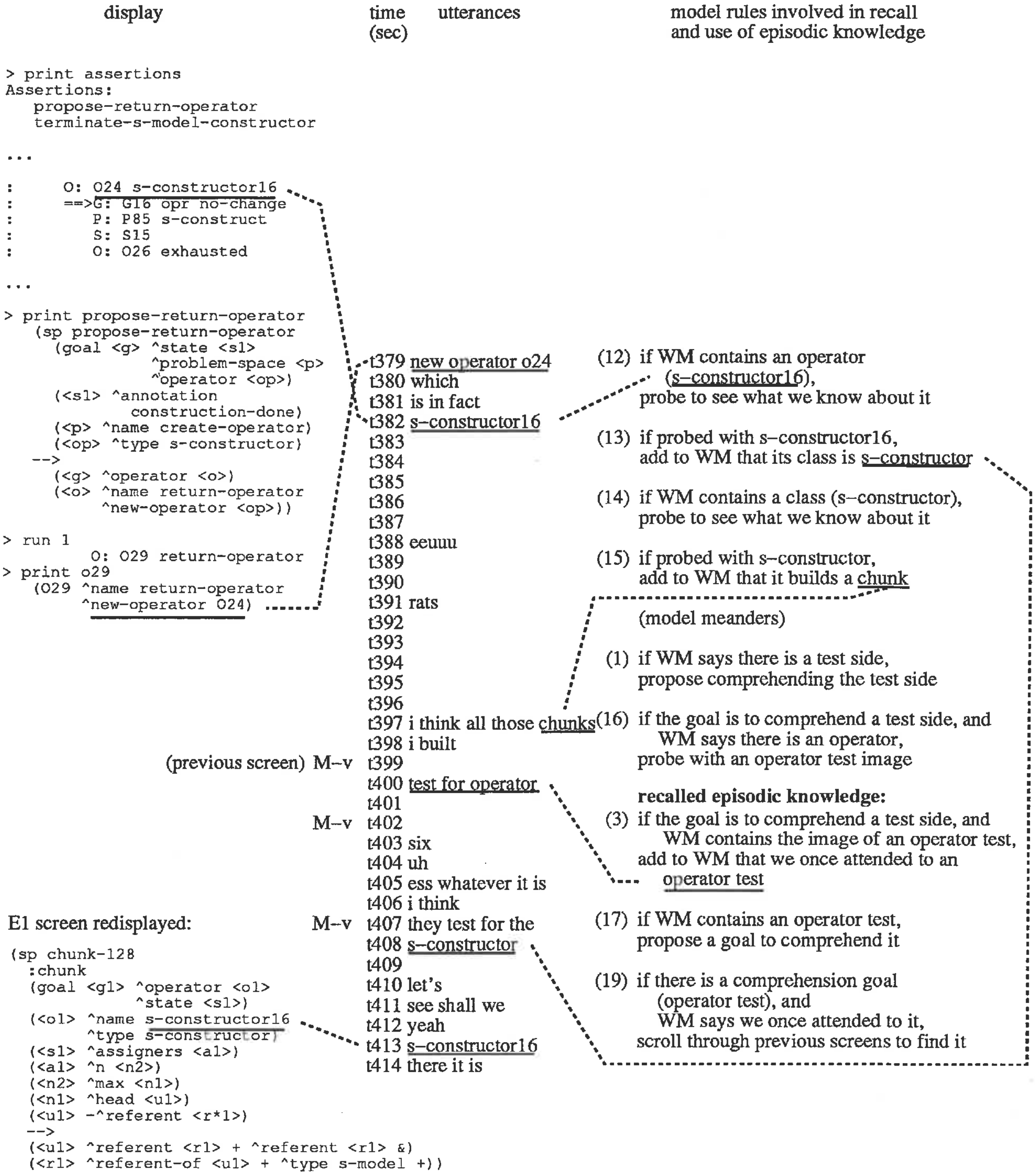
Displayed:

```
(O29 ^name return-operator
     ^new-operator O24)
```

Attention (language knowledge):

```
if the goal is to comprehend an identifier (O29), and              (6)
  the display shows a subidentifier (O24)
add to WM the subidentifier (O24) and the attribute that describes it (new-operator)
```

[1] The act of attending to o24 builds an episodic rule similar to that learned during E1. These rules are always learned and are omitted from the rest of the description of the model's behavior.



display

```
> print assertions
Assertions:
   propose-return-operator
   terminate-s-model-constructor

...

:      O: O24 s-constructor16
:      ==>G: G16 opr no-change
:         P: P85 s-construct
:         S: S15
:         O: O26 exhausted

...

> print propose-return-operator
   (sp propose-return-operator
     (goal <g> ^state <s1>
               ^problem-space <p>
               ^operator <op>)
     (<s1> ^annotation
               construction-done)
     (<p> ^name create-operator)
     (<op> ^type s-constructor)
   -->
     (<g> ^operator <o>)
     (<o> ^name return-operator
          ^new-operator <op>))

> run 1
         O: O29 return-operator
> print o29
  (O29 ^name return-operator
       ^new-operator O24)
```

(previous screen) M–v

M–v

E1 screen redisplayed: M–v

```
(sp chunk-128
  :chunk
  (goal <g1> ^operator <o1>
             ^state <s1>)
  (<o1> ^name s-constructor16
        ^type s-constructor)
  (<s1> ^assigners <a1>)
  (<a1> ^n <n2>)
  (<n2> ^max <n1>)
  (<n1> ^head <u1>)
  (<u1> -^referent <r*1>)
  -->
  (<u1> ^referent <r1> + ^referent <r1> &)
  (<r1> ^referent-of <u1> + ^type s-model +))
```

time (sec) utterances

t379 new operator o24
t380 which
t381 is in fact
t382 s–constructor16
t383
t384
t385
t386
t387
t388 eeuuu
t389
t390
t391 rats
t392
t393
t394
t395
t396
t397 i think all those chunks
t398 i built
t399
t400 test for operator
t401
t402
t403 six
t404 uh
t405 ess whatever it is
t406 i think
t407 they test for the
t408 s–constructor
t409
t410 let's
t411 see shall we
t412 yeah
t413 s–constructor16
t414 there it is

model rules involved in recall and use of episodic knowledge

(12) if WM contains an operator
(s–constructor16),
probe to see what we know about it

(13) if probed with s–constructor16,
add to WM that its class is s–constructor

(14) if WM contains a class (s–constructor),
probe to see what we know about it

(15) if probed with s–constructor,
add to WM that it builds a chunk

(model meanders)

(1) if WM says there is a test side,
propose comprehending the test side

(16) if the goal is to comprehend a test side, and
WM says there is an operator,
probe with an operator test image

recalled episodic knowledge:

(3) if the goal is to comprehend a test side, and
WM contains the image of an operator test,
add to WM that we once attended to an
operator test

(17) if WM contains an operator test,
propose a goal to comprehend it

(19) if there is a comprehension goal
(operator test), and
WM says we once attended to it,
scroll through previous screens to find it

Figure 4: Display, protocol, and model rules for episode E2. Dashed lines make connections where display features and utterances are explicit evidence for the model's knowledge.

Comprehending the subobject. The model then moves on to trying to comprehend object o24. Both knowledge and mechanism play a role in this decision: language-specific knowledge about what entities are important to comprehend, and a general mechanism for switching to a goal that arises during work on the current one.

Goal proposal (language knowledge):
if WM contains an identifier (`o24`), (7)
propose comprehending the identifier

Goal selection (mechanism):
if a new goal is proposed (`o24`) (8)
  and we attended to that goal object on the display,
terminate the current goal (`o29`) and select the new one

Rule 7 represents knowledge about data structures in the language. They are often linked by identifiers in complex lattices, and this rule allows the model to traverse a lattice to comprehend each node in turn.

3.2.2. Following perceptual links from o24 to s-constructor16

Probing with an attribute. The model has now set itself the goal to comprehend the object identified by o24. It probes with the attribute new-operator, which describes o24. The probing mechanism knows that recollections about an attribute can inform the object it describes. To recall what the model might know about new-operator, the probing mechanism proposes new-operator as a semantic cue.

Probing (mechanism):
if there is an identifier in WM (`o24`), (9)
  and an attribute (`new-operator`) describing that identifier,
probe with that attribute

Attention enabled by previous semantic cue. Each probe leaves its cue behind in WM. Semantic cues, like the one above, are represented in WM as goals. Because attention knowledge is often goal-sensitive, semantic cues left behind by probing can enable new acts of attention. Thus the longer the model spends trying to understand an object, the more external memory it searches.

The new-operator cue enables the model to attend to a rule on the display in which new-operator appears. On the action side of the rule, new-operator describes the variable <op>. On the test side of the rule, this same variable is described by the operator attribute. Because new-operator also describes o24, the model infers that o24 is also an operator.

Displayed:

```
(sp propose-return-operator
    (goal <g> ^state <s1>
              ^problem-space <p>
              ^operator <op>)
    (<s1> ^annotation
           construction-done)
    (<p> ^name create-operator)
    (<op> ^type s-constructor)
  -->
    (<g> ^operator <o>)
    (<o> ^name return-operator
         ^new-operator <op>))
```

Attention (language knowledge):
if there is a goal to comprehend an attribute (`new-operator`), and (10)
  there is a rule on the display, and
  the attribute describes a variable on the action side, and
  another attribute (`operator`) describes the same variable on the test side,
add to WM that objects described by the first attribute (`new-operator`)
  are also described by the other attribute (`operator`)

Attention enabled by previous attention. The model has now inferred that object o24 is an operator, by following perceptual links from the new-operator attribute in object o29, to the new-operator attribute on the action side of a rule on the display, to the operator attribute on the test side of the same rule.

The model now follows another perceptual link. Another region of the display shows a stack describing the current state of the program. The stack contains slots for major program objects, including operators. Knowing that o24 is an operator, the model looks among the operator slots in the stack. One of the operators in the stack has identifier o24. The model attends to the name of that operator (s-constructor16), adding it to WM.

Displayed:

```
:      O: O24 s-constructor16
:      ==>G: G16 opr no-change
:         P: P85 s-construct
:         S: S15
:         O: O26 exhausted
```

Attention (language knowledge):

```
if there is a goal to comprehend an identifier (O24), and                    (11)
  WM says the identifier is an operator, and
  the display shows a stack with an operator with that identifier,
add the name of that operator (s-constructor16) to WM
```

The model has now been led, by expert knowledge interacting with the probing and attention mechanisms, to the conclusion that new-operator o24 is, in fact, s-constructor16.

3.2.3. Probes that lead from s-constructor16 to episodic recall

Probing with an operator. The operator s-constructor16 has appeared in WM, and the model knows that operators are both ubiquitous and one of the main functional units in the language. The probing mechanism guesses that knowledge about an operator will inform any context in which the operator appears, and uses operators in WM as semantic cues. Thus the model probes to see what it knows about operator s-constructor16.

The expert knowledge retrieved by this probe is that s-constructor16 is an instance of a particular class that is central to the program (Figure 2). The s-constructor class is abstract, representing the many instances like s-constructor16 that the program generates during execution.

Probing (mechanism):

```
if WM contains an operator (s-constructor16),                                (12)
probe to see what we know about it
```

Recall program knowledge:

```
if probed with s-constructor16,                                              (13)
add to WM that its class is s-constructor
```

In general, any number of rules like 13 can fire in parallel in response to a probe. Thus different pieces of information about s-constructor16 could appear at this point in the trace. In particular, had the model attended to s-constructor16 as the name of the operator test (during E1), it would have encoded episodic knowledge about having seen s-constructor16. That rule could have fired now, reminding the model of the earlier episode. However, in E1 the model attended only to the operator test, thus has no episodic knowledge about its name. This lack of knowledge plays a role in the explanation the model provides of why the programmer had to scroll back to the E1 display to confirm her hypothesis about the name of the operator test (see Section 3.3).

Probing with a class. No new goal has been proposed to replace the current goal (o24), so the model continues probing and attending to the display. The knowledge it has recalled about the class of s-constructor16 provides a new semantic cue: the probing mechanism knows that information about a class could be relevant to instances of that class. Thus the model probes with the class, s-constructor.

This probe retrieves an association between the s-constructor class and chunks. All s-constructor operators build chunks; this is expert knowledge about the program. This association is key to the programmer because she wants to modify the program so that s-constructor operators build slightly different chunks.

Probing (mechanism):
if WM contains a class (`s-constructor`), (14)
probe to see what we know about it

Recall program knowledge:
if probed with s-constructor, (15)
add to WM that it builds a chunk

Meandering that intervenes in the probe sequence. The model now digresses from the train of thought that leads it from attending to operator s-constructor16 during E2 to a recollection about the E1 display. We see this digression as the model's way of trying different avenues of thought, not all of which are directly relevant to the eventual outcome. The model thinks about the operator recently selected by the program, some rules printed on the display that are ready to fire in the program, and test sides and action sides of rules.

In the programmer's behavior, there are 14 seconds of pause and expletive between her statement that new-operator o24 is in fact s-constructor16, and her hypothesis that the chunks the program built earlier test for an operator. The model's digression suggests that the programmer may be following similar dead-end avenues of thought during this pause.

Probing with an image cue. Through its meandering, the model has accumulated several seemingly unrelated items in WM. On one hand it has knowledge in WM about an operator. On the other hand, rule 1 has fired, and the model has selected the goal to comprehend test sides. Language expertise puts these elements together, recalling that operators are ubiquitous tests and that the model may have attended to an operator test recently. The model generates an image cue consisting of an operator test.

Probing (mechanism):
if the goal is to comprehend a test side, and (16)
  WM says that there is an operator,
probe with an operator test image

Recall of episodic knowledge. With the model trying to comprehend a generic test side, and WM containing an operator test image, rule 3 fires. Encoded during E1, this rule recalls for the model that it saw an operator test on the display earlier this session, by recognizing the image cue the model generated now.

Recall episodic knowledge:
if the goal is to comprehend a test side, and (3)
  WM contains the image of an operator test,
add to WM that we once attended to an operator test

Rule 3 also recalls the content of what was attended to earlier on, namely an operator test. The appearance in WM of an operator test activates language knowledge that operator tests are important to comprehend. This knowledge proposes a goal. The goal is salient because it is also a cue (generated by rule 16, above).

Goal proposal (language knowledge):
if WM contains an operator test, (17)
propose a goal to comprehend it

Goal selection (mechanism):
if a new goal is proposed (operator test) (18)
  and that goal object is a probe cue,
terminate the current goal (test side) and select the new one

3.2.4. Searching external memory based on episodic recall

At this point the model has both the goal to comprehend a feature and the knowledge in WM that this feature was previously on the display. This activates tactical display-manipulation knowledge that says that under these circumstances it is useful to re-display the hidden information. The model knows that it can find hidden information in a process buffer by scrolling through previous screens.

```
Goal proposal (display-manipulation knowledge):
if there is a comprehension goal (operator test), and                    (19)
  WM says we once attended to it,
scroll through previous screens to find it
```

3.3. Why did the programmer scroll?

During episode E2 the programmer attends to s-constructor16: "New-operator o24, which is in fact s-constructor16" (t379, Figure 4). This appears to lead to a hypothesis about the chunk she saw during episode E1: "I think all those chunks I built test for operator six, uh, s-whatever-it-is; I think they test for the s-constructor" (t397). She scrolls back, and confirms this hypothesis: "Yeah, s-constructor16, there it is" (t412). The s-constructor16 cue in E2 seems to evoke some memory for the E1 display, but her behavior suggests that this recollection is somehow tentative or imprecise.

The model suggests that the programmer did not encode the s-constructor16 test during E1. It suggests that she did encode the neighboring operator test, and that this memory was retrieved during E2. The memory for the operator test, related to s-constructor16 by her knowledge about the language and the program, led her to search external memory for confirmation.

4. Discussion

We have presented a computational model of how people may manage the large amount of internal and external memory needed for expert programming. Specifically, the behavior we modeled requires the use of *long-term working memory* (Ericsson and Kintsch, 1995) to retrieve hidden external knowledge. Our model encodes episodic knowledge as a byproduct of problem solving; this inherent learning comes for free by adopting Soar as the underlying cognitive architecture. This episodic knowledge mediates behavior that expertise and external memory cannot explain by themselves.

The learning mechanism in Soar imposes a key constraint on our model: encoded rules are highly-specific to the contents of WM at the time the rule was encoded. This specificity reflects the difficulty that people have with recall, compared to recognition. For the model to retrieve episodic knowledge it has encoded, it must to some degree reconstruct WM with the appropriate cues. The model thus provides a specific theory of how people retrieve such knowledge: they generate cues, consisting both of previous objects of comprehension and previous objects of attention. This cue-generation process is guided by expert knowledge about program objects and their relationships.

We have also identified a functional definition of the salience of goals. This allows the model to have general knowledge about what objects may be useful to comprehend, and to avoid being distracted by always wanting to think about everything important. Salience is defined by the model's current train of thought: what the model attends to, and what it wants to retrieve knowledge about through probing. One measure of the plausibility of our definition of salience is how well the model fits the data, in terms of general goals being selected at the right time and filtered out when they would not fit. In the course of the 7-minute lifetime of our model, it sets over 80 comprehension goals. The match to the data is on the order of the matches represented in Figure 4. An important next step will be to evaluate both the fit of the model and the effects of goal selection in more precise terms.

Our model is preliminary in several aspects. One is that the knowledge that guides probing is very weak. Probing and episodic recall are essentially default behaviors, that "just happen" to the model. This is implausible, in the sense that other knowledge probably bears on memory search to make it more directed. Given that people know a lot of things, it must take a lot of knowledge to tell them which of those things they should try to recall.

One source of knowledge that probably guides memory search is the programmer's high-level goal. In the behavior we examined, the programmer wants to change the s-constructor class to build slightly different chunks (Figure 2). This high-level goal could provide additional activation to her memory for both chunks and the s-constructor class. This could have helped guide the train of thought that led from the display feature s-constructor16, to a hypothesis about an s-constructor test, to scrolling to confirm the hypothesis.

The model is also preliminary is that it represents a single mode of programming behavior. This mode is a tight loop of interaction with a language interpreter, primarily for the purpose of comprehension. Studying this kind of program comprehension behavior will eventually fill out theories of program comprehension (e.g., Brooks, 1983), and complements comprehension studies in which the task is set by the experimenter and the presentation limited to hardcopy (e.g., Pennington, 1987, Wiedenbeck, 1991). Our protocol data include episodes of planning and coding, and thus provide an opportunity to extend the model to other programming subtasks.

Finally, our model now accounts for only one example of the use of episodic knowledge to retrieve external knowledge. The mechanisms described here need to be applied to other examples within the same protocol, and to other programmers.

5. Conclusions

Episodic knowledge is an important factor in expert programming. The behavior we modeled changed qualitatively because of a recollection that linked separate regions of a display. Episodic knowledge interacts with external knowledge and expertise to affect behavior — in our example, expertise first guided attention (and hence encoding), then connected a current display feature to recollection of a previous one.

Mechanisms for memory retrieval, as well as for attention and goal selection, enable the different knowledge sources to interact to bring about behavior. The structure of these mechanisms is highly constrained by adopting a computational cognitive theory as the underlying architecture.

6. References

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

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

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, and Newell, A (1983). *The Psychology of Human-Computer Interaction.* Hillsdale NJ: L Erlbaum.

Davies, S P (1993). Externalising information during coding activities: Effects of expertise, environment, and task. *Empirical Studies of Programmers: 5th workshop* (pp 42-61). Norwood NJ: Ablex.

Ericsson, K A, and Kintsch, W (1995). Long-term working memory. *Psychological Review, 102*(2), 211-245.

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.

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

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: L Erlbaum.

Kitajima, M, and Polson, P G (1992). A computational model of skilled use of a graphical user interface. *Human Factors in Computing Systems: Proceedings of CHI 92* (pp 241-249). New York: ACM Press.

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

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

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

Pennington, N (1987). Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive Psychology, 19*(3), 295-341.

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

Rieman, J, Lewis, C, Young, R M, and Polson, P G (1994). "Why is a raven like a writing desk"? Lessons in interface consistency and analogical reasoning from two cognitive architectures. *Human Factors in Computing Systems: Proceedings of CHI 94* (pp 438-444). New York: ACM Press.

Rist, R S (1995). Program structure and design. To appear in *Cognitive Science.*

Vera, A H, Lewis, R L, and Lerch, F J (1993). Situated decision-making and recognition-based learning: Applying symbolic theories to interactive tasks. *Proceedings of the 15th Annual Conference of the Cognitive Science Society* (pp 84-95). Hillsdale NJ: L Erlbaum.

Wiedenbeck, S (1991). The initial stage of comprehension. *International Journal of Man-Machine Studies, 35*, 517-540.