

Episodic Memory for External Information

Erik M. Altmann

August, 1996

CMU-CS-96-167

School of Computer Science

Carnegie Mellon University

Pittsburgh, PA 15213-3890

*Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy*

Thesis committee:

Bonnie John, Chair

John Anderson

Jim Morris

Clayton Lewis, University of Colorado, Boulder

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: Cognitive science, Soar, episodic memory, psychology of programming, human-computer interaction, artificial intelligence.

Abstract

People make use of hidden external information, first recalling that it exists and then finding it. This dissertation investigates the memory phenomena involved in recalling that external information exists. We present data in which a programmer navigates to hidden features in a real-world task environment. We then present a model that accounts for this navigation by encoding and using simple episodic memories for having seen a feature. The model inherits constraints from its underlying cognitive architecture, which specify that learning is passive and pervasive, and that it creates simple memories that depend on the feature itself being present as a cue. The nature of these memories requires the model to recall features to its mind's eye as cues in order to retrieve them. This retrieval process requires domain knowledge: familiarity with features in order to imagine them, and an idea of when it would be useful to recall having seen them. Recalling that a hidden feature exists prompts the model to scroll to that feature. Thus the model's access to external information is a function of passively-encoded episodic memories, and retrieval of these memories using knowledge. As a claim applied to people, this appears to overlap with a recently-published theory of *long-term working memory*. This theory proposes that experts, for example in chess, use long-term memory to expand their working memory in their domain of expertise. We propose a ubiquitous episodic long-term working memory, in which people store information about features with little effort, and from which they retrieve this information when it is relevant.

Acknowledgements

Thanks to Bonnie John for excellent advice time and time again, for fixating the strengths and weaknesses and retrieving them when they were relevant. Thanks to John Anderson, Clayton Lewis, and Jim Morris for their insights and questions, and to Jill Larkin, Jill Lehman, Peter Polson and Rob Rist for key discussions as the model took shape.

The School of Computer Science and the Soar community have been both home and brain trust. I'm deeply grateful for the investments of Allen and Noel Newell, and to those who participated in and carried forward Allen's vision.

My parents made extraordinary decisions in extraordinary times, and have lived values of which I am proud and for which I am exceedingly fortunate. They armed us, and then my sisters marked the target. I also gratefully acknowledge the guidance of my mentors at the University of Alberta, and the support of the Alberta Heritage Scholarship Fund and the Natural Sciences and Engineering Research Council of Canada.

Thanks to the hundreds of people at WRCT 88.3fm, a magnetic companion of impeccable taste, good humor, and competence. And thanks to my colleagues at the First Church of Christ, Abortionist, with whom I've enjoyed countless delightful excursions, and one delightful incursion.

—

Colleges like CMU are inquisitors, if more kindly than those of a darker age. CMU devised a speech code that made "offensiveness", as perceived by the "victim", the criterion for illegal speech; censored sexual expression from its netnews feed, boasting that the ensuing publicity helped with name recognition; and has routinely intimidated those who express unpopular or critical opinions. When a religious group here proclaimed that sex and abortion were magnificent gifts from a benevolent god, the administration hounded the group as "reprehensible" and banned it from the campus activities center — notwithstanding the fancy assurances of religious and academic freedom to be found in CMU's official publications (see <http://www.contrib.andrew.cmu.edu/~fcc>). Elsewhere, Princeton banned national politics from student web pages, Brown banned "Mexican night" at the fraternities, Emerson banned rap music from its radio station, UMass/Amherst banned "negative stereotyping", Penn the phrase "water buffalo", and so on.

Today's routine inquiry was often the heresy of another time, and a university pandering to cultural taboos is a mission failure as contemptible as it is paradoxical. Fortunately, we can thank others for protecting our interest in asking new questions. Recently Judges Sloviter, Dalzell, and Buckwalter disposed of the "Communications Decency Act". And one jurist has been this century's preeminent architect of pluralism. There is a moral vision, stalked by those like William Brennan, in which we are proud and curious owners of mind.

To Justice William Brennan, Jr.

Chapter 1

Introduction

Our environment is 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. We might return to that area in the book to refresh our memory, or to examine the context more carefully.

This dissertation 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 display at once. Most of this information is hidden, scrolled out of the way by the programming environment to make room for new information. However, old information remains accessible, and the programmer occasionally scrolls some hidden information back into view.

We set out to answer two specific questions about the programmer's scrolling behavior. First, what is it that she remembers about the old information she returns to? She must learn something about this information when it first appears, in order to remember later that it exists. We would like to know what she encodes, and under what circumstances she encodes it. Second, what causes the retrieval of these memories? She scrolls not randomly, but when the target information is relevant. Her recollections about hidden information appear to come out of her task-related activity. We want to understand the role of domain knowledge as a cue for memories about hidden information.

One way to characterize a memory for having seen something is as an *episodic* memory. An episodic memory represents an event — something occurring on a particular occasion, distinct from a memory for a fact with no temporal component (Tulving, 1983). Use of the notion of episodic memory has precedent in studies of programming behavior. For example, in a study of novice and experienced software designers, one novice failed to remember a previous design decision (Jeffries et al., 1981), whereas experienced designers show no such failures. The novice subject made notes about his initial decision — that is, he recorded information externally. Nonetheless, he apparently did not remember his notes even when they were relevant to his task. Jeffries et al. describe this as a failure to recall a previous problem-solving episode. One small part of the forgotten episode was the event of writing notes about it. Had the subject recalled the event of writing these notes, when this recollection was relevant to his train of thought, he could have used them to reconstruct his previous decision. A simple episodic memory could have enabled access to a richer information context.

The reason that experts in the Jeffries et al. study showed no failures of episodic memory may be that expert behavior includes the ability to gain access to external information when it is relevant. This would be consistent with *skilled memory theory* (Chase and Ericsson, 1982, Ericsson and Staszewski, 1989), and more recently the theory of *long-term working memory* due to Ericsson and Kintsch (1995). These theories argue that effective working memory — the “rapid and reliable access of a particular piece of information at a specific time” (Ericsson and Kintsch, p. 215) — improves as a function of knowledge about the task. For example, mnemonists use specialized knowledge to encode information so they can recall it later. More generally, experts show superior memory for dynamic information that arises in the course of problem solving in their domains of expertise. This performance edge again raises the question of how domain knowledge might enable recollections about task-relevant external information, particularly when this external information arises dynamically in the course of problem solving, with comparatively little opportunity for study.

The approach taken in this dissertation is to emulate the programmer’s scrolling behavior with a computational cognitive model. The purpose of the model is to help fill gaps in our knowledge about the programmer’s thinking. The details of its computations constitute hypotheses about cognitive activity that we could not observe directly.

Such hypotheses are plausible to the extent that the model’s behavior is grounded in constraints external to the model. The constraints on our model come from two sources: (1) the observable data on the programmer’s behavior — verbal and keystroke protocols — and (2) the cognitive architecture in which the model is implemented. To illustrate the role of the data, suppose that a feature arises on the programmer’s display for the first time, and that the programmer recalls this feature later after it becomes hidden. If these events are properly represented in the model and its simulated task environment, then the model must encode some memory for that feature and retrieve the memory later. The processes of encoding and retrieval that occur in the model are possible accounts of what occurred in the programmer’s mind. To illustrate the role of the cognitive architecture, suppose it provides a mechanism for encoding information, as does the architecture in which our model is implemented. An architectural learning mechanism will influence the nature of the memories encoded by the model (Howes and Young, 1996a), and hence also the processes necessary to retrieve encoded memories. An independently-motivated learning mechanism greatly improves the plausibility of the model’s hypotheses about encoding and retrieval.

There are other computational cognitive models that learn about hidden information in a computer interface. Models by Howes (1994) and Rieman et al. (1996) account for exploratory learning of interfaces that contain hidden features in the form of menu items. The models “pull down” menus and encode memories for having seen particular items, which the models use later to guide their behavior on successive passes over the menu. However, the set of menu items is limited, and is static in the sense of being a persistent part of the interface. Subjects who are computer users are likely to bring prior knowledge about menus to an exploratory-learning task, knowledge that may affect how and what they learn about menu items they encounter during the task. Thus these task environments, and hence the resulting models, do not directly address the question of what people remember about external information that is much more dynamic. For example, in the task environment we studied, there are large numbers of features, but a given

feature may be unique to a particular session. Any memories encoded about such a feature must be encoded while the feature is visible. This may be only briefly, and the amount of mental processing allocated to any one feature may be small.

The interval of behavior we have modeled contains several events in which the programmer scrolls to hidden information that was generated earlier in the same interval. Based on the dual constraints of this scrolling behavior and the underlying architecture's learning mechanism, our model leads to two hypotheses about the encoding and retrieval of episodic memories:

1. People passively encode large amounts of simple episodic information about what they see, and
2. They retrieve this information as a function of their knowledge about the task environment.

These hypotheses contribute to the study of how people make use of long-term memory to store dynamic information. The theories of skilled memory and long-term working memory introduced above account for the deliberate encoding of complex working information in long-term memory. We propose that people also use long-term memory more passively and pervasively, to store simple episodic information that enables access to hidden external information.

1.1. Outline of the thesis

In the rest of the thesis we examine how we arrive at the hypotheses above. Chapter 2 describes the task environment and the behavior we studied, describing in detail the examples of navigation through external information. Chapter 3 describes the cognitive model. Chapter 4 describes the model's account of the navigation events in the data, comparing model behavior to programmer behavior. Chapter 5 steps back to examine broader measures of the model's fit to the protocol data, taken over the entire life of the model. Chapter 6 discusses outstanding issues — the role of domain knowledge in bringing about scrolling behavior, a review of what the model learns as it runs, the role of a limited working memory, the challenge of modeling goal selection, and some limitations of the model. Chapter 7 relates our data and model to Ericsson and Kintsch's theory of long-term working memory (Ericsson and Kintsch, 1995). Chapter 8 summarizes the contributions to cognitive science, human-computer interaction, and the psychology of programming, and indicates directions for future work.

Appendix A is a complete configuration diagram of the model accompanied by a review of the different components. Appendix B describes key elements of the model in Soar terms. It maps our vocabulary for describing the model onto standard Soar vocabulary; describes the implementation of goal selection, fixation, and imagining; and gives an example of the Soar process of data chunking. Appendix C is the model source code, with rules indexed alphabetically and by category. Appendix D contains the actual model traces and code on which the abstract traces in Chapter 4 are based. Appendix E contains the contents of the programmer's display, the programmer's verbal and keystroke protocols, and the model trace. All three are aligned at the commands issued by the programmer and the model. Finally, Appendix F contains a very detailed trace of the entire life of the model, showing all operator selections, learning activity, and rule firings.

Chapter 2

The data

This chapter describes the behavior we studied, which encompasses several examples of navigation through external information. Section 2.1 describes the task environment and the session we observed, and makes global observations about the data. Section 2.2 introduces the domain of the programmer. Section 2.3 describes five navigation events in detail.

2.1. Task environment and overview of session

The programming session we studied was part of a long-term project to create a natural-language comprehension program in a production language. (The program and language are described in Section 2.2.) The programmer's high-level goals for this session include increasing her understanding of the program and changing it in a specific way.

The session lasts 80 minutes. The programmer ran her program interpretively in a GNU Emacs process buffer, and toward the end of the session visited existing files of code and created new ones. The programmer thought aloud, and we recorded her utterances and gestures on video. We instrumented Emacs to record a time-stamped keystroke protocol and the contents of the language-interpreter and file buffers.

2.1.1. Global observations of the programmer's navigation

The programmer used both scrolling and string searching to find hidden information, with scrolling predominant. There were 26 scrolling events — each consisting of consecutive, same-direction scrolling commands — in the 80-minute session, or roughly one event every 3 minutes. In total the programmer scrolled 2482 lines of text through a 60-line window, or roughly 41 screens. Figure 1 shows the distribution by number of screens covered. Most events (14) covered only one screen, and the most protracted event covered only 6 screens.

The programmer searched only three times. One search succeeded in finding the target string, with roughly 2 screens between the start position and target string. The two other searches failed to find the target string. After both failed searches, the programmer tried scrolling. Both scrolling sequences also failed, one after 3 screens and the other after 6 screens. While the very limited use of methods may seem surprising, it is consistent with a finding that experienced interface users use only small subsets of the commands available to them in an editor, ignoring even important cursor-movement commands (Payne, 1991).

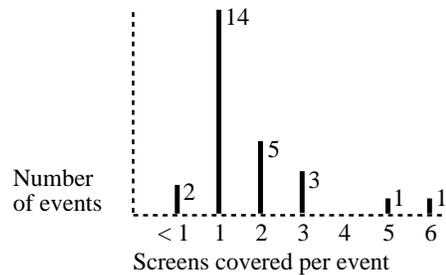


Figure 1: Scrolling events, by screens covered

These data suggest that scrolling in particular and navigation in general are common during real programming. Understanding the underlying mental processes could have practical importance for the design of navigation support.

2.1.2. Scrolling from long-term memory

Scrolling relied heavily on long-term memory (LTM). In 17 of the 26 scrolling events — roughly two-thirds — the target information had not been on the display in the past 30 seconds. This is the duration identified by Card et al. (1983) as the length of a unit task. This is also the length of interruption used to test the contents of readers' LTM in text comprehension studies (reported in Ericsson and Kintsch, 1995). Judging from the programmer's comments about features on the display, the objects she tries to comprehend appear to change more frequently than this, implying that there is enough activity between the disappearance of old information from the display and the programmer's recollection of it to interfere with rehearsal. There is little chance in these scrolling events that short-term working memory (WM) could account for the programmer's recollections. We refer to these 17 events as *LTM scrolling events*.

For 8 of these LTM scrolling events we were able to identify the episode during which the programmer encoded the memory that later prompted her to scroll. Figure 2 shows these 8 events on a timeline as they occur during the session. For each one, the beginning of the thick line is when the feature is generated, and the end of the thick line is when the feature becomes hidden. The "s" character is when the programmer scrolled to redisplay the feature. Because the memories for these features must have been encoded during the session, we refer to these 8 LTM scrolling events as *situation-specific*.

Each situation-specific LTM scrolling event has two halves. In the first half, which we refer to as the *encoding episode*, the programmer takes away some memory for some feature on the display. In Figure 2, the encoding episode is a subinterval of the thick line. The thick line denotes the time that the feature is visible. The programmer's commands and utterances narrow the encoding episode down to a subinterval during which she is most likely to have encoded a memory for the feature.

In the second half of each scrolling event, which we refer to as the *recall episode*, the programmer scrolls to the screen of the encoding episode, presumably on the basis of some memory for it. In Figure 2, the recall episode is in the neighborhood of the "s", which indicates only the actual scrolling command. The

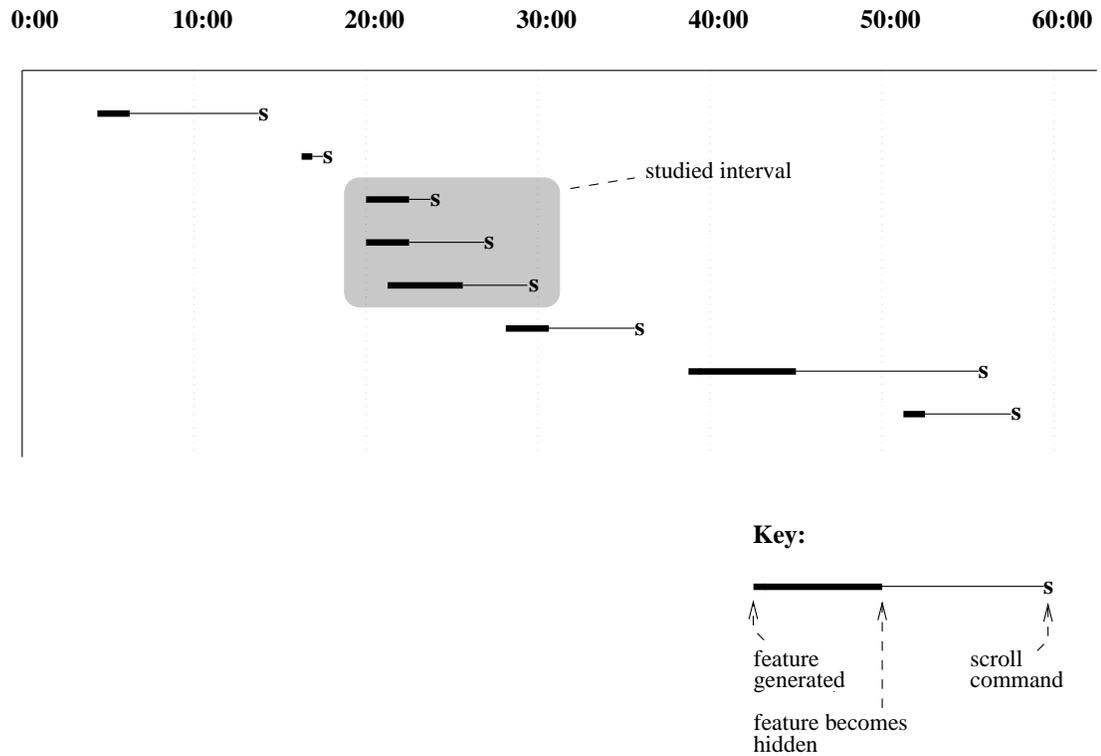


Figure 2: Timeline of long-term memory scrolling events

protocol before and to some extent after the scrolling command suggests what might have led to the programmer's recollection.

The situation-specific nature of these 8 events is linked to the programmer's extensive use of the process buffer. Because the information in the buffer is generated dynamically, any memories the programmer has about information in the buffer must also be encoded dynamically. In contrast, the remaining 9 of the 17 LTM scrolling events involve navigation through files of code that existed prior to the session; the encoding episode (or set of episodes) occurred in the past and is not available for study.

2.1.3. The studied interval of the session

From the 80-minute session we selected a 10.5-minute interval to study in detail (shaded in Figure 2). This interval contains a cluster of three situation-specific LTM scrolling events, which are typical of the rest in how long the feature is visible and then hidden. This interval also contains two other situation-specific scrolling events in which the programmer could have maintained information about the target screen in WM, making five events in total. Thus the interval provides a high concentration of navigation activity.

The main activity of the programmer during the studied interval is to comprehend a particular phase of the program's behavior at a fine grain. She works entirely in the process buffer, running the program one step

at a time in the interpreter and periodically printing out data structures, the stack of runtime execution contexts, and code. The printed information appears at the bottom of the process buffer, with Emacs automatically scrolling old output off the top when more room is needed.

2.2. Overview of the program and the language

The programmer's program is a large natural-language comprehension system. It is written in Soar, a cognitive architecture based on a production-system language with an inherent learning mechanism (Rosenbloom et al., 1992, Newell, 1990). This section describes the elements of the program and the language that arise in the discussion of the programmer's protocol.

The programmer's domain is complex and her knowledge detailed, raising a standard obstacle to the study of memory behavior in experts (Ericsson and Kintsch, 1995). However, some understanding of the main elements of the language and the program is necessary to make sense of the programmer's behavior. As the vehicle for this introduction, we use three screens taken from the scrolling events we describe later (Figure 3). These screens together show all the domain objects that arise in our discussion. To explain the contents of each screen, we step through the programmer's commands and their effects, elaborating where appropriate on the objects printed out and their function or structure. Further details about the domain are introduced as needed.

In Figure 3, the top screen (from scrolling events 1 and 2) shows the programmer running the program, printing out information about what code is going to fire next, and then printing some of this code. The middle screen (also from events 1 and 2) shows a partial print-out of the execution stack, and an example of the program running and modifying itself. The bottom screen (from scrolling event 3) shows the format of data structures in the language.

In the top screen, the top *run* command runs the program one cycle, which is the smallest commonly-used step. Every few cycles, the program generates output corresponding to some internal event. The top run command leads the program to select an *operator*. An operator is a functional object that modifies the program's *state*, which is the primary data structure in a *problem space*. The actions of an operator are carried out by *SPs* (Soar productions) that fire when that operator is selected.

When the operator is selected, the programmer issues a *match-set* command, showing which SPs are *asserted* (matched). These represent the program code that will carry out the selected operator.

The programmer prints out one of the asserted SPs (rather than visiting the code file that contains the SP). SP names are typically too long to type effectively, so the programmer relies on editor commands to copy SP names from sources like the asserted set. An SP has a *left-hand side*, which contains the *conditions* that must be satisfied for the SP to fire, and a *right-hand side*, which specifies the *actions* that will be carried out when the SP fires.

In the middle screen, the top of the screen shows a partial *problem-space stack*, which we also refer to as

sample screens from scrolling events

legend

<pre>Soar> run 1 ----- 76: O: O25 create-referent(cop) ----- Soar> ms ----- Assertions: s-construct*create-referent*touch-conjunct-symbol s-construct*create-referent ----- Retractions: Soar> p s-construct*create-referent ----- (sp s-construct*create-referent (goal <g> ^operator <o> ^problem-space <p> ^state <s>) (<o> ^name create-referent ^for <obj>) (<p> ^name s-construct) --> (<obj> ^referent <r> + ^referent <r> &) (<r> ^referent-of <obj> + ^type s-model +))</pre>	<p>run command, causing the program to select an operator</p> <p>match-set command, showing asserted SPs (Soar productions about to fire)</p> <p>print command for an SP</p> <p>left-hand side, made up of conditions</p> <p>right-hand side, made up of actions</p>
<pre>: ==>G: G15 state no-change : P: P68 create-operator : S: S15 : O: O24 s-structor16 : ==>G: G16 operator no-change : P: P85 s-construct ----- : S: S15 ----- : O: O25 create-referent(cop) ----- Soar> run 1 ----- Build: chunk-128 Build: chunk-129 Soar> p chunk-128 (sp chunk-128 :chunk (goal <g1> ^operator <o1> ^state <s1>) (<o1> ^name s-structor16 ^type s-model-constructor) (<s1> ^assigners <a1>) (<a1> ^n <n2>) (<n2> ^max <n1>) (<n1> ^head) (-^referent <r*1>) --> (^referent <r1> + ^referent <r1> &) (<r1> ^referent-of + ^type s-model +))</pre>	<p>problem-space stack, showing the current problem space, state, and operator</p> <p>run command, causing the program to build chunks</p> <p>a chunk (new SP)</p>
<pre>Soar> p o25 ----- (O25 ^name create-referent ^for U20) ----- Soar> p u20 (U20 ^left-edge W8 ^right-edge W13 ^bar-level max ^word-id W13 ^category n ^annotation specified ^empty-node E15 ^spec U14 ^zero-head U15 ^head U17)</pre>	<p>print command for an object</p> <p>object identifier</p> <p>attribute/value pair pointing to a subobject</p>

Figure 3: Description of features in the domain

the *execution stack*. The current problem space, in which the program now executes, is at the bottom right. Each problem space contains a state that is modified by operators selected in sequence. Eventually the program will transfer control to the next higher problem space, or *superspace*, and the current space will be garbage collected. The state in the superspace is the *superstate*.

The programmer runs another cycle. This time the interpreter builds two *chunks*. These are SPs that the Soar interpreter's learning mechanism builds at run-time and adds to the program. The interpreter builds chunks whenever an SP modifies the state in a problem space other than the current one. In this case, the SPs that were about to fire in the top screen now do fire, modifying the current state. The current state (s15) is also the superstate (s15), causing the language interpreter to build a chunk. The chunk caches the state modification, for future use in the higher problem space (Laird et al., 1986).

The programmer then prints out one of the chunks (chunk-128). Chunks are just SPs, and look the same.

The bottom screen shows two linked *objects*. The first object (o25) is the operator from the top screen. The operator has an attribute (^for), with a value that is another identifier (u20). The attribute *points to* a subobject. Printing the operator lets the programmer print this subobject, which is the operator's argument.

Figure 4 contains a glossary of commonly-used terms in the language.

2.3. The studied scrolling events

The five scrolling events we studied in detail are shown in Figure 5. All the output generated during the modeled interval is shrunk to fit on the page. The windows overlaid on the buffer roughly delineate the information on display at the time of each scrolling event.

Each event has two associated screens of information. The *encoding screen* is the one during which the programmer notices the feature that she scrolls to later. The protocol doesn't always specify exactly what the scrolled-to feature is, but strongly limits the possibilities. The boundaries of the encoding screen are determined by where the programmer stops scrolling. Her comments and commands after that narrow the field at least to the output of one particular command.

The *recall screen* is the one at which the programmer is looking when she decides to scroll to the encoding screen.

Scrolling events 1, 3, and 5 follow one basic pattern. The encoding screen is visible for some amount of time, and then becomes hidden for 30 seconds or more. The length of time that the feature is hidden, and the programmer's intervening activity, imply that the scrolling event is based on knowledge recalled from LTM. The programmer encodes some memory during the encoding screen, and recalls it during the recall episode, presumably prompted by cues from the recall screen.

Scrolling events 2 and 4 are different, in that the encoding screen is hidden briefly, for less than 15 seconds. The programmer could have kept the scrolled-to feature in WM while it was hidden (though we retain the

Term	Definition
action	an element of the <i>right-hand side</i> ; modifies data structures (e.g., <i>states</i>)
asserted SP	an <i>SP</i> in the <i>match-set</i>
chunk	an <i>SP</i> that Soar creates on the fly, when an <i>action</i> modifies a <i>state</i> in an older <i>problem space</i>
condition	an element of the <i>left-hand side</i> ; tests data structures (e.g., <i>states</i>)
current space	the newest <i>problem space</i> , occurring rightmost in a <i>problem-space stack</i> on the display
identifier	unique symbol, generated at run-time, that identifies an <i>object</i>
left-hand side	the set of <i>conditions</i> , or "if" part, of an <i>SP</i> ; if this matches, the <i>SP</i> enters the <i>match-set</i>
match set	the set of <i>SPs</i> whose <i>left-hand sides</i> now match and that are about to fire
object	the basic data structure, created and modified by <i>SPs</i> , consisting of an identifier and a set of attribute/value pairs
operator	functional object that modifies the <i>state</i> and objects attached to it; these modifications are carried out by <i>SPs</i>
problem space	a run-time execution context, in which a <i>state</i> is modified by a sequence of <i>operators</i>
problem-space stack	the run-time execution stack; the program transfers control from the <i>current space</i> to the <i>superspace</i>
right-hand side	the set of <i>actions</i> (the "then" part) of an <i>SP</i>
SP	Soar production, an if-then rule with <i>conditions</i> and <i>actions</i>
state	main data structure in a <i>problem space</i> ; contains other <i>objects</i>
superspace	the next older space than the <i>current space</i> , to which the current space returns control

Figure 4: Glossary of terms in the domain

terms "encoding" and "recall" to describe the screens and episodes). Scroll 2 reverses scroll 1 after a few seconds, taking the programmer back to what is then the bottom of the buffer. Scroll 4 (bottom) moves the window three lines, to get to a symbol that was forced off the top of the window by the output of the previous command.

In the following five subsections, we examine the programmer's behavior during these five events in detail.

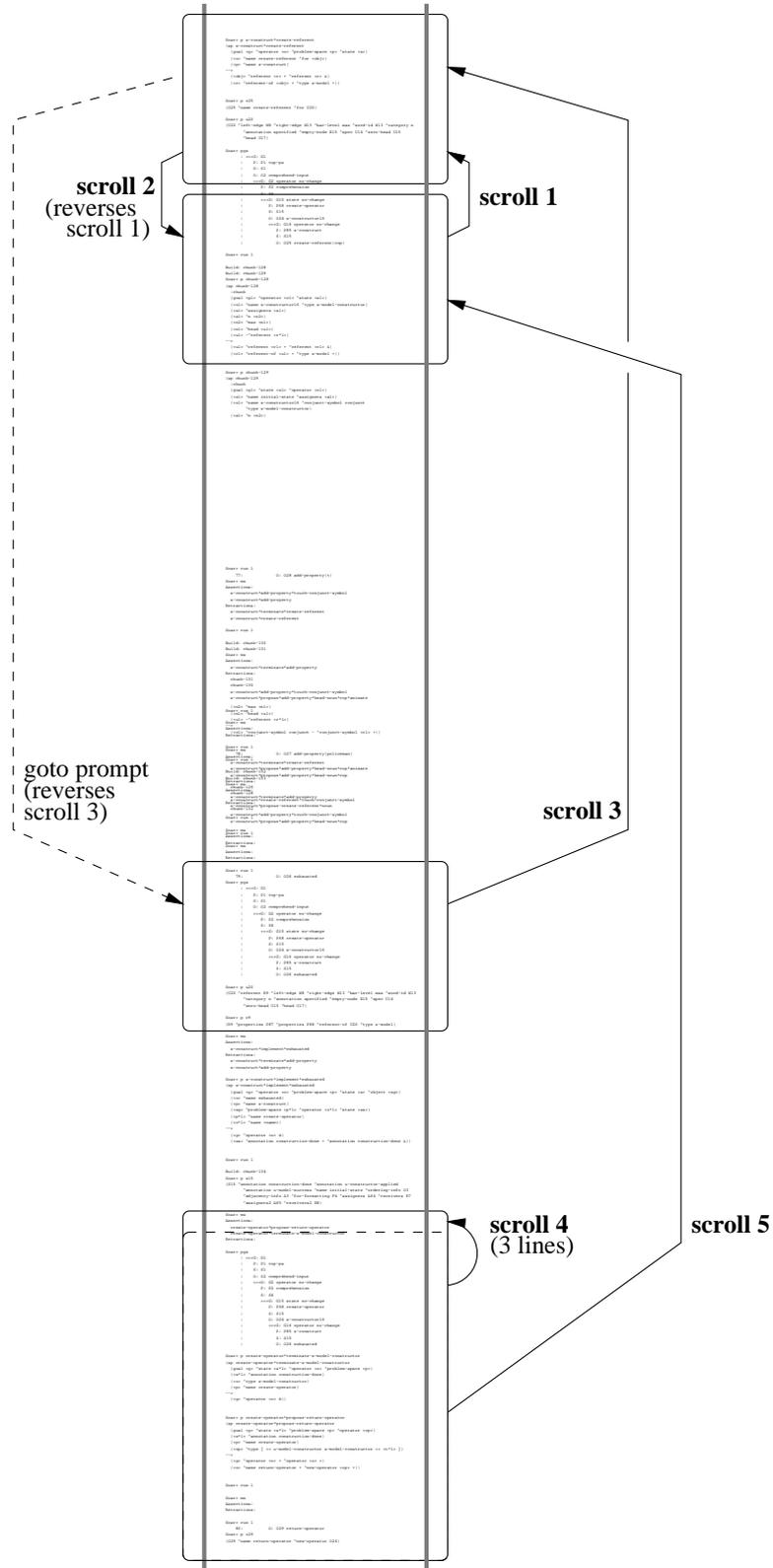


Figure 5: Overview of scrolling events

Each section opens with a brief overview of the event, then describes the details of the encoding and recall episodes, and closes with a summary of the recollection that the programmer appears to have had.

The notation used in the figures showing the programmer's behavior is as follows (see Figure 6). The verbal protocol is in Roman font, with elisions shown by ellipsis (*[...]*). Commands issued by the programmer are in `courier` font. The temporal sequence of protocol excerpts is marked by arrows. Times (e.g., t266) are second markers from the protocol timecourse.¹

Display excerpts are marked by boxes. The line numbers in the upper-right corner of each box (e.g., "display lines d1050 to d1065"), are the line numbers we added to the programmer's interpreter buffer.

Dashed lines and shading connect protocol elements to their referents on the display. Our descriptive comments are in *italics*.

2.3.1. Scrolling event 1

Overview: The programmer scrolls back to and re-examines an SP she printed out earlier (Figure 6). The SP (s-construct*create-referent, abbreviated create-referent) has been hidden for 58 seconds.

Encoding episode: The programmer identifies the SP that she is thinking about ("the production that's in the match set now", t266). She goes on to examine at least one of its conditions ("create-referent for"), and comes to some understanding of the SP's function ("right, this is the one that's actually going to create the referent").

Roughly a minute and a half later (t371), the programmer wants to understand some chunks that the program built ("let's see what the chunks are doing"). She issues a command to print one of the new chunks ("p chunk-128"). The output from the print command displaces the SP she was looking at.

Recall episode: After examining chunk-128, the programmer sees that it contains no condition specifying a problem space ("oh, it's not testing for a problem space"). This answers a question she had earlier ("that's why, ok"), but introduces a new question about how the chunk came to be built. At first she seems to have an explanation ("so it must have just changed it on the superstate? oh these are shared states, i see"), but then becomes dissatisfied with it ("no i don't see; what built that?"). She scrolls back to the top screen and examines the SP, beginning with its conditions ("this said, if you're in the s-construct problem space", t435).

Recollection summary: The recollection that triggers the scrolling event is related to a new chunk ("chunk-128"), and to a hidden SP. The cause of the chunk may involve the superstate, or that the current state and the superstate are the same (shared by the two problem spaces).

¹Protocol timecourse, model trace, and contents of the programmer's interpreter buffer are aligned in Appendix E.

programmer sees feature

```
(display lines d1050 to d1065)
Soar> run 1
76:          0: O25 create-referent(cop)
Soar> ms
Assertions:
  s-construct*create-referent*touch-conjunct-symbol
  s-construct*create-referent
Retractions:

Soar> p s-construct*create-referent
(sp s-construct*create-referent
 (goal <g> ^operator <o> ^problem-space <p> ^state <s>)
 (<o> ^name create-referent ^for <obj>)
 (<p> ^name s-construct)-
-->
 (<obj> ^referent <r> + ^referent <r> &)
 (<r> ^referent-of <obj> + ^type s-model +))
```

programmer's protocol

t266:
ok the production that's in the match set now is going to actually create, create referent for, right, this is the one that's going to actually create the referent
[...]

t435:
this said, if you're in the s-construct problem space ...

t371:
let's see what the chunks are doing
print chunk-128 (t373)

[...]
oh, it's not testing for a problem space, that's why, ok
[...]
so it must have just changed it on the superstate?
oh these are shared states, i see;
no, i don't see, what built that?

scroll event 1:
window up
t431 (+58 seconds)

print command causes top screen to scroll off

programmer recalls feature

```
(display lines d1084 to d1109)
:      ==>G: G15 state no-change
:      P: P68 create-operator
:      S: S15-----
:      O: O24 s-structor16
:      ==>G: G16 operator no-change
:      P: P85 s-construct
:      S: S15-----
:      O: O25 create-referent(cop)

Soar> run 1
Build: chunk-128
Build: chunk-129
Soar> p chunk-128
(sp chunk-128
 :chunk
 (goal <g1> ^operator <o1> ^state <s1>)
 (<o1> ^name s-structor16 ^type s-model-constructor)
 (<s1> ^assigners <a1>)
 (<a1> ^n <n2>)
 (<n2> ^max <n1>)
 (<n1> ^head <ul>)
 (<ul> -^referent <r*1>)
-->
 (<ul> ^referent <r1> + ^referent <r1> &)
 (<r1> ^referent-of <ul> + ^type s-model +))
```

Figure 6: Scrolling event 1, programmer's behavior

2.3.2. Scrolling event 2

Overview: The second scrolling event (Figure 7) reverses the first, a few seconds after the first occurs. The programmer returns to the bottom screen, with information in mind from the top screen.

Encoding episode: The encoding episode for the second event (t422) is the recall episode from the first. The programmer sees one state ("s15") shared between two problem spaces, and a new chunk built by the program. The first scrolling event takes her to the top screen, to examine the SP (create-referent) that just fired.

Recall episode: At the top screen, the programmer finds out which problem space the SP fires in ("this said, if you're in the s-construct problem space", t435). She also finds out the SP's actions ("you slap that attribute, on the object"). This takes only a few seconds, and 14 seconds after scrolling to the top screen (t431) the programmer scrolls back to the bottom one (t445).

Back at the bottom screen, the programmer matches it against the information from the top screen. The current space, which she determines from the stack, is the same as the SP's firing space ("so i'm in the s-construct problem space"). Because the state in the current space ("s15") is also the state in the superspace (to the left and up), the firing resulted in a chunk.

Recollection summary: The question of why the program built a chunk seems to lead the programmer first to the top screen and then back to the bottom screen. The shared state is part of her explanation, both before leaving the bottom screen and after returning to it.

2.3.3. Scrolling event 3

Overview: The programmer scrolls back to an object she printed out previously, to retrieve its identifier (Figure 8). The identifier is a required parameter for printing an object. Once she knows what the identifier is, she returns to the prompt and prints a fresh copy of the object.

Encoding episode: The programmer sees the object of interest for the first time ("what is u20", t250). She prints the object, using its identifier ("u20"), and recognizes it as an *utterance model* ("the for argument is the profile in the u-model"). Sometime later (t306), with the object still on display, the programmer notes that some attributes are missing from the object ("this is just the bare node, it doesn't have any of the properties"). The screen containing the utterance model becomes hidden roughly a minute later (t373).

Recall episode: The programmer wants to know the current status of the program ("ok where am i", t638), and prints the problem-space stack. She recalls that this status includes a particular object ("s15 now has an utterance model object"). This utterance model object is the object of interest from the encoding episode. To print this object, she needs its identifier.

Identifiers are alphanumeric symbols generated by Soar at runtime. Their only semantic content is the alphabetic component, which is a single letter. This is the first letter of the variable designating the

feature briefly hidden

```
(display lines d1050 to d1065)
Soar> run 1
76:          O: 025 create-referent(cop)
Soar> ms
Assertions:
  s-construct*create-referent*touch-conjunct-symbol
  s-construct*create-referent
Retractions:

Soar> p s-construct*create-referent
(sp s-construct*create-referent
(goal <g> ^operator <o> ^problem-space <p> ^state <s>)
(<o> ^name create-referent ^for <obj>)
(<p> ^name s-construct)
-->
(<obj> ^referent <r> + ^referent <r> &)
(<r> ^referent-of <obj> + ^type s-model +))
```

programmer's protocol

t435:
this said, if you're
in the s-construct
problem space,
you slap,
that attribute
on the object
scroll event 2:
window down
(t445)

programmer sees feature

```
(display lines d1084 to d1109)
:      ==>G: G15 state no-change
:      P: P68 create-operator
:      S: S15-----
:      O: 024 s-constructor16
:      ==>G: G16 operator no-change
:      P: P85 s-construct
:      S: S15-----
:      O: 025 create-referent(cop)

Soar> run 1

Build: chunk-128
Build: chunk-129
Soar> p chunk-128
(sp chunk-128
:chunk
(goal <g1> ^operator <o1> ^state <s1>)
(<o1> ^name s-constructor16 ^type s-model-constructor)
(<s1> ^assigners <a1>)
(<a1> ^n <n2>)
(<n2> ^max <n1>)
(<n1> ^head <ul>)
(<ul> ^referent <r*1>)
-->
(<ul> ^referent <r1> + ^referent <r1> &)
(<r1> ^referent-of <ul> + ^type s-model +))
```

t422:
oh these are
shared states, i see;
no, i don't see, what
built that?
scroll event 1
(t431)

so i'm in the
s-construct problem
space, i'm going to
slap that thing on
there, and lo and
behold,
i get this chunk,
because they
share the state
t458

Figure 7: Scrolling event 2, programmer's behavior

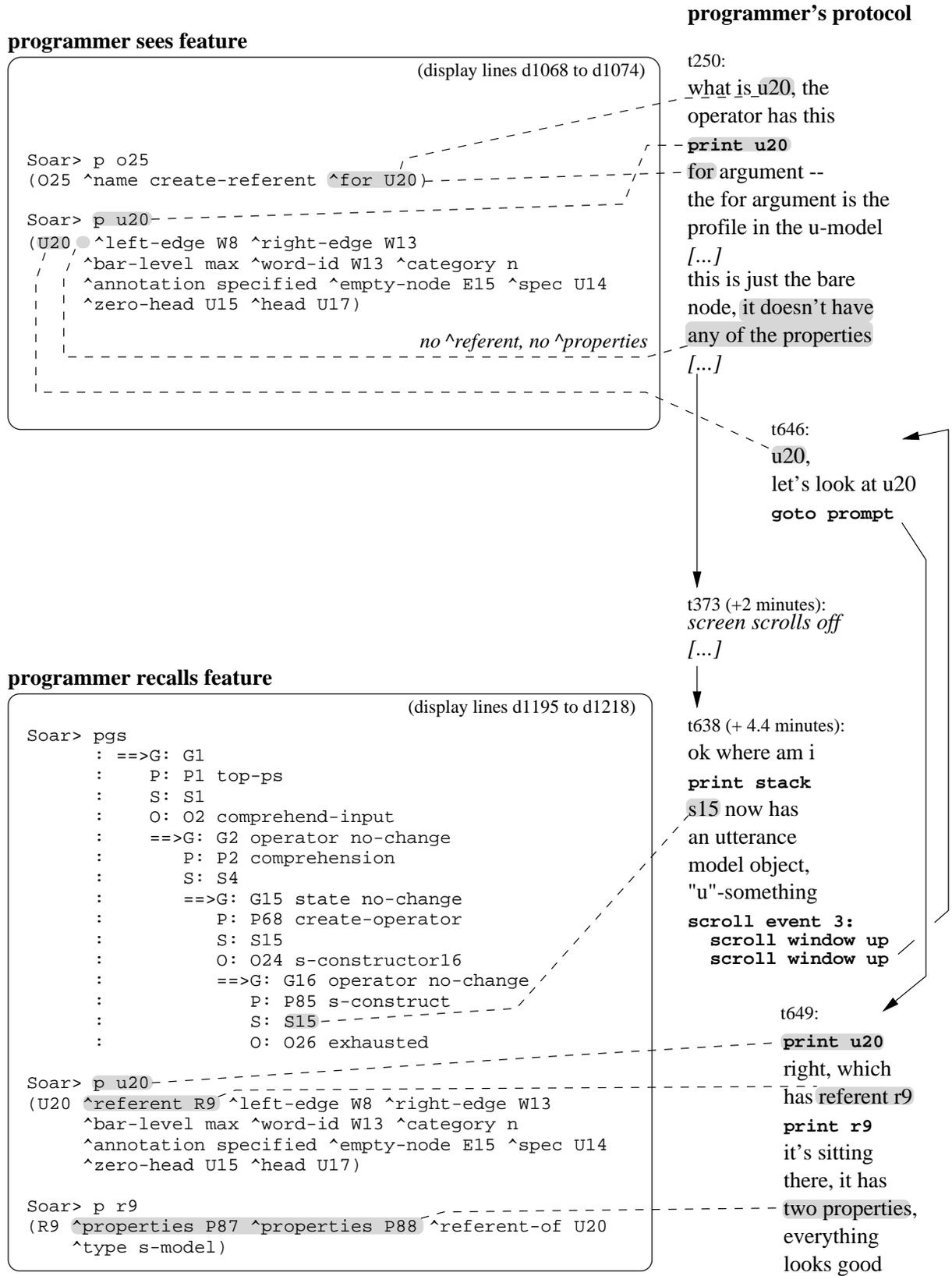


Figure 8: Scrolling event 3, programmer's behavior

identified object, in the SP that created the object. Thus an identifier has the first letter of a meaningful name.

Either from memory or by inference from its name, the programmer determines that the identifier of the utterance model begins with "u" ("u'-something"). She then scrolls back to the previous copy of the utterance model.

Back at the top screen, the programmer sees the identifier, ("u20", t646), and then immediately returns to the prompt to issue a print command ("p u20", t649). The fresh copy of the utterance model has the attributes that were missing from the previous copy (the referent attribute points to a subobject that contains the properties attribute).

Recollection summary: The recollection behind this scrolling event concerns the first displayed copy of the utterance model. To complete the print command in the recall episode, the programmer decides to find the identifier from this previous copy. An alternative method for finding the identifier would have been to use visible information from the problem-space stack on the display. She could have found the utterance model's identifier with a sequence of three print commands, starting with a command to print the current state (s15). With this other method available, she chose scrolling instead. This suggests that despite the length of time that the previous utterance model was hidden, she nonetheless has a robust and rapidly-accessible memory for it.

2.3.4. Scrolling event 4

Overview: The programmer scrolls to a symbol that was only recently displaced, and is only a few lines off the top of the screen (Figure 9). When the symbol reappears, the programmer copies it into a print command.

Encoding episode: The programmer sees two SPs that are about to fire (appearing in the top screen as asserted SPs). The first of these ("propose return operator", t745) suggests that the program is approaching behavior that the programmer is anticipating ("i think we're getting close to the right place"). The programmer eventually prints out and examines both SPs, but for now chooses the second one ("terminate s-model-constructor, see what that's doing").

Recall episode: Printing the body of terminate-s-model-constructor forces the name of the other SP to scroll off the top of the display. When the programmer has finished with terminate-s-model-constructor ("ok fine that looked for the construction done", t767), she scrolls back three lines to propose-return-operator. When it reappears, she copies it into a print command ("now what is this doing", t773).

Recollection summary: The programmer seems to keep in mind that there was another asserted SP, while she comprehends terminate-s-model-constructor. In the recall episode, she hasn't finished describing terminate-s-model-constructor by the time she begins scrolling the name of the second SP into view. She seems to have planned to examine both asserted SPs.

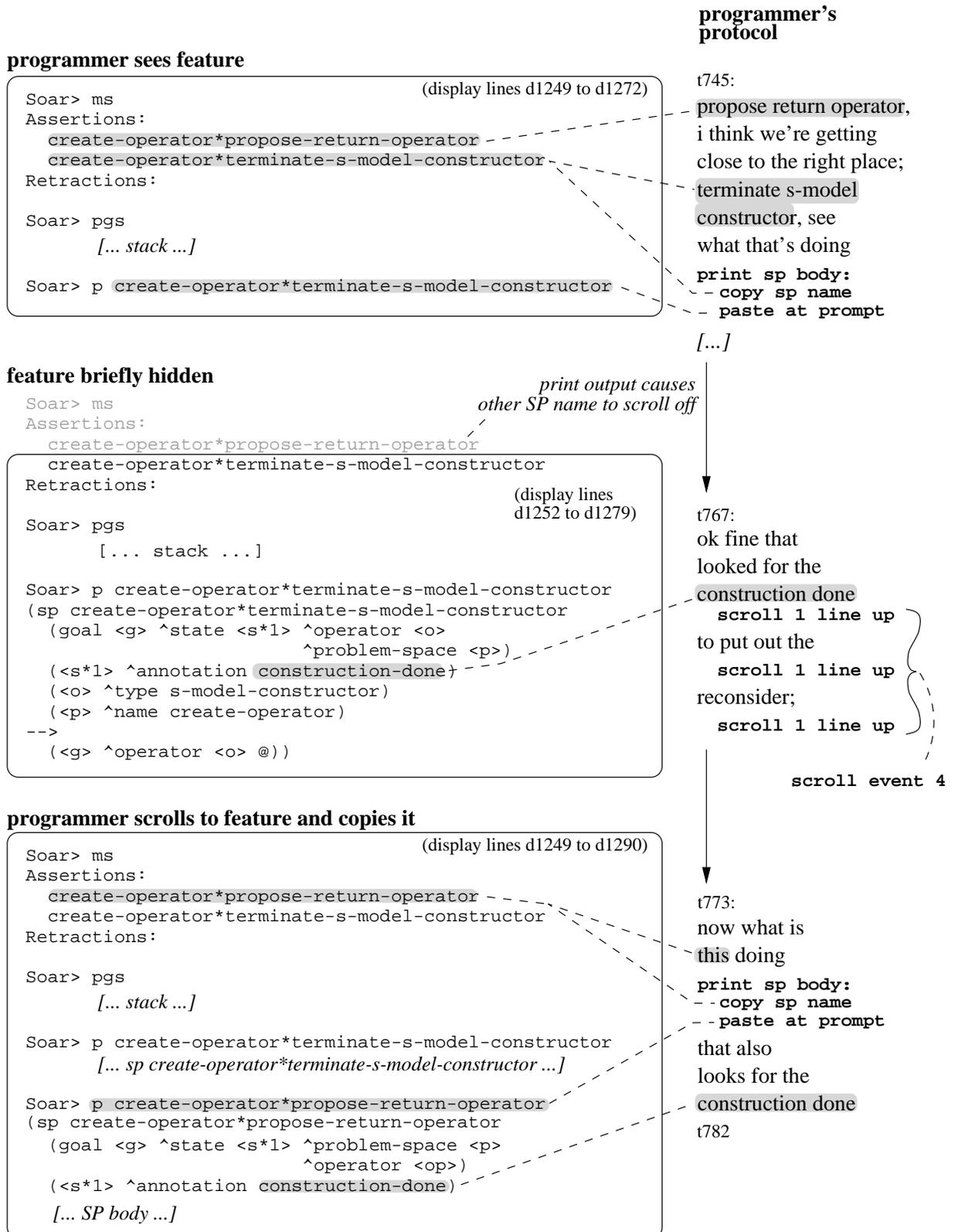


Figure 9: Scrolling event 4, programmer's behavior

2.3.5. Scrolling event 5

Overview: The programmer scrolls back to chunks she printed out several minutes earlier, on a hunch about their conditions (Figure 10). The hunch seems to be triggered by information on display during the recall episode.

Encoding episode: The programmer sees a particular condition as part of a chunk she is examining ("ok, this chunk is testing for s-structor16", t379). The screen containing this chunk becomes hidden three minutes later (t564).

Recall episode: The programmer has just printed an operator, and examines the operator's argument ("ok it says new-operator o24", t821). This argument is itself an operator, selected in one of the older spaces in the problem-space stack ("which is in fact s-structor16"). The identifier o24 links the argument and the selected operator.

Several seconds after seeing s-structor16, the programmer recalls something about the conditions in the chunk from the encoding episode ("i think all those chunks i built test for...", t843). The recollection seems to be either about condition s-structor16 ("six, uh, 's' whatever it is"), or about the operator type to which s-structor16 belongs ("i think they test for the s-structor"). This "s-structor" reference may be to a condition on the operator type ("^type s-model-structor"), which appears in both SPs on the bottom screen.

Back at the encoding screen (t856), the programmer looks at and recognizes the s-structor16 condition ("let's see shall we, yeah, s-structor16, there it is").

Recollection summary: The recollection behind this scrolling event concerns a symbol that the programmer had seen several minutes earlier. In the recall episode, this symbol (s-structor16) appears in a different but semantically-related context (as an operator in the problem-space stack).

2.4. Summary: memories that lead to scrolling

In scrolling event 1, the recollection that triggers the scrolling event is related to a chunk ("chunk-128"), a modified state ("s15"), and an SP that modified the state. The chunk and the state are on the display, and the SP is hidden. The question of why the program built the chunk seems to lead the programmer to scroll back to the SP.

In scrolling event 3, the programmer recalls that an object she is thinking about now was on the display previously. She scrolls to the old copy to retrieve its identifier ("u20"), and uses the identifier to print a new copy.

In scrolling event 5, the programmer recalls something about a symbol ("s-structor16") she saw previously as a chunk condition. During the recall episode, the same symbol is visible as a selected operator.

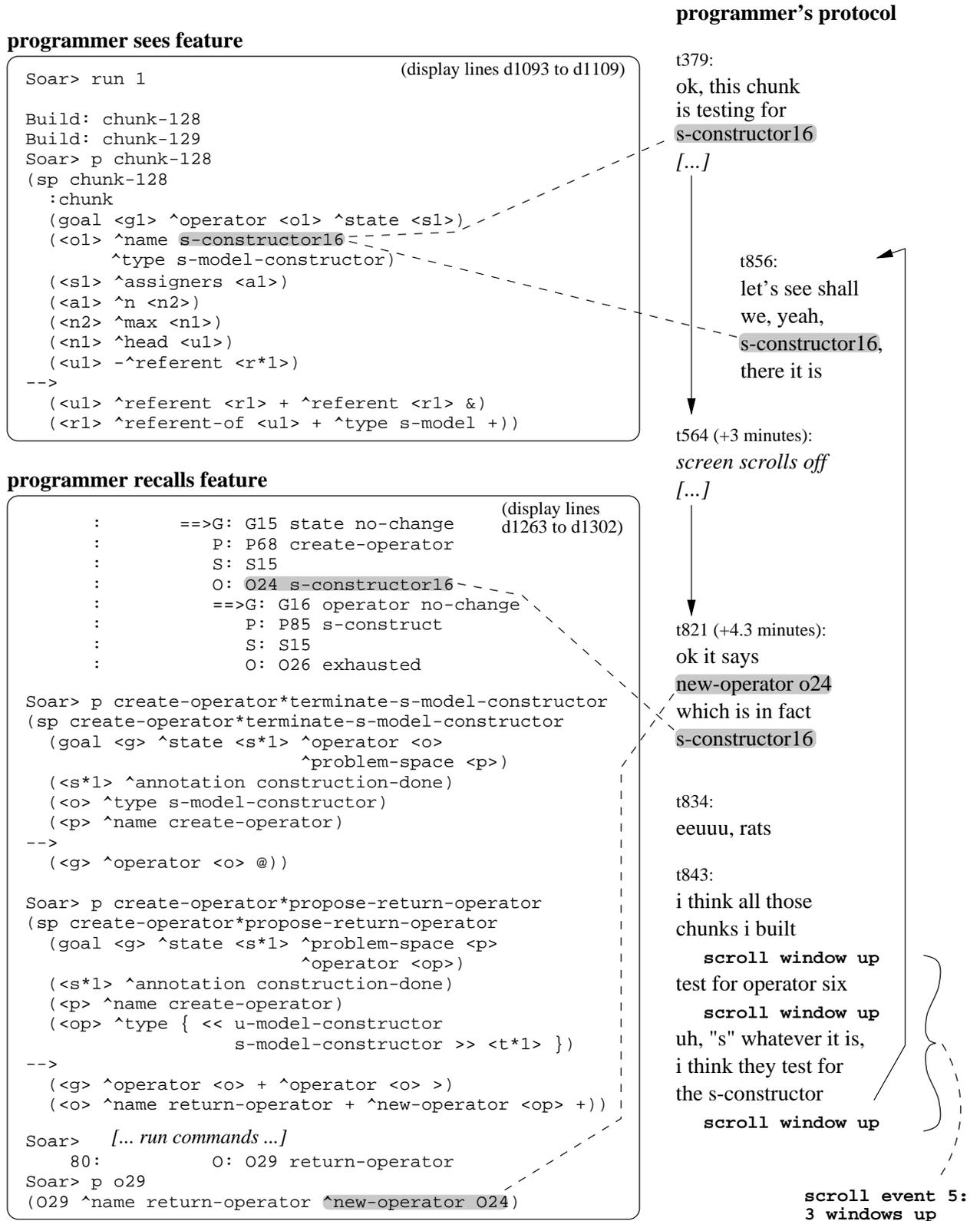


Figure 10: Scrolling event 5, programmer's behavior

In all three events, the programmer acts on some memory for a feature generated earlier in the session but too long ago (more than 30 seconds) for her to have maintained it in WM. The memory seems to be triggered by some feature or set of features currently on display, since the protocol suggests that these features are what she is thinking about at recall time. What is the nature of her memories for hidden features, and how are these memories activated at recall time?

In scrolling events 2 and 4, the programmer scrolls to a nearby feature that has been hidden for less than 15 seconds. What information could she have maintained in WM that would have caused her to scroll?

In the next chapter, we introduce a model that provides one set of answers to these questions. In Chapter 4 we take up these questions in detail, as we describe the model emulating the behavior of the programmer.

Chapter 3

The model

The model has three components that interact to account for the behavior described in the previous chapter: 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 key function of learning through performance, encoding information about the programming session into long-term memory (LTM).

Mechanisms provide key cognitive functions that the architecture leaves unspecified. We organize our discussion of these mechanisms in terms of model inputs, internal cognitive functions, and model outputs, though the flow of control and data is more interactive than this organization suggests. There are mechanisms to retrieve knowledge from the display (Section 3.4), to retrieve information from LTM (Section 3.5), to deliberate by selecting goals and subgoals (Section 3.6), and to change the display (Section 3.7).

The summary (Section 3.8) presents a table of brief descriptions of the model's knowledge and mechanisms. This is in preparation for Chapter 4, which traces the model's behavior and refers to the mechanisms as they come into play. Appendix A (p. 111) traces through a complete configuration diagram of the model.

3.1. Overview of the model

In the behavior we model, 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*. (The term *goal* often implicitly refers to the object being comprehended.) 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 (Section 3.6.1).

Having selected a comprehension goal, the model proceeds by trying to retrieve knowledge relevant to that goal. Retrieved knowledge accumulates in working memory (WM), which has a limited persistence,

holding only the knowledge retrieved during the current goal and the immediately previous goal (Section 3.5.4).

WM persistence is long enough that features can remain in WM even after they become hidden on the display. The model detects when a feature in WM becomes hidden, allowing the model to scroll back to a feature that is hidden but was seen recently (Section 3.4.1.1).

The model uses *subgoals* to retrieve relevant information about its current goal. This information comes from the display and from LTM (Figure 11). There are three kinds of information-retrieval subgoals. *Fixations* copy features from the display into WM, together with information that the source of the feature is the display, and information about the time of the fixation. The model automatically encodes episodic memories for what it saw, as a result of these features entering WM (Section 3.4.2). *Images* emulate fixations by copying features from LTM into WM (Section 3.5.2). The model can imagine important, common features in the domain, features that an experienced programmer is likely to be familiar with. These images can trigger episodic memories for those features, which make the model aware that it has seen the feature before. *Probes* mimic comprehension goals, to trigger recollections from LTM about an object. A probe selected in the context of a goal imports facts about the probe into the context of that goal (Section 3.5.1).

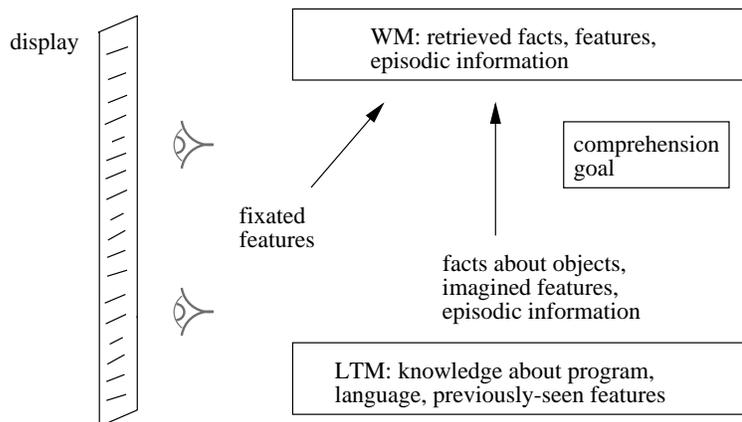


Figure 11: The model comprehends an object by retrieving information

Interleaved with comprehension goals are occasional *command goals*, which represent interpreter or editor commands for changing the information on the display (Figure 12). These goals arise from 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.

The model's commands are interpreted by a display emulator (Section 3.7.2). The emulator responds to a

command by updating a representation of the screen. The model makes contact with the emulated display by means of *attend* subgoals (Section 3.4.1).

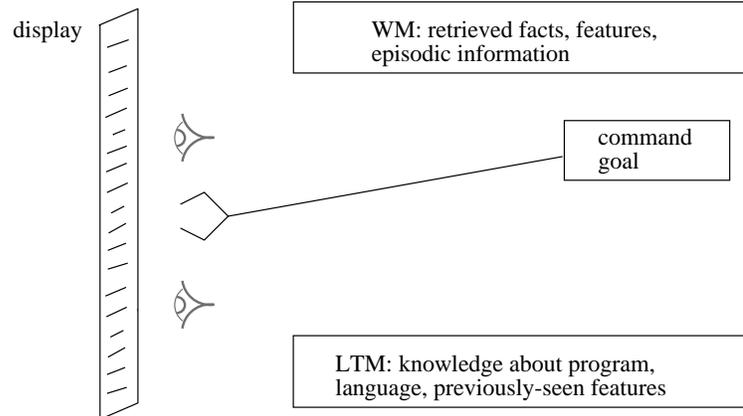


Figure 12: The model issues commands to generate new output and scroll

3.2. Knowledge: expert, external, and episodic

The model contains 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 (John and Kieras, 1994, Card et al., 1983).

The model's expertise includes facts about objects in the programmer's task environment. Figure 13 shows a small subset of these facts, some of which are general to the programmer's language (Soar), and some specific to the program itself. The model also knows about objects to comprehend and features to look at. Objects and features are often the same. For example, a chunk (at the left of Figure 13) is a feature that the model fixates on in some situations, and an object that the model proposes comprehending when it enters WM. For some features, the model notices when they are absent. For example, a problem-space condition gives important information about a chunk. The model fixates this information if it is present and fixates its absence if not. Finally, the model knows how to manipulate the display to change what external knowledge is visible.

External knowledge consists of visible and hidden features. Hidden features can be made visible by navigation, in this case by scrolling. External and internal knowledge interact to extend a problem-solver's

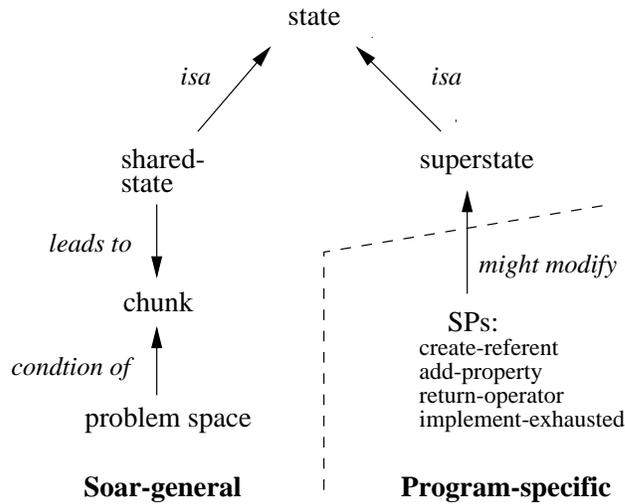


Figure 13: Examples of expert knowledge in the model

effective memory (Larkin, 1989, Davies, 1996, Green et al., 1987). However, computational models that address the display as external knowledge often treat only the immediately-available component — for example, characters in a video game (Bauer and John, 1995) — or treat only those hidden elements that are a stable part of an interface — for example, hidden menu items (Rieman et al., 1996, Howes and Young, 1996b, Kitajima and Polson, 1995, Howes, 1994). In real programming situations, as in our study, external knowledge can be generated dynamically, and only a fraction of it is visible at a given time.

The model's episodic knowledge consists primarily of memories for fixation events (Section 3.4.2.2). These memories let the model recall seeing features on the display. Each memory associates a feature in WM with a timestamp that symbolizes the fixation event. Because this timestamp is generated by the model dynamically, the model cannot know it ahead of time, and so must learn it when it arises. This contrasts with 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), which begin with a basic competence and improve upon it. The model has to remember dynamically-generated information, making learning an essential part of its competence.

The model also learns episodic knowledge about regions of output generated by the model's commands (Section 3.4.1). This knowledge lets the model distinguish between novel regions of output and regions it has examined before. The model uses this information heuristically to prefer newly-generated features (Section 3.6.2).

3.3. Soar as the underlying architecture

The model's underlying cognitive architecture is Soar (Rosenbloom et al., 1992, 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. Rules in Soar's LTM propose the model's comprehension and command goals, as well as its information-retrieval subgoals. Goals and subgoals are both represented as cognitive operators, selected one at a time by the architecture. The learning mechanism encodes episodic knowledge automatically as a consequence of information retrieval (we expand on this in Section 3.3.1, below).

The relationship between goals and subgoals is implemented using Soar's *universal subgoal*ing mechanism (Laird, 1984).² To achieve a comprehension goal, the model tries to learn more about the object to be comprehended. Every goal thus represents a lack of knowledge, which Soar treats as an *impasse*. When Soar encounters an impasse, it creates a new problem-solving context and transfers control to it, suspending the context in which the impasse occurred. The new context affords an opportunity to generate knowledge to resolve the impasse. The model uses this new context to select information-retrieval subgoals. Each of these is an atomic Soar operator that retrieves what information it can and is then replaced by another operator. The model selects subgoals until enough information has been accumulated about the current goal object that it is time to move on to a new one. Soar by itself says little about when a goal is achieved, and the decision about how much information is enough is made by mechanisms specific to the model (Section 3.6.1).

3.3.1. Soar's learning mechanism

Soar's key provision, with respect to our model, is its learning mechanism, because this affects how the model remembers what it has seen. The model inherits three constraints from Soar's learning mechanism: learning in the model is *pervasive*, *passive*, and *recognition*al. Learning is pervasive in that every element that enters the model's WM causes to encode a new rule. Learning is passive in that encoding a side effect of the information entering WM, rather than the result of a deliberate decision.

Learning is recognition

al in that Soar performs little induction on learned rules. When an element enters the model's WM, Soar traces backwards from that element, through the rule that created it, to the elements matched by that rule, and so forth, until it reaches elements that existed when the current goal was selected (Laird et al., 1993, Laird et al., 1986). It then encodes a new rule that associates the new element with these pre-existing elements, finding these pre-existing elements to be the conditions that led to the new element entering WM (Howes and Young, 1996a). Consistent with the encoding specificity principle (Tulving, 1983), the new rule will only fire, and hence retrieve the encoded WM element, in the presence of cues that were present in WM at encoding time.

²The vocabulary we use to describe the model, including the terms "goal" and "subgoal", is non-standard with respect to Soar. This arises from the difficulty of describing a Soar model of someone working on a Soar model. The mapping between model constructs and Soar constructs is described in Appendix B.

For the episodic memories encoded by the model, the cue is the feature itself, and the retrieved element is the time of the fixation event. However, because Soar learns pervasively, it also encodes rules that cache the features it retrieves from the display, and also recodes facts it retrieves from LTM. We introduce these kinds of learning as we describe the related information-retrieval mechanisms (Sections 3.4 and 3.5). Figure 20 (p. 41), in the summary of this chapter, summarizes the different kinds of learned rules.

3.4. Mechanisms to retrieve information from the display

To comprehend an object, the model retrieves features from the display. Before it can do this, it must be aware that there are features to be retrieved. The attention subgoal puts the model in contact with new regions of output on the display (Section 3.4.1). The model must then bring features into WM, which it does with fixate subgoals (Section 3.4.2). The model encodes memories for this feature as a side effect of fixation.

3.4.1. Attending: making contact with the display

The model attends to display changes in the sense of creating a new channel to the environment (Newell, 1990). An *attend subgoal* puts the model in contact with a new region of output, by placing a pointer to the new region in the model's WM. The model's fixation knowledge perceives features by following this pointer. A display pointer remains in WM until the region it points to becomes hidden, at which point the pointer automatically drops out of WM. Thus the model's fixation knowledge has access to all features in all visible regions of the display.

The attend operator also maintains a representation of novelty of regions. This representation is used by heuristics that prefer looking at novel features (Section 3.6.2). Representing novelty requires an episodic memory for previous occurrences. The model learns to recognize each new region it attends to. If a region appears again, the model recognizes it.

The model uses these attended-region memories to compute whether there is a novel region on the display. The newest region on the display is usually novel, but not always. After any scroll command, the newest region on the display is there again, rather than for the first time. After scrolling, the newest region is no more likely than any other to contain the target feature. The novel-region heuristic only applies when the most recent command generated new output.

3.4.1.1. Noticing when features become hidden

The model also uses attended-region memories to notice when features in WM become hidden. Each feature in WM is tagged with the region it comes from. (This information is also necessary for the model to prefer features from novel regions.) However, an attended-region memory itself is dynamic, in that it falls out of WM as soon as the attended region disappears from the display. This lets the model recognize when features in WM become hidden. The hidden-feature information persists in WM as long as the feature itself persists.

Figure 14 shows an example of how hidden-feature information can be used as the basis for scrolling decisions based entirely on WM. The model fixates on a feature during goal 1. The model later selects goal 2, and selects a command in service of goal 2 that causes the feature to become hidden. After the command, the goal-selection mechanism re-selects goal 2 (Section 3.6.1). The feature is now hidden, and the model automatically adds this information to WM. Thus the model can work on goal 2 and continue to be aware of a recently-fixated but hidden feature. This awareness can be the basis for a scrolling decision (Section 4.3).

The proposal rules for attend operators are indexed in Section C.1, p. 124.

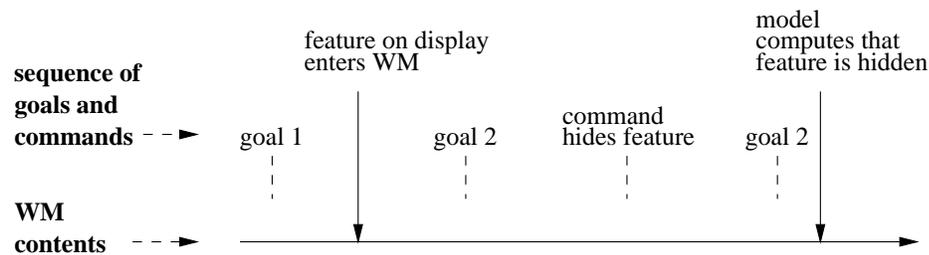


Figure 14: Persistence in WM of a feature that becomes hidden

3.4.2. Fixating: copying features from display to WM

The model sees a feature on the display by selecting a *fixation subgoal* corresponding to that feature. The underlying architecture distinguishes between subgoal *proposals* and subgoal *selections*. The model uses this distinction to separate *perception* and *heeding*.³ In perception, the external representation of a feature becomes represented inside the model. In heeding, the internal representation is deposited in WM.

Proposal rules, stored in LTM, perceive features of the display, and also respond to the contents of WM and the current goal (Figure 15). Proposal rules perceive features through a pointer to the emulated display (Section 3.4.1), and fire when information in WM and possibly the goal say the features are relevant. (The goal-dependence of fixation knowledge is discussed in Section 3.6.3.) When a proposal rule fires, it proposes a fixate subgoal. This subgoal is an internal representation of the perceived feature, but the feature has not yet been added to WM.

A feature enters WM when the model selects a fixate subgoal corresponding to that feature (Figure 16).

³We take the term "heeding" from Ericsson and Simon (1992). In the Soar theory of cognition (Newell, 1990), the term for "heeding" is "encoding", but we use "encoding" here to refer to the storage of memories in LTM.

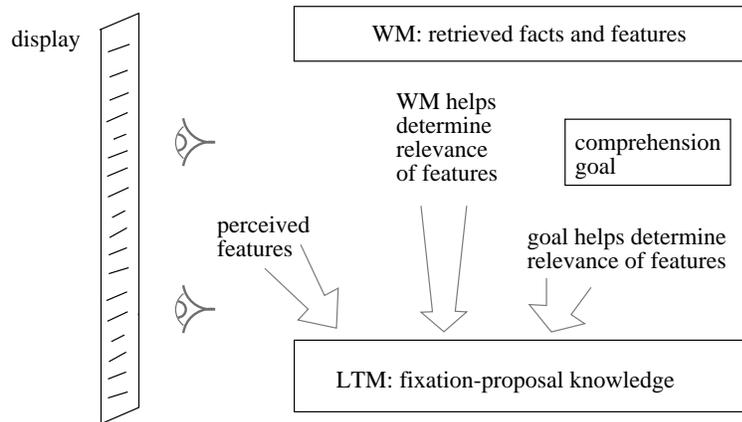


Figure 15: The model uses display, WM, and goal to propose fixation subgoals

When the subgoal is selected, the model deposits the internal representation of the feature into WM.⁴

As a result of fixation, the model encodes two kinds of information: the feature itself, and episodic information about seeing the feature. These memories are described in the next section.

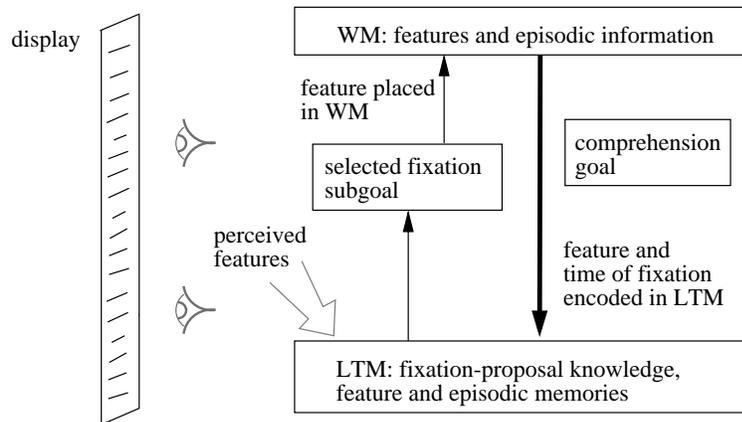


Figure 16: The model selects a fixation subgoal and encodes memories

⁴The model's internal and external representations of features are the same, as a matter of implementation tractability. We mostly ignore the distinction, and speak simply of "features".

3.4.2.1. Encoding feature memories

A *feature memory* caches a feature, allowing the model to heed it more quickly in the future. It is represented as a new rule that associates the feature's internal representation with its external representation and internal cues. These internal cues consist of the elements in WM that caused the model to propose fixating the feature in the first place, and also the goal that was selected then. If the feature appears on the display again, with the proposal-causing elements in WM and the same goal selected, the feature memory will simply retrieve the (internal representation of the) feature to WM. The model will heed the feature without having to select the fixate subgoal.

This preemption of fixation subgoals through learning plays a key role in the model's ability to comprehend objects. Objects on the display (e.g., left-hand sides of SPs) often have sub-features (e.g., conditions). When the model tries to comprehend a left-hand side, it knows to look at conditions. The model may not get through all conditions before it selects a new goal (Section 3.6.1). However, the conditions it did get through are cached in feature memories. These will activate immediately if the model selects left-hand sides again, letting the model fixate new conditions. Feature memories thus lead to *incremental comprehension* of the display (Rieman et al., 1996; see Section 6.6 for a discussion of similar mechanisms in their model and ours.)

3.4.2.2. Encoding episodic memories

An *episodic memory* caches the event of the model fixating a feature. Once the fixation subgoal has deposited a feature in WM, the model adds a new element to WM representing the time of the fixation event. Every comprehension goal receives a unique symbol, or timestamp.⁵ Once a fixated feature appears in WM, the model tags it with the current timestamp.

Tagging a feature with a timestamp causes the model to encode an *episodic memory* for the feature. This memory associates the feature (in WM) with the time it was seen. Later, if the model imagines this feature, the episodic memory will recognize the feature and retrieve the timestamp. The model compares the retrieved timestamp to that of the current goal. If they are different, then the model knows that it fixated the feature at a previous time.

A guide to the implementation of the fixate and imagine mechanisms appears in Section B.4 (p. 119).

⁵These symbols are timestamps in the sense that they represent what there is to the model's representation of time, which is very lean. The only operation that the model can carry out on these timestamps is equality testing, which allows it to tell the difference between past and present.

3.5. Mechanisms to retrieve information from LTM

As described in the previous section, the model automatically learns rules that recognize features. This knowledge is accessible only in given WM contexts. A feature memory is accessible when the feature is actually on the display and the model has selected the goal that was current when the model saw the feature. An episodic memory for a feature is accessible only when the model has that feature in its mind's eye in WM. The model searches its LTM for both kinds of memories by reconstructing WM contexts that the memories then recognize. Just as with people, the model knows something when it sees it, but has to work to recall it.

To gather new information about one object, the model can probe for knowledge associated with other objects (Section 3.5.1). Probing can activate feature memories and also facts, which represent display-independent expert knowledge about objects. To search for episodic memories about features it might have seen, the model can imagine familiar features in its mind's eye (Section 3.5.2). Probing and imagining together in principle let the model access all its knowledge in any context (Section 3.5.3).

Retrieving information from LTM is necessary because the model has limited WM persistence (Section 3.5.4). Without some such limit, LTM would be unnecessary, because the model could simply maintain information in WM until it becomes useful.

3.5.1. Probing: retrieving information about a related object

Like fixation, probing occurs through subgoals selected one at a time. A probe is the mock selection of one goal while another goal is actually selected. Much of the model's expert knowledge consists of facts about objects that it recalls when those objects are selected as goals. Probing serves to import knowledge associated with one goal (the probe) into the WM context of another (the current goal).

Probes are generated from objects in WM, by heuristic knowledge in LTM about what objects in the programmer's language are important and could retrieve knowledge relevant to the current goal (Figure 17). For example, the model knows that the SP (Soar production) is a primary functional unit in the language, and that knowledge about a specific SP in the program could be relevant to any goal. When a specific SP appears in WM, from the display or from LTM, the model automatically proposes it as a probe. The rules representing probe-generation knowledge are indexed under "probe proposals" in Section C.1, p. 124.

In addition to recalling facts from the model's pool of expert knowledge, probing also activates feature memories. In response to a probe, the model recalls visible features that it examined when the probe was a goal. The effect of a feature memory is to retrieve a feature much more quickly than if the model had not seen the feature before. Instead of having to deliberately fixate on the feature, the model recognizes it and brings it into WM automatically.

Because probing retrieves elements to WM, the model encodes new rules (Figure 18). Rules encoded from

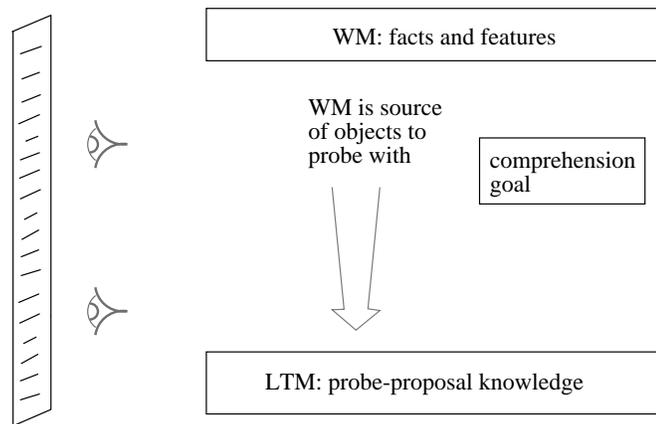


Figure 17: The model uses WM and LTM to propose probe subgoals

probing map the current goal to facts retrieved by the probe. Next time the goal is selected, either as a goal or as a probe, the model will recall all the facts it retrieved during previous selections of that goal, without having to search for it. Over time the model increases the semantic connectedness of its knowledge, by recoding facts to be associated with new objects.

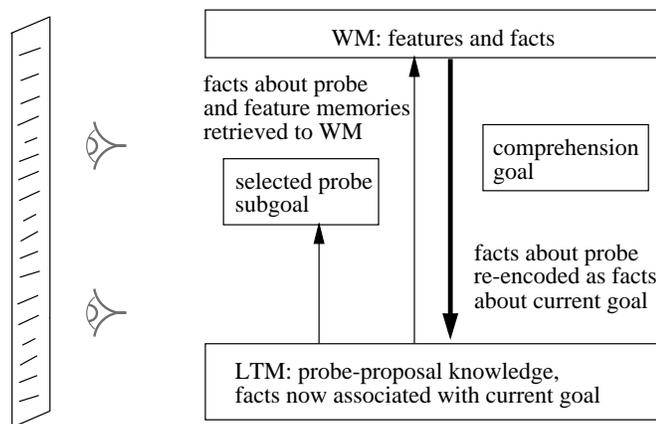


Figure 18: The model selects a probe subgoal, recalling and recoding facts

3.5.2. Imagining: retrieving features to the mind's eye

Like fixation and probing, imagining occurs through subgoals, one image per subgoal. An image is a feature that the model can retrieve to the mind's eye in WM, without the feature actually being on the display. The ability to generate images represents expertise in the form of familiarity with visual information.

Generating images gives the model access to the episodic information encoded from previously seeing a feature on the display. If an episodic memory recognizes an image, the memory retrieves its timestamp to WM. If the retrieved timestamp is different from the current timestamp, the model adds to WM that the feature is now imagined but was seen in the past. As a process for recalling episodic information, this generate-and-recognize process is consistent with two-process theories of free recall (Anderson and Bower, 1972, Kintsch, 1970). It is also similar to the proposal that people generate cues from expert knowledge to retrieve working information from LTM (Ericsson and Kintsch, 1995), a similarity discussed in detail in Chapter 7.

As an example, the model has knowledge that lets it imagine what SPs are about to fire and thereby modify the program's state data structure. The model is able to imagine all four state-modifying SPs that occur in the studied interval of the programmer's protocol. The images take the form of SP names appearing in the set of asserted SPs. With such images, the model can search for episodic knowledge about external knowledge. If the model recalls an SP about to fire, it gets a hint that the state probably has been modified.

Because imagining adds elements to WM, it causes the model to encode memories. Imagining adds both a feature and a tag that distinguishes the feature as imagined, rather than fixated. The model encodes both the imagined feature and the image tag. When these memories activate in the future, the model uses the tag to avoid "hallucinating" that it saw a feature before, when it was only an image.

3.5.3. Coverage of probing and imagining

In principle, probing and imagining can activate any of the model's knowledge, both expert and episodic. In practice, the effectiveness of probing and imagining depends on how successful they are at generating WM contexts that retrieve lots of relevant knowledge. This success depends on knowledge about knowledge — the ability to generate probes and images that help the model recall what it needs to, to carry out its task. This ability is a key element of Ericsson and Kintsch's (1995) proposed theory of human long-term working memory. Section 7 discusses the model in terms of this theory.

3.5.4. Limited working memory

Soar's working memory comes with few constraints. In particular, it has no inherent limits on persistence or span, and no parameters for setting such limits. However, some limit on WM is an essential part of a model that hopes to account for human behavior arising from the interaction of short-term, long-term, and external memory.

The model's WM consists of knowledge retrieved recently from LTM and from the display. "Recently" means during the current or the previous goal. This window of two goals was chosen for pragmatic reasons related to ease of implementation in Soar rules, and to our desire to exercise the model's ability to reconstruct WM continually (as discussed in the next paragraph). It is in fact minimal, in that with one-goal persistence the model would pursue every new goal with a blank mind, and would inherently forget why it was doing anything.

The model can accommodate this artificially forgetful WM because it can use LTM and the display to compensate. This approach to offloading WM onto encoded rules was foreshadowed by Howes's *recognition-based problem-solving* (Howes, 1993). When our model selects a goal, it retrieves everything it knows about that goal, including features it fixated with that goal selected (Section 3.4.2). The model can also retrieve this knowledge by probing to see what it knows about objects that are already in WM (Section 3.5.1). This lets the model reconstruct WM contexts, by using information already in WM to retrieve more.

The model's WM is tightly coupled to the architecture's learning mechanism. The architecture encodes a new memory for every element that enters WM.⁶ The rules that implement the model's WM are indexed under "working memory" in Section C.1, p. 124.

3.6. Mechanisms for deliberation

The model deliberates by selecting objects to comprehend and elaborating them with information. The model usually has multiple potential goals in mind at once, requiring a mechanism for deciding when to finish with the current goal, and which one to select next (Section 3.6.1).

Having selected a goal, the model must decide how to retrieve information about it. The model usually has multiple information-retrieval subgoals in mind at once, requiring a mechanism for selecting them in some order until the goal is done (Section 3.6.2).

The model's very lean comprehension process, which only elaborates objects with information, is not meant as a full account of program comprehension. We discuss the limitations of the model's comprehension in Section 6.2.2.

3.6.1. Comprehension-goal selection: converging evidence of relevance

Modeling comprehension-goal selection is difficult, because the termination criteria are vague. The purpose of a comprehension goal is to accumulate knowledge about an object, but how much is enough? And when is it time to start thinking about a different object?

Many factors have to be integrated into the decision to declare one comprehension goal accomplished and

⁶Soar avoids encoding a redundant memory, when it can tell that the same element is associated with the same cue already.

select a new one. These factors include the availability of other goals, whether those goals are by some criterion 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.

Until a new goal is proposed, there is no issue: the model continues working on the current one. When a new goal is proposed, however, the model must decide whether to terminate the current goal and select the new one. This choice point is where the tension between purposefulness and flexibility occurs. *Purposeful* behavior requires that the model stay with a goal long enough to make progress on it, and that the new goal is related to the current one so as to make the goal sequence seem coherent. *Flexible* behavior requires that the model be able to emulate humans in leaping 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 from the environment and from LTM. Given that there can be too much to think about, there must be ways to pick and choose.

The model selects a proposed comprehension goal when that goal gets *converging evidence* of relevance from the model's train of thought. This converging evidence takes the form of an information-retrieval subgoal that matches the proposed goal. All three kinds of information-retrieval subgoal can furnish converging evidence. If the model sees the proposed goal as a feature on the display, imagines the proposed goal as a feature, or uses the proposed goal as a probe, that goal will be selected.

For example, goals sometimes arise in sets, as when the model sees a set of asserted SPs on display and proposes each as a goal. (This example is taken from the model's behavior in scrolling event 4, Section 4.5.) While the model works on one, it may probe with one of the others, because SPs are important objects and therefore a source of probes. The probe may not occur immediately, because subgoal-selection knowledge may choose other subgoals first. When the probe does occur, it constitutes the necessary converging evidence for the corresponding goal and prompts the model to select it.

This converging-evidence criterion enables purposeful behavior, by delaying selection of a new goal past the point when the new goal is proposed. Converging evidence also makes for coherent goal selections. The proposed object is one that the model tried to retrieve information about during the current goal, which means the object was somehow relevant to the current goal. Converging evidence enables flexible behavior because the model contains general, goal-independent knowledge for proposing relevant subgoals (Section 6.4.2).

The goal-selection mechanism uses the distinction in Soar between proposing and selecting an operator; the implementation is discussed in Appendix B, Section B.3.

3.6.2. Subgoal selection: general heuristics

The model has a number of general heuristics that govern the selection of information-retrieval subgoals (fixate, imagine, and probe). Here we describe two important ones that draw the model to thinking about new features. These two are typical in being independent of any specific goals or other domain knowledge.

The *novel region* heuristic prefers fixating on features in a region that is on the display for the first time. This reflects the purpose of issuing commands, which is to generate new external knowledge.

The *fixate-then-probe* heuristic prefers probing with a feature immediately after it enters WM. This causes the model to retrieve whatever it might already know about some object it has decided to print out. This heuristic interleaves with the novel-region heuristic, to prefer an alternating sequence in which the model first looks at a feature and then probes with it. This sequence is the model's representation of the human process of examining a novel part of the environment, feature by feature.

The model also has feature-dependent preferences for some fixate and imagine subgoals that are proposed only in specific situations. These preferences allow more relevant special-purpose knowledge to dominate general-purpose knowledge. For example, when the model has scrolled to an old copy of an object it is comprehending, it prefers to look at object identifiers. This and related fragments of knowledge constitute a method for retrieving identifiers by scrolling. (This example is taken from the model's behavior in scrolling event 3, Section 4.4).

When the subgoal-selection heuristics are insufficient to determine the next subgoal uniquely, the model chooses indifferently from the best candidates. For example, if there is no novel region on the display, but a number of fixate subgoals have feature-dependent preferences, the model will choose indifferently among these preferred subgoals.

General subgoal-selection heuristics are indexed under "subgoal selection" in Section C.1, p. 124. Some feature-dependent preferences are indexed under "fixate preferences", and others are asserted directly by fixate and imagine proposals.

3.6.3. Widening the search for the next goal

The model keeps track of objects it has probed with during the current goal. This *relevant-objects* set represents directly-available information about what objects the model has considered relevant to the current goal. The initial value of the set is the current goal itself. The model uses this information to prefer probes it hasn't tried yet.

The model also uses relevant-objects information as a guide for what to look at. Roughly half of the model's fixate proposals (13 out of 31) depend on an object being in the relevant-objects set.⁷ For example, for the model to fixate a condition, left-hand side must be a relevant object (po*fixate*condition, p. 157).

⁷In the model code, the relevant-objects set is simply the attribute `^goal` on the second-level state.

As the model works on a goal, the relevant-objects set expands. This increases the area of the display within which the model looks for information relevant to the current goal. Because the goal selection mechanism uses fixations to help select the next goal (Section 3.6.1), the growing relevant-objects set makes a new goal easier to select. This prevents the model from getting stuck on one goal, and allows it to be flexible in selecting the next one.

3.7. Mechanisms to change the display

The model proposes command goals to change the display (Section 3.7.1). When a command goal is selected, it is interpreted by a display emulator (Section 3.7.2).

3.7.1. Command-goal selection: Immediate relevance

The model selects command goals whenever they are proposed. This makes new external knowledge available as soon as the model thinks it might be useful. After a command, the model automatically re-selects the comprehension goal that preceded the command. This minimizes the effect of the interruption, by having the model continue with the goal that the command was issued in service of. For example, after scrolling, the model still knows why it scrolled.

3.7.2. The display emulator

Display changes are brought about by a display emulator. The emulator is implemented within Soar for programming convenience, but kept distinct from the model. The emulator responds to a command by updating a representation of the screen in a separate region of Soar's working memory.

The model's commands abstract away certain aspects of the programmer's commands. For example, the model does not reason about which direction to scroll. This knowledge effectively resides in the emulator, which reads the context of a command — for example, the feature that the model wants to scroll to — and returns the screen that the programmer saw after the same command.

Some of the model's commands map to sequences of the programmer's commands. These *compound commands* in the model reflect what seem to be the programmer's more abstract intentions for changing the display. For example, the model issues only one scroll command for each consecutive sequence of scroll commands by the programmer. The programmer might have performed visual search during the pauses between consecutive scrolling commands (e.g., the pauses in scrolling event 5; see Figure 10, p. 21). The model only represents the more abstract goal of scrolling to a feature. The emulator takes care of the details.

Compound commands will be discussed further when we evaluate the model's performance (Section 5.1.2).

3.8. Summary: mechanisms and knowledge

To summarize this chapter, we present brief descriptions of the mechanisms and knowledge in the model. Figure 19 shows the mechanisms. The model retrieves knowledge by making contact with regions of output on the display (attending), then bringing individual features into WM (fixating). The model retrieves information from LTM by searching for knowledge about objects (probing), and episodic knowledge about features (imagining). The model deliberates by selecting goals to comprehend, and subgoals to retrieve information about the current goal. Finally, the model issues commands to change the display that are interpreted by a display emulator.

Figure 20 describes the expert, episodic and external knowledge available to the model. Expert knowledge fits into roles defined by the model's mechanisms (in *italics* in Figure 20). The model retrieves information from the display by recognizing regions of output and relevant features. The model retrieves information from LTM by proposing probes and images relevant to the current goal and to knowledge already retrieved. The model proposes comprehension goals based on knowing what objects are important to retrieve knowledge about, and commands based on knowing when to generate new information or scroll to hidden information.

The model learns episodic knowledge about regions and features (encoded knowledge is underlined in Figure 20). Episodic knowledge about features plays the more important role in our account of the programmer scrolling through external information. Except where we specify otherwise, "episodic knowledge" refers to knowledge about features.

In addition to episodic knowledge, the model also caches features, facts, and images. Feature memories improve access to features on display, letting the model heed them without selecting a fixate subgoal. Recoded facts associate facts directly with the current goal, when they were retrieved by probes selected in service of the current goal.

In the next chapter, which describes the model's account of the programmer's scrolling behavior, we use a format similar to Figure 19 to indicate which mechanistic components come into play in which scrolling events.

Retrieval from display	Mechanisms that let the model bring features into WM
Attend subgoal	Makes contact with display, determines novel regions, notes when features in WM scroll off
Fixate subgoal	Adds perceived feature to WM; architecture encodes feature and episodic memory for fixation event
Retrieval from LTM	Mechanisms that generate cues to retrieve knowledge from LTM
Probe subgoal	Mock goal that imports associated knowledge into context of current goal
Imagine subgoal	Generates a feature in the mind's eye, to retrieve episodic information about having seen that feature
Working memory (WM) with 2-goal persistence	Maintains elements for current and previous goal, making LTM necessary
Deliberation	Mechanisms that select comprehension goals and information-retrieval subgoals
Comprehension-goal selection based on converging evidence	Model selects proposed goal when prompted to by subgoal
Subgoal selection	Based on heuristics like fixate-then-probe and novel-region, and on feature-dependent preferences
Changing display	Mechanisms that let the model generate new information and scroll to hidden information
Command-goal selection	Based on proposal, to make external information available as soon as it seems useful
Display emulator	Responds to commands by representing the display changes that the programmer saw

Figure 19: Mechanisms in the model

Expert knowledge	Knowledge about the language, the program, and manipulating the display
<i>for retrieval from display</i>	
Attend proposals	Knowledge about what constitutes a region of output on the display
Fixate proposals	Knowledge about what features on the display are relevant
<u>Feature memories</u> (encoded from fixates)	Cache features, letting the model heed them more quickly in the future
<i>for retrieval from LTM</i>	
Probe proposals	Knowledge about what objects could retrieve knowledge relevant to current goal
Facts about objects	Display-independent knowledge about objects
<u>Recoded facts</u> (encoded from probes)	Cache facts, associating them directly with current goal
Imagine proposals	Knowledge for generating features in the mind's eye in WM
<u>Image memories</u> (encoded from imagines)	Cache images, ignored by the model
<i>for deliberation</i>	
Comprehend proposals	Knowledge about what objects are important to comprehend
<i>for changing display</i>	
Command proposals	Knowledge about when to generate and scroll through external knowledge
Episodic knowledge	Encoded by architecture when the model symbolizes events in WM
<u>About regions</u> (encoded from attends)	Represent attention events, letting the model distinguish novel regions from old ones
<u>About features</u> (encoded from fixates)	Represent fixation events, letting the model recall seeing features
External knowledge	Features visible on the display, and hidden features accessible through scrolling

Figure 20: Expert, episodic, and external knowledge available to the model. Expert knowledge is grouped by mechanisms, shown in *italics*. Learned knowledge is underlined.

Chapter 4

The model's behavior

Our thesis is that access to external information is a function of both expertise in the task domain and episodic knowledge about what features have appeared on the display. This chapter provides support for this claim, by tracing the model's behavior as it accounts for the programmer's scrolling events. Chapter 5 examines the model's fit to the protocol data at a higher level over the model's full lifetime.

4.1. Challenges in describing the model's behavior

The model's behavior is complex and difficult to represent, for several reasons. First, there is a lot of parallelism. Multiple goals and subgoals can be proposed in parallel, and multiple memories for facts and features can be retrieved in parallel.

Second, the model's trace is denser than the programmer's protocol, in terms of number of features and objects it "refers to". The model generally selects a comprehension goal or an information-retrieval subgoal for every object referred to by the programmer, and usually more. For example, where the programmer appears to focus on one SP condition, the model may fixate a number of them. This onto mapping from model trace to protocol is acceptable on the assumption that verbal protocols are incomplete (Newell and Simon, 1972, Ericsson and Simon, 1992). However, it means that in comparing the model trace to the protocol, we have to elide many of the model's actions.

Third, a lot of sequential activity can intervene between related events. For example, if an SP becomes visible on the display, and it has several conditions, the model may propose several fixate subgoals in parallel (one for each condition). But subgoals are selected in sequence, which means that a given subgoal may be selected several subgoals after it is proposed. Thus, a model action may not follow from the results of the immediately-preceding action in the trace.

Fourth, the programmer's knowledge is highly-specialized, compared, for example, to other programming tasks that have been modeled (e.g., Brooks, 1975, and Rist, 1995). Constructs common to procedural languages, and simple tasks like reversing an array or calculating average rainfall, let the reader evaluate the model's behavior without esoteric domain knowledge. In contrast, the knowledge we attribute to our programmer will be difficult for readers to evaluate unless they are versed in the program she is working on and the language she is working in.

One approach to these problems is to use several levels of detail. In our descriptions of the first two scrolling events, we aim to give a sense of the specialized knowledge and the mechanisms that must be engineered into a model of authentic expert behavior. Two events described in detail seem sufficient for this, and we describe the remaining three at a higher level.

The first two scrolling events demonstrate most of the mechanisms. The text describing these events includes a point-form list of the mechanisms introduced. The list uses the terms from Figure 19 (page 40), which summarizes the mechanisms described in the previous chapter.

The model traces that appear in the figures in this chapter abstract away many details of the model's behavior, leaving mainly those steps that lead directly to scrolling. The figures show only a small subset of rule firings, comprehension goals, and information-retrieval subgoals. Appendix D maps the model traces in the figures to complete goal- and subgoal-traces. The figures also leave out the trace segment between the encoding and recall episodes (as the figures in Chapter 2 left out the corresponding protocol segment). This means that where we pick up the model's behavior in each episode may seem somewhat arbitrary.

4.1.1. Conventions used in the figures

Figures 21 to 25, like the corresponding figures in Chapter 2, present two columns of information: display screens on the left, and a behavior trace on the right. To read the trace, begin at **model's behavior** at the top right and read down, following arrows as they appear. Where an arrow points back up, it means the model has issued a scrolling command to return to the upper screen.

In the trace, the model's goals and subgoals are shown in `Courier` font. A subgoal is indented 2 characters to the right, beneath the comprehension goal for which it was selected.

Rules that propose a goal or subgoal are shown in Roman font. Rules are numbered consecutively within each figure, for concise reference in the text. Rules encoded by the model (feature and episodic memories) are shown in white letters on a black background, at both encoding and recall time.

Comments in *italics* summarize behavior that occurs between goals and subgoals, and are used for other remarks.

4.2. Scrolling event 1

In this scrolling event, the programmer recalls something about a new chunk (Section 2.3.1, page 13). The cause of the chunk may involve a state being shared by two problem spaces and an SP that scrolled off when she printed the chunk. The model emulates this behavior by recalling the hidden asserted SP, selecting the SP as the goal to comprehend, and scrolling to it (Figure 21).

Figure 21 shows the relevant screens from the programmer's display (on the left) and a model trace (on the right). Section 4.2.1 describes the screens. Sections 4.2.2 and 4.2.3 describe the model's behavior during

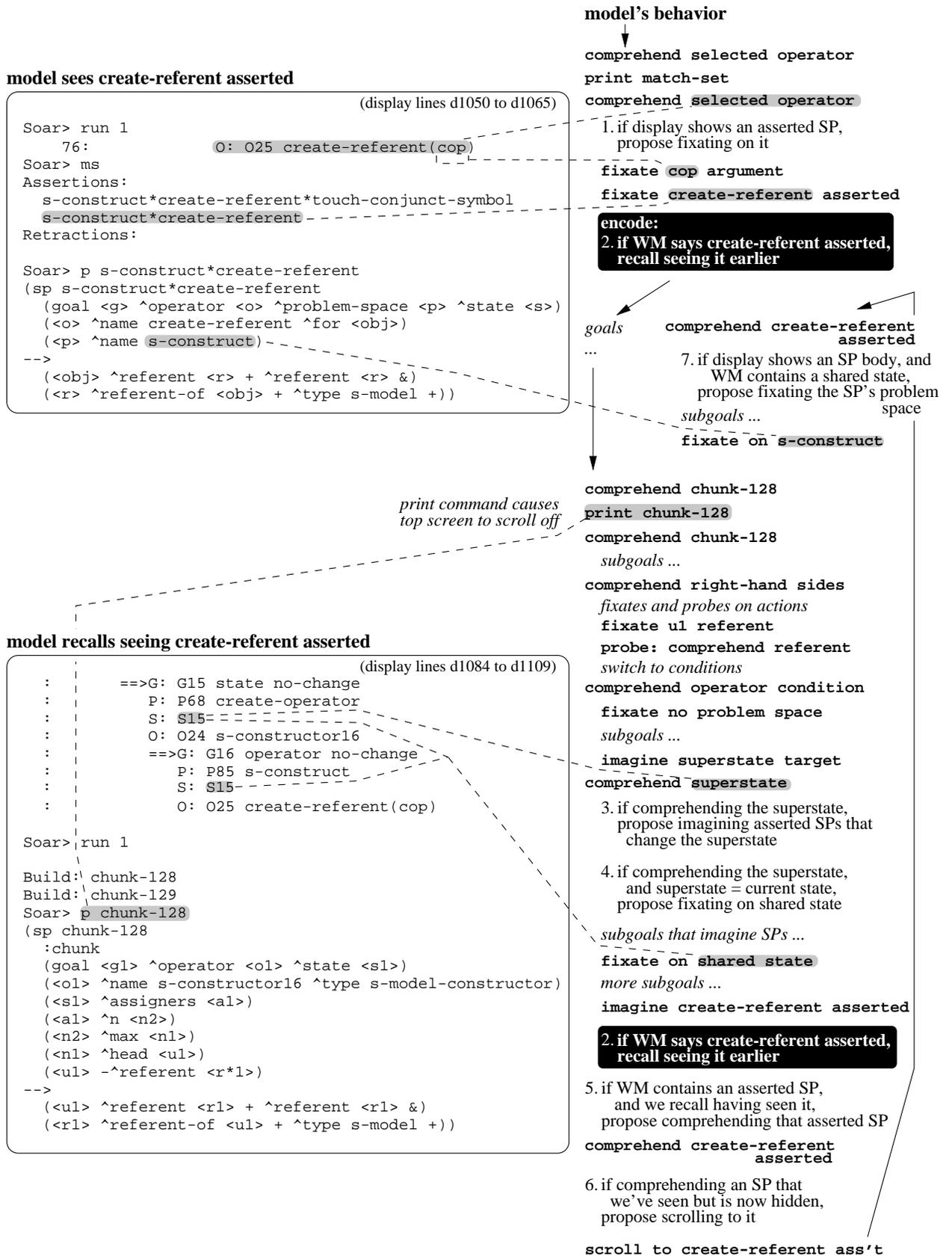


Figure 21: Scrolling event 1, model's behavior

the encoding and recall episodes, respectively. In the text describing the figure, each element of the model trace — *goal*, *subgoal*, *rule*, or *comment* — appears at the start of the paragraph that describes it (a rule at the beginning of a paragraph is underlined for emphasis).

The trace in Figure 21 leaves out many details. A more complete trace and the actual code for the rules appears in Appendix D.

4.2.1. The display

On the left of Figure 21 are two screens showing output of the programmer's program. The encoding screen, on top, shows the features on the display when the model encodes the episodic memory that causes it to scroll later. The recall screen is the one at which the model is looking when it decides to scroll to the encoding screen. The conventions used in the syntax of the model trace are described in Section 4.1.1, above.

Both screens contain line numbers in their upper right corners. These give the screen's location in the programmer's process buffer (Appendix E).

Encoding screen. The encoding screen shows three commands and their output. Each command appears after a prompt (*Soar>*), which belongs to the interpreter running in the process buffer. The first command (*run 1*) runs the program one cycle. During this cycle, the program selects an operator (*O: o25 create-referent (cop)*). The second command (*ms*) generates a match-set, which shows the two SPs about to fire to apply the operator. The third command (*p s-construct*create-referent*) prints out the body of one of the asserted SPs. (The interpreter and the language are also discussed in Section 2.2.)

The three commands work together to show what the program is about to do, at increasing levels of detail. The first prints an operator, which represents the program's next step but which hides most of the details. The match set shows more detail about what the operator does, and the SP body shows what specific changes the SP will make to the program's data structures.

Recall screen. The top of the recall screen shows the bottom part of an execution stack. The operator that the program is currently executing is at the bottom right (*O25: create-referent (cop)*). This is the same operator that the program selected during the encoding episode.

Below the execution stack are two commands. The first is a run command (*run 1*), which causes the program to build two chunks. The chunks arise from actions carried out by the current operator. The second command prints the body of one of the chunks (*print chunk-128*). As in the encoding episode, the run command causes the program to generate a visible event, and a subsequent print command generates information about that event.

4.2.2. Model behavior during encoding episode

The model's behavior during the encoding episode begins at the top right of Figure 21. The behavior contains examples of the following mechanisms:

- **Changing display:** Selecting a command goal immediately, and re-selecting the previous comprehension goal (`print match-set, comprehend selected operator`)
- **Retrieval from display:** fixate subgoal encoding an episodic memory (rule 1, `fixate create-referent asserted`)

The encoding episode begins after operator `o25` has been selected by the program and the model has set a goal to `comprehend selected operator`.

`print match-set` — The model has display-interaction knowledge that says to print the match-set after the program selects an operator. The match-set provides information on what the operator does, by listing the SPs about to fire and carry out the operator's actions (Section 4.2.1). The display emulator responds to the command by adding a representation of the match-set to the emulated display.

`comprehend selected operator` — After the print command, the model re-selects the goal to comprehend the selected operator. This allows the model to continue with the interrupted goal, now with new external information that was generated in service of that goal (Section 3.7.1).

rule 1 (propose fixating on asserted SPs) — The model perceives both asserted SPs in parallel, proposing two fixate subgoals. This is an example of expert knowledge about relevant features, which is parallel to the display-interaction knowledge that prompted the `print match set` command initially. The model selects neither subgoal right away. Other fixate subgoals have also been proposed, and one of these is selected first.

`fixate cop argument` — In parallel with rule 1, the model also proposed fixating the argument of the selected operator (`cop`). Operator arguments, like asserted SPs, provide information about the operator. The model fixates on the argument first. The model encodes feature and episodic memories as a result, but these are not shown in the figure.

`fixate create-referent asserted` — The model next selects one of the two subgoals proposed by rule 1, fixating on the create-referent assertion. The fixate subgoal deposits the create-referent assertion into WM, and timestamps it (Section 3.4.2).

rule 2 (episodic memory, encoded) — The architecture encodes two memories from this fixate subgoal. The feature memory (not shown) associates the create-referent assertion in WM with the create-referent assertion on the display and with the selected-operator goal. (A feature memory is described in a later scrolling event, in Section 4.3.) The episodic memory associates a timestamp with the create-referent assertion in WM. If the create-referent assertion enters WM in the future, either as a feature or as an image, the timestamp will be retrieved and the model will effectively recall that it saw the feature before.

4.2.3. Model behavior during recall episode

The model's behavior during this episode contains examples of the following additional mechanisms:

- **Deliberation:** goal selection based on converging evidence, both when converging evidence is delayed (`comprehend right-hand sides`) and when it occurs immediately (`comprehend create-referent asserted`)
- **Deliberation:** the fixate-then-probe subgoal-selection heuristic (*fixates and probes on actions*, `fixate referent ul,probe referent ul`)
- **Retrieval from LTM:** imagine subgoal that retrieves an episodic memory (`imagine create-referent asserted`)

Many comprehension goals later, the model has selected a goal to `comprehend chunk-128`. This goal corresponds to the programmer's desire to examine the chunks built by the program: "Let's see what the chunks are doing `{print chunk-128}`." The model's subsequent behavior, like the programmer's, is concerned with understanding `chunk-128`.

`print chunk-128` — Having selected the goal to `comprehend chunk-128`, the model selects a command to print out the actual body of the chunk. The output generated from this command causes the contents of the encoding screen to scroll off.

`comprehend chunk-128` — Just prior to the model printing `chunk-128`, the model's expert knowledge proposed several new goals, for example to `comprehend chunk-128's right-hand side`. Mechanistic knowledge prefers to continue with the comprehension goal that caused the display to be changed (Section 3.7.1). Therefore `comprehend chunk-128` is re-selected. However, the other proposed comprehension goals persist, and if converging evidence (Section 3.6.1) for one of them presents itself, control will be switched to that proposed goal.

subgoals — In service of comprehending `chunk-128`, the model selects a subgoal that probes with the concept of left-hand sides. This reflects knowledge of the syntax of the language. Left-hand sides, together with right-hand sides, make up the body of an SP. The left-hand-sides probe is a cue for fixation knowledge that looks at the individual conditions of an SP. The model thinks about SPs hierarchically, breaking them down into sides first, and then into individual conditions and actions. After probing with left-hand sides, the model probes with right-hand sides.

`comprehend right-hand sides` — Because the model had previously proposed the goal of comprehending right-hand sides, and because the conditions for that proposal are still valid, and because a probe for right-hand sides is now selected as a subgoal, converging evidence for the right-hand sides goal has occurred. Therefore `comprehend right-hand sides` replaces `comprehend chunk-128` at the goal level. Converging evidence like this occurs before subsequent switches in comprehension goals, but we will not continue to refer to it.

fixates and probes on actions,
`fixate referent ul,`
`probe referent` — The model examines the right-hand side of `chunk-128` by fixating on individual

actions. One of these actions is `<u1> ^referent`, which creates the referent attribute on the object `<u1>`. After the model fixates on this action, it probes to see what knowledge it has about the action. The model has knowledge for generating probes that says that knowledge about actions could be relevant to any other goal (a*state*important-objects, p. 172). The fixate-then-probe subgoal-selection heuristic (Section 3.6.2) interleaves fixations with probes, so that the model first fixates on something and then probes with it. The fixation on `<u1> ^referent` is an example of a step taken by the model for which there is no corresponding step in the protocol. The programmer may have examined the actions of chunk-128, but did not report doing so.

switch to conditions — The model's knowledge about the syntax of the language says that knowledge about conditions can inform the right-hand side, because conditions bind variables that appear in actions. Thus when the model is finished fixating and probing on the actions of chunk-128, it switches to the conditions.

comprehend operator condition — As it fixates chunk-128's conditions, the model sees an operator condition, and selects a goal to comprehend it. This represents language knowledge that operator conditions are important to comprehend. Getting the correct operator conditions into chunks is key to an idiom used in the programmer's program.⁸

fixate no problem space — While comprehending the operator condition, the model notices that chunk-128 does not have a problem-space condition. (Programmer: "Oh, it's not testing for a problem space.") The problem-space condition in an SP says which problem space the SP fires in. This scoping information can be useful in understanding how a chunk (a newly-created SP) will affect program execution in the future. An expert in the language might look for this condition when trying to understand a chunk, and notice if the condition is absent. The model has language knowledge that notices when a problem-space condition is absent from an SP.⁹

imagine superstate target — Because the problem-space condition is missing, the model tries to guess which problem space was modified. The model proposes imagining generic data structures in problem spaces other than the current one. It proposes imagining two such data structures: the superstate, and the top state. The superstate belongs to the next older context in the execution stack. The top state belongs to the oldest context in the execution stack. The superstate and the top state are the ones that programs typically modify.

comprehend superstate — The model considers only the superstate important enough to propose comprehending as a goal. This represents knowledge about this specific program, namely that it modifies the superstate but not the top state during this phase of its execution.

⁸The idiom creates new Soar operators dynamically, and sets of chunks that apply these operators. The operator condition tests the dynamically-generated (gensymed) name of a new operator.

⁹The absence of a problem condition is also a key element of the idiom referred to in the previous footnote. Chunks have to transfer from the problem space in which they are built to the one in which they will be used. Thus they are conditional on the operator, which arises in both spaces, but not on the problem space.

rule 3 (propose imagining asserted SPs) — While comprehending the superstate, the model begins imagining SPs that might have modified the superstate. (Programmer: "So it must have just changed it on the superstate?") One effect of SPs modifying the superstate is that the program builds chunks. With these imagine subgoals, the model is searching for episodic memories of SPs that might have led to chunks. Which SPs modify the superstate, and hence lead to chunks, depends on the program. In the programmer's program, four SPs that modify the state arise in the lifetime of the model. Rule 3 proposes four corresponding imagine subgoals in parallel.

rule 4 — In parallel with these multiple imagine proposals, rule 4 perceives that the superstate is the same as the current state (the symbol s_{15} appears in two contexts in the execution stack). Rule 4 is also conditional on the goal being to comprehend the superstate. This represents language knowledge about the effects of shared states. If the superstate is also the current state, any SP that modifies the current state also modifies the superstate, and hence will lead to chunks. Thus the model proposes retrieving information relevant to the creation of chunks, despite chunk-128 no longer being the goal. The goal to comprehend chunk-128 initiated a chain of goals (continuing with right-hand sides, operator condition, and superstate) that has led to retrieving information about the cause of chunks.

subgoals that imagine SPs — Because of rules 3 and 4, several subgoals are competing. The model has no knowledge that prefers one over another, so it picks randomly and first imagines an SP that applies a return-operator. It has no episodic memory of seeing this SP, so it goes on to imagine an SP that adds a property. Again, it has no episodic memory of seeing this SP.

fixate on shared state, more subgoals — The model now selects the subgoal to fixate on the shared state (proposed by rule 4, above). To the programmer, this shared state at first appears to be a sufficient explanation for the chunks that the program builds: "Oh these are shared states, I see". The model adds the shared state to WM, but not in any way that explicitly represents an explanation or hypothesis. (The leanness of the model's WM structures is discussed in Section 6.2.2.) The shared state does affect the model's behavior later, when rule 7 proposes fixating on the context of the shared state. In the meantime, the model continues imagining asserted SPs.

imagine create-referent asserted — Eventually the model imagines the create-referent assertion. In executing the imagine subgoal, the model deposits the imagined feature into WM, in a way that mimics a fixated feature (Section 3.5.2).

rule 2 (episodic memory, retrieved) — The episodic memory from the encoding episode recognizes the create-referent assertion, and deposits a timestamp into WM. The model compares the retrieved timestamp to the timestamp of the current goal. Because the timestamps are different, and because the model knows that the feature in WM is imagined and not actually on the display, the model infers that it saw the feature before.

rule 5 (propose comprehending an asserted SP) — Having recalled seeing a create-referent SP asserted, the model acts on this recollection by proposing to comprehend create-referent at the goal level. This proposal

reflects the effort invested by the model in searching for memories of asserted SPs. Having recalled an object with effort, the model proposes retrieving more information about it.

`comprehend create-referent asserted` — The model selects the goal immediately because the `imagine` subgoal itself provided converging evidence for the goal's relevance. Thus, the converging evidence mechanism (Section 3.6.1) can work both over a delayed interval, as with `comprehend right-hand sides` above, or immediately, as in this case.

rule 6 (`propose scrolling to recalled assertion`) — Having selected a goal to `comprehend` an asserted SP, and having recalled that the asserted SP is hidden, the model proposes scrolling to the asserted SP. Rule 6 is general in that it proposes scrolling when the model recalls either code about to execute (asserted SPs) or code fragments (conditions or actions). The same rule accounts for two other scrolling events (Sections 4.5 and 4.6). The model also has general rules for scrolling to data structures. With these general scrolling rules, the model's scrolling behavior is limited mainly by when it chooses to search for episodic memories by `imagining` (Section 3.5.2).

`scroll to create-referent asserted` — As discussed above, whenever a command goal is proposed, the model selects it immediately (Section 3.7.1) to make the information it generates available as soon as possible. Thus, the model scrolls back to the encoding screen, where it saw the `create-referent` assertion earlier. The programmer scrolls apparently after changing her mind and no longer being satisfied with visible information ("No I don't see, what built that?"). Similarly, the model scrolls because it remembers some relevant hidden information, though it does not have an explicit representation of doubt about a previous explanation.

`comprehend create-referent asserted` — After scrolling, the model again re-selects the comprehension goal from before the command, and continues comprehending what it scrolled to.

rule 7 (`propose fixating on problem-space condition`),
`fixate on s-construct` — Having scrolled to the asserted SP, the model finds the SP printed out. Rule 7 perceives the printed-out SP, and proposes fixating on the SP's problem-space condition. Rule 7 is conditional on WM containing a shared state (see rule 4) and on WM not containing the problem space for that state. This represents the causal connection in the language between problem-space conditions and chunks. The problem space condition of an SP determines whether the SP builds a chunk when it fires.¹⁰ With rule 7, as with rule 4, the model is still retrieving information about chunks, as an indirect consequence of having selected the chunk-128 goal earlier.

¹⁰If the current state is shared, and the SP fires in the current space (as determined by the problem-space condition), it will cause a chunk, because the current state is also the superstate. If the SP fires in the superspace, it will not cause a chunk, because the SP's effects will be contained in one space, rather than crossing from one space into another. (Section 2.2 gave an overview problem spaces and chunks.) In this case, the SP fires in the current space, which means that it led to the program building chunks.

4.2.4. Summary

The recollection that leads the model to scroll occurs because the model deliberately imagines asserted SPs (SPs about to fire and modify data structures) that could have modified the state it is thinking about. The model recalls having seen the create-referent SP asserted, and scrolls back to find out more about it. The programmer's apparent goal of understanding the cause of chunk-128 is not the goal that immediately prompts the model to scroll. Rather, the model's goal to comprehend chunk-128 sets off a chain of goals and subgoals that eventually leads to recalling an asserted SP. The link between chunk-128 and the information retrieved about it is implicit, distributed over a collection of rules representing individual elements of expert knowledge.

The model scrolled because it remembered an asserted SP, not because it remembered the hidden SP body. The programmer's verbal protocol is ambiguous on this point, giving no direct evidence as to whether she remembered the assertion or whether she remembered the displayed SP itself. The model's account is plausible, because scrolling back to an asserted SP has its own benefits. SP names are often too long and complicated to generate from memory accurately, let alone efficiently. In another scrolling event (Section 2.3.4), the programmer scrolls an asserted SP into view, then uses the editor to copy the SP name into a print command. In the current scrolling event, the programmer might have scrolled to retrieve the SP name as part of a method for printing SPs, and simply found the SP already printed out.

4.3. Scrolling event 2

In this scrolling event, the programmer scrolls back to the screen she left during the first scrolling event, apparently still concerned about why the program built chunks. The shared state is part of her explanation, both before leaving the bottom screen and after returning to it.

This scrolling event is different from the first in that the encoding screen is hidden only for a short time (14 seconds, from t431 to t445; see Figure 7, page 16). The programmer could have kept information in WM long enough to span from the encoding episode to the recall episode, without having to resort to encoding and retrieval from LTM.

The model accounts for the programmer's behavior by using episodic information that persists in WM. While the model's behavior does not depend on encoding and recalling information from LTM, we use the same terminology as before. During the encoding episode, the model adds information to WM. During the recall episode, the model uses this information to return to the encoding screen.

Figure 22 shows the screens and the model trace. The screens are the same as in the first scrolling event, and are described in Section 4.2.1 (above). The model's behavior begins opposite the lower screen, at the middle right of Figure 22, with the encoding episode. The model's behavior continues at the top right with the recall episode, as indicated by the gray arrow leading up from the `scroll window up` command. At the end of the recall episode, the model scrolls back to the encoding episode, indicated by the dark arrow leading down from the `scroll to state` command.

shared state briefly hidden

```
(display lines d1050 to d1065)
Soar> run 1
76:          O: 025 create-referent(cop)
Soar> ms
Assertions:
  s-construct*create-referent*touch-conjunct-symbol
  s-construct*create-referent -----
Retractions:

Soar> p s-construct*create-referent
(sp s-construct*create-referent
(goal <g> ^operator <o> ^problem-space <p> ^state <s>)
(<o> ^name create-referent ^for <obj>)
(<p> ^name s-construct)-----
-->
(<obj> ^referent <r> +/^referent <r> &)
(<r> ^referent-of <obj> + ^type s-model +))
```

model sees shared state

```
(display lines d1084 to d1109)
:      ==>G: G15 state no-change
:      P: P68 create-operator
:      S: S15-----
:      O: O24 s-constructor16
:      ==>G: G16 operator no-change
:      P: P85 s-construct
:      S: S15-----
:      O: O25 create-referent(cop)

Soar> run 1
Build: chunk-128
Build: chunk-129
Soar> p chunk-128
(sp chunk-128
:chunk
(goal <g1> ^operator <o1> ^state <s1>)
(<o1> ^name s-constructor16 ^type s-model-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 +))
```

model's behavior

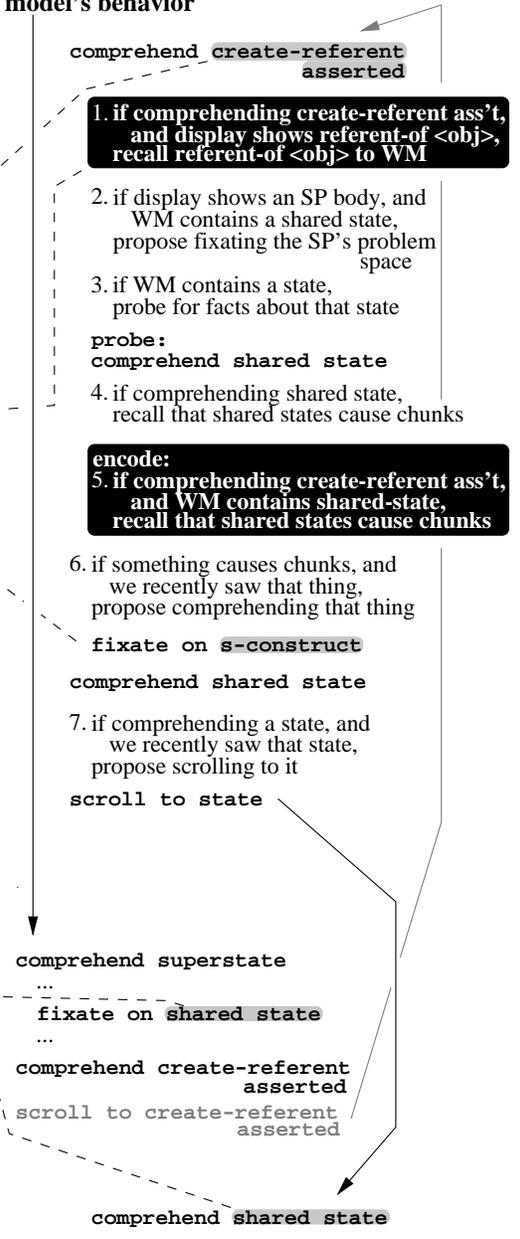


Figure 22: Scrolling event 2, model's behavior

Sections 4.3.1 and 4.3.2 (below) describe the model's behavior during the encoding and recall episodes, respectively. As in the description of scrolling event 1, an element of the model trace (goal, subgoal, rule, or comment) appears at the start of the paragraph that describes it.

4.3.1. Model behavior during encoding episode

The trace of the encoding episode, beginning with the `comprehend superstate` goal next to the bottom screen, is a synopsis of the more complete trace described in the previous scrolling event's recall episode (Section 4.2.3).

The model's behavior during this episode contains an example of the following additional mechanisms:

- **2-goal persistence of WM:** The model remembers features through current and previous goal (entire episode)
- **Retrieval from display:** The attention mechanism recognizes when features in WM become hidden on the display (`scroll to create-referent asserted`)

`fixate on shared-state` — The key step in this episode is that the model fixates on the shared state, in service of a goal to comprehend the superstate. (Programmer: "Oh these are shared states, I see; no I don't see, what built that? {`scroll window up`}".) When this subgoal is selected, the model learns a feature memory and an episodic memory (a side-effect of all `fixate` subgoals), but these memories will not be necessary to evoke scrolling in this case.

`comprehend create-referent asserted` — As described in detail in scrolling event 1, the model eventually replaces the `comprehend superstate` goal with the `comprehend create-referent asserted` goal. The model has a WM persistence of 2 goals, so the `shared-state` feature persists in WM through this change in goals.

`scroll to create-referent asserted` — The model scrolls to the `create-referent` assertion in service of the `comprehend create-referent asserted` goal. Command goals like scrolling or printing SPs are always in service of a comprehension goal. The model does not count them toward the 2-goal persistence limit on WM, so it continues to consider `comprehend create-referent asserted` the current goal, and the `shared-state` feature remains in WM. After the scroll, the `shared-state` feature is no longer on the display. The model automatically recognizes when features in WM become hidden (Section 3.4.1.1), and tags the `shared-state` feature in WM with this information.

4.3.2. Model behavior during recall episode

The recall episode, shown at the top right of Figure 22, begins immediately after the encoding episode. The model's behavior during this episode contains examples of the following additional mechanisms:

- **Retrieval from display:** retrieving a feature memory (rule 1)
- **Retrieval from LTM:**
 - Probe subgoal proposed by knowledge of what objects could inform other objects (rule 3)
 - Subgoal-selection heuristics that prefer probes when the probed-with element is likely to drop out of WM soon (`probe: comprehend shared-state`)
 - Fact recalled about probe object (rule 4)
- **Recording a fact:** The model encodes a new rule that associates a fact with the goal current when the fact is retrieved (rule 5)

`comprehend create-referent asserted` — The scroll command was in service of this comprehension goal, so the model re-selects this comprehension goal immediately.

rule 1 (feature memory, retrieved) — The model has feature memories in LTM for features of the top screen. The model encoded these feature memories the last time this screen was on display. They are conditional both on the WM elements that caused the corresponding fixate subgoals to be proposed, and on the comprehension goal at the time (Section 3.4.2.1). The effect of these memories is to recognize certain features and deposit them in WM automatically, without the model having to select the corresponding fixate subgoals. One of these feature memories is rule 1. This memory retrieves `^referent-of <obj>`, an action of the create-referent SP which is printed out on the display. This memory becomes active whenever that action is visible and comprehending the create-referent assertion is the goal.

rule 2 (propose fixating on problem-space condition) — Many different subgoals are proposed in service of comprehending the create-referent assertion. Some of these are proposed in response to feature memories retrieved when the model selected the goal. Other subgoals are proposed by expert knowledge reacting directly to the new display. Rule 2 is one of the latter (rule 2 here is rule 7 in scrolling event 1, and is described in detail on page 51). This rule proposes fixating on the problem-space condition of an SP when WM says that a state is shared between two execution contexts (problem spaces). Given a shared state, the problem-space condition indicates whether the SP containing that condition causes chunks to be built. Rule 2 can fire because the 2-goal persistence allowed the shared-state feature to remain in WM.

rule 3 (propose probing with shared-state) — The model knows that a shared state is a kind of state, and that information about states could inform any given goal. Rule 3 therefore proposes probing for facts about shared states. The model is also able to probe with other generic data structures, including superstates and operators, and with code fragments, including chunks, SPs, conditions, and actions. If the model is aware that a given element belongs to one of these categories, it will propose that element as a probe. Appendix:appendix describes the model's probe-proposal knowledge in detail.

`probe: comprehend shared-state` — The model has heuristic subgoal-selection knowledge that

prefers probing with hidden features. Hidden features still in WM are likely to fall out of WM soon, without the possibility of retrieving them from the display, and therefore should receive priority if they are to be probed with at all. Because WM still contains the shared-state feature from the previous goal, and because the feature is tagged as hidden, the model selects this probe as soon as possible.

rule 4 (knowledge that shared states cause chunks) — When the model probes with the shared state, it recalls a fact about shared states, namely that they cause chunks to be built. Up to now, the model has had no explicit representation of the causal link between shared states and chunks. In contrast, the programmer guessed during the encoding episode that the shared state caused the chunks built by the program ("Oh these are shared states, I see; no I don't see, what built that?"). Had the model probed with the shared state during the encoding episode (after fixating on the shared state), it would have retrieved the causal link then.

rule 5 (fact about shared states, recoded) — So far, the knowledge that shared states cause chunks has only been accessible by choosing shared state as a goal or as a probe. Retrieving a fact to WM causes the architecture to learn a new rule that adds another path by which the model can gain access to this fact (Section 3.5). In this case, the new rule associates the fact that shared states cause chunks with the current goal (`comprehend create-referent asserted`) and the information in WM that the current state is shared. Probing plus the architecture have recoded general language knowledge (rule 4) as an increment of program-specific knowledge (rule 5).

rule 6 (propose comprehending something that built chunks) — The model has just recalled that shared states cause chunks (rule 4). It still remembers the (hidden) shared-state feature from the previous goal. Expert knowledge proposes that if something causes chunks and that thing is hidden, then set a goal to comprehend that thing (rule 6). A programmer might, in a similar fashion, set out to verify an explanation, when the verifying information is known to exist as hidden external information. Rule 6 combines language knowledge (that the cause of chunks is important to understand) with low-level tactical knowledge about verifying information from LTM against external information. Rule 6 is conditional on the external information being hidden. The short-term episodic information indicating this hiddenness is a function of both the attention and WM mechanisms. Attention generates the episodic information, by noticing when a feature in WM becomes hidden (Section 4.3.1). WM maintains this episodic information as the model works on an intervening goal.

`fixate on s-construct` — The model eventually selects this subgoal, which was proposed by rule 2 (above). The proposal for this subgoal represents knowledge of the link between problem spaces, shared states, and chunks (discussed on p. 51). When selected, the subgoal adds to WM that the current context (the one in which the SP on display recently fired) is the s-construct problem space, and that this context shares a state (leading to chunks being built). The problem-space condition is one of two features that the programmer apparently looks at: "This said, if you're in the s-construct problem space, you slap that attribute on the object". The last clause ("you slap that attribute on the object") is probably not accounted for by the model. The feature the programmer refers to is an SP action that affects an "object" — probably the action `^referent <r>`, which affects the variable `<obj>`. The model does not have a chance to fixate on this action before it scrolls back to the bottom screen, and this action is not the one for which the model has a feature memory (rule 1).

`comprehend shared state` — The previous fixate subgoal, which represents information about the shared state, also represents the converging evidence for the shared-state goal to be selected. The earlier shared-state probe did not provide converging evidence, because the ability of a probe to provide converging evidence is qualified. When WM says that the model recently scrolled in search of an imagined feature, only features can provide converging evidence. This forces the model to make use of its scrolling action. Before the model can move on to a new goal, it must examine the display context surrounding the scrolled-to feature.

rule 7 (propose scrolling to a hidden state) — Having selected the goal to comprehend a state, and knowing that the state is hidden, the model proposes scrolling to the state. Rule 7 proposes scrolling to any goal that the model is aware is a state, if that state is known to exist as a hidden feature.

`scroll to state` — The model's preference for getting more external information causes this command goal to be selected immediately.

`comprehend shared-state` — Back at the bottom screen, the model re-selects the shared-state goal that was the impetus for scrolling. WM maintains the information that the model retrieved while at the top screen, in particular the problem-space condition (`s-construct`).

The programmer at this point appears to carry out a matching process between information she retrieved from the top screen, and what is now visible on the bottom screen. In particular, she notes from the visible execution stack that the current execution context is `s-construct`. This matches the problem-space condition of the create-referent SP. Thus the create-referent assertion fired in the current execution context, which is the circumstance under which it would cause a chunk to be built. The programmer seems to have verified the causal connection between the problem space, the shared state, and the resulting chunk: "So I'm in the `s-construct` problem space, I'm going to slap that thing on there, and lo and behold, I get this chunk, because they share the state".

The model does not carry out this kind of explicit matching. The model's representation of features is not rich enough to allow such comparisons. For example, the model has the `s-construct` problem space in WM from the top screen, but this prevents it from fixating on the `s-construct` problem space again on the bottom screen. The model's representation lacks additional contextual or episodic information that would distinguish the two features.

Even if the representation were rich enough to allow two instances of `s-construct` to be in WM at the same time, the model has no process for explicit matching of information in WM. Such a process would have to be added were the model to account plausibly for detecting and correcting errors in code, for example.

4.3.3. Summary

During the encoding episode, the model fixates on the shared state. For reasons independent of this feature, the model scrolls to the top screen. While the model is there, the shared state persists in WM, tagged as hidden. The model probes for what it knows about the shared state, because states are good probes. This causes the model to recall that shared states cause chunks. Based on this recollection, and on episodic knowledge maintained in WM that a shared state exists as hidden information, the model scrolls the shared state back into view. Because the shared-state feature was recently seen, this scrolling action is triggered without the use of LTM.

4.4. Scrolling event 3

In the previous 2 scrolling events — one based on LTM and the other on WM — we walked through the model’s behavior in considerable detail. We hope that the syntax of the figures and the role of the underlying mechanisms are clear enough that we can retreat from the details of the model’s routine behavior. For the remaining 3 scrolling events, we describe the model’s behavior at roughly the level that we described the programmer’s behavior in Chapter 2, and include relevant protocol excerpts as quotations. More detailed traces and code appear in Appendix D.

Overview: The programmer scrolls back to an object she printed out previously, to retrieve its identifier. Having retrieved the identifier, she returns to the prompt and prints a fresh copy of the object (the identifier is a required parameter for printing an object). Her memory for the hidden copy of the object seems reliable and rapidly accessible, because she scrolls to the hidden copy instead of using another method, based on visible information, for retrieving the object’s identifier.

The model emulates this behavior by scrolling back to an object it recalls having seen. Back at the object, it sees the object’s identifier and decides to print a fresh copy of the object (Figure 23). The model explains the programmer’s robust memory for the scrolled-to object by positing that the episodic memory for the object was retrieved for another purpose as well, namely to gauge the program’s progress in adding structure to that object.

Encoding episode: The model is comprehending an attribute (`for`) of an operator that the program just selected. The model has expert program knowledge that says that the object associated with this attribute is important to comprehend, and selects a goal to `comprehend for object`. The `for` attribute points to its associated object by means of a linking object identifier (`u20`). From knowledge about how to use the interpreter to generate relevant information, the model knows to print the object associated with a familiar attribute, when the object identifier is in WM.

t250: What is u20, the operator has this `{print u20}` for argument

Having printed the `u20` object, the model recognizes the object as an *utterance model*, a conceptual construct specific to the program. The model recognizes the utterance model by looking at the object’s

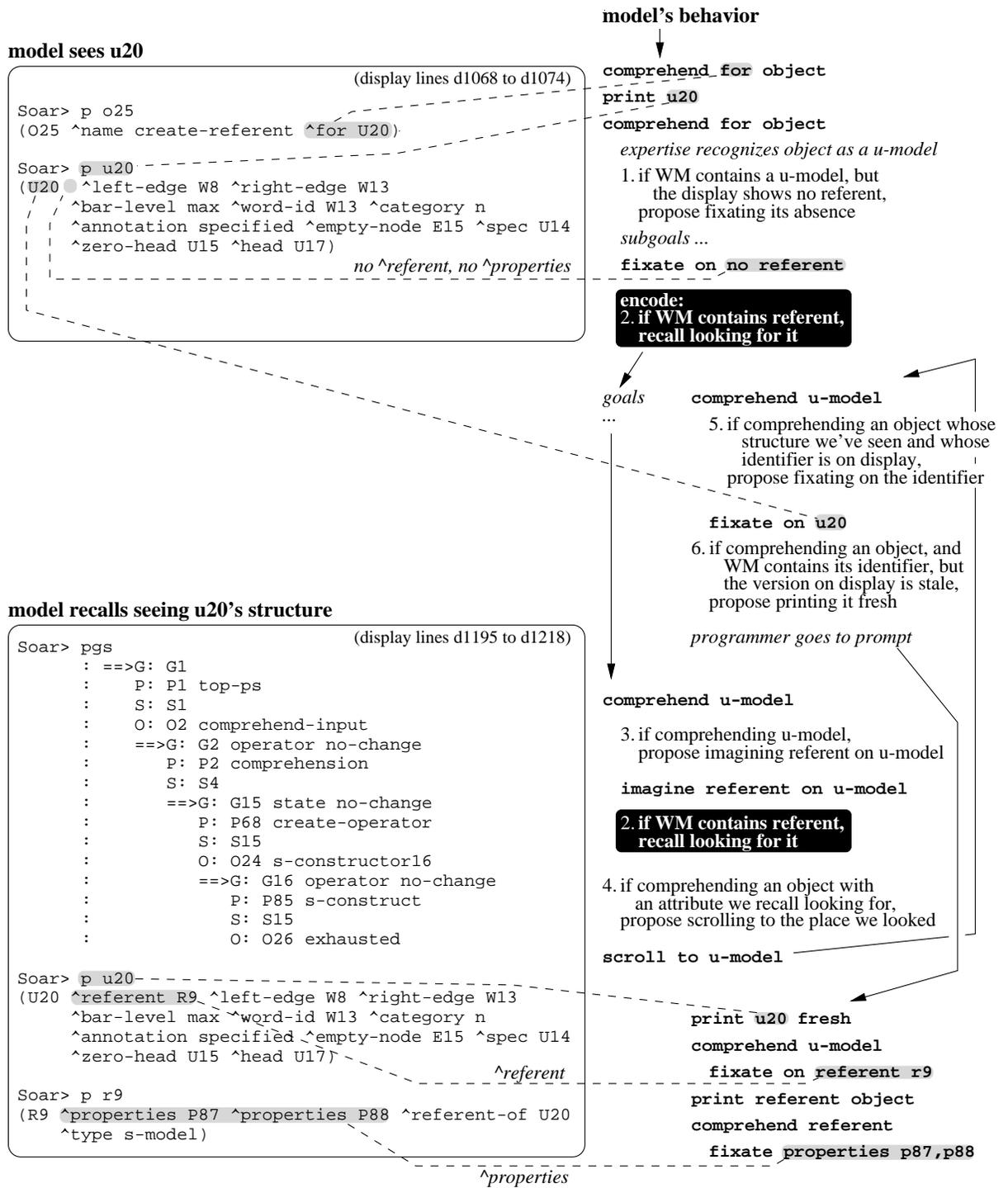


Figure 23: Scrolling event 3, model's behavior

attributes and probing for facts about them. Probing recalls program knowledge that these attributes are part of an utterance model.

The for argument is the profile in the u-model

Knowing that u20 is an utterance model, the model notices that a specific attribute is missing (the referent attribute, rule 1). This is knowledge about the program. The referent attribute points to an object that the program is about to create in this phase of its execution. When complete, the referent object will include the `properties` attribute mentioned by the programmer.

This is just the bare node, it doesn't have any of the properties

The model encodes a memory for having looked for the referent attribute (rule 2).

Recall episode: The model is comprehending the program's utterance model (`comprehend u-model`), though the utterance model is not on display. The model imagines a referent attribute as part of the utterance model, deliberately trying to recall whether it has seen the referent attribute yet (rule 3). As noted above, the referent attribute and its associated object are under construction. One purpose for imagining the referent attribute would be to retrieve episodic information about whether the program had created the attribute yet.

The image activates the memory from the encoding episode (rule 2), reminding the model that it looked for the referent attribute earlier. Still comprehending the utterance model, and knowing that a referent is part of the utterance model, the model scrolls to where it looked for the referent (rule 4). This represents an inference that the model examined an utterance model in the past.

```
t638: Ok where am I {print stack} s15 now has an utterance model object, "u"-something {scroll
window up, scroll window up}
```

Back at the top screen, the model sees u20. The model knows to look at the identifier of the object containing the recalled feature (rule 5). This implicitly assumes that the setting of a recalled feature is worth examining.

The model is still comprehending the utterance model, and now has an utterance model identifier in WM. Because the visible utterance model had to be scrolled into view, the model infers that it is stale, and prints a new copy (rule 6). The fresh copy has a referent attribute (`fixate on referent r9`). The model prints the referent subobject (`print referent object`), and sees on the object's two properties attributes (`fixate properties p87, p88`).

```
t646: u20, let's look at u20 {goto prompt, print u20} right, which has a referent {print r9} it's
sitting there, it has two properties, everything looks good
```

4.4.1. The model's explanation of why the programmer scrolled

The model makes some very specific claims about what the programmer was thinking. It imagined a referent as a key attribute of an utterance model. This retrieved a memory for having looked for a referent, which implied the existence of a hidden utterance model, which led to scrolling. Here we try to separate these specific claims from the model's general hypotheses about information access. This is in preparation for a discussion later (Section 6.7), in which we speculate on how the model could be extended to give an alternative account of the programmer's behavior.

The model scrolls as the end result of a chain of knowledge. First, at fixation time the model knew to look for a referent attribute on the utterance model, even though the attribute was missing. Timestamping this fixation event caused the architecture to encode an episodic memory. Second, the model was able to imagine the referent feature, retrieving the episodic memory. Third, the model knew to do this when it was useful. Fourth, the model knew to act on the retrieved memory, by scrolling to the setting of the recalled feature.

The first general hypothesis to come out of this sequence is that the programmer previously encoded an episodic memory for some aspect of the utterance model. The model makes a specific claim, grounded in the protocol, that this feature was the referent. However, if another model hypothesis is correct, that people passively encode large quantities of information about what they see, then the programmer could have had memories for many different features of the hidden utterance model.

The second general hypothesis is that the programmer deliberately searched for memories of an utterance model. The specific motive for initiating this search, and what feature it recalled, are in some sense incidental. Our claim, again grounded in the protocol, is that the programmer tried to recall a referent as a measure of program progress. This claim shows that the general hypothesis is tenable, but other explanations are plausible. One is that the programmer knew before scrolling that she wanted to print a fresh copy of the utterance model, and scrolled directly to retrieve the identifier. This explanation also supposes that she deliberately tried to recall seeing an utterance model.

4.5. Scrolling event 4

Overview: The programmer scrolls to a symbol that only recently scrolled off and is only a few lines over the top of the screen. When the symbol reappears, the programmer copies it into a print command. The model emulates this behavior by keeping this symbol in WM while it is hidden, and scrolling the symbol back into view after working on an intervening goal (Figure 24).

This scrolling event shows another way in which episodic memories affect behavior. The model sees two items in a set, and makes the same choice that the programmer does about which to select as the goal. The model makes the choice based on having seen one item before, but not the other.

Encoding episode: The model is comprehending how to return a new data-structure pointer from the

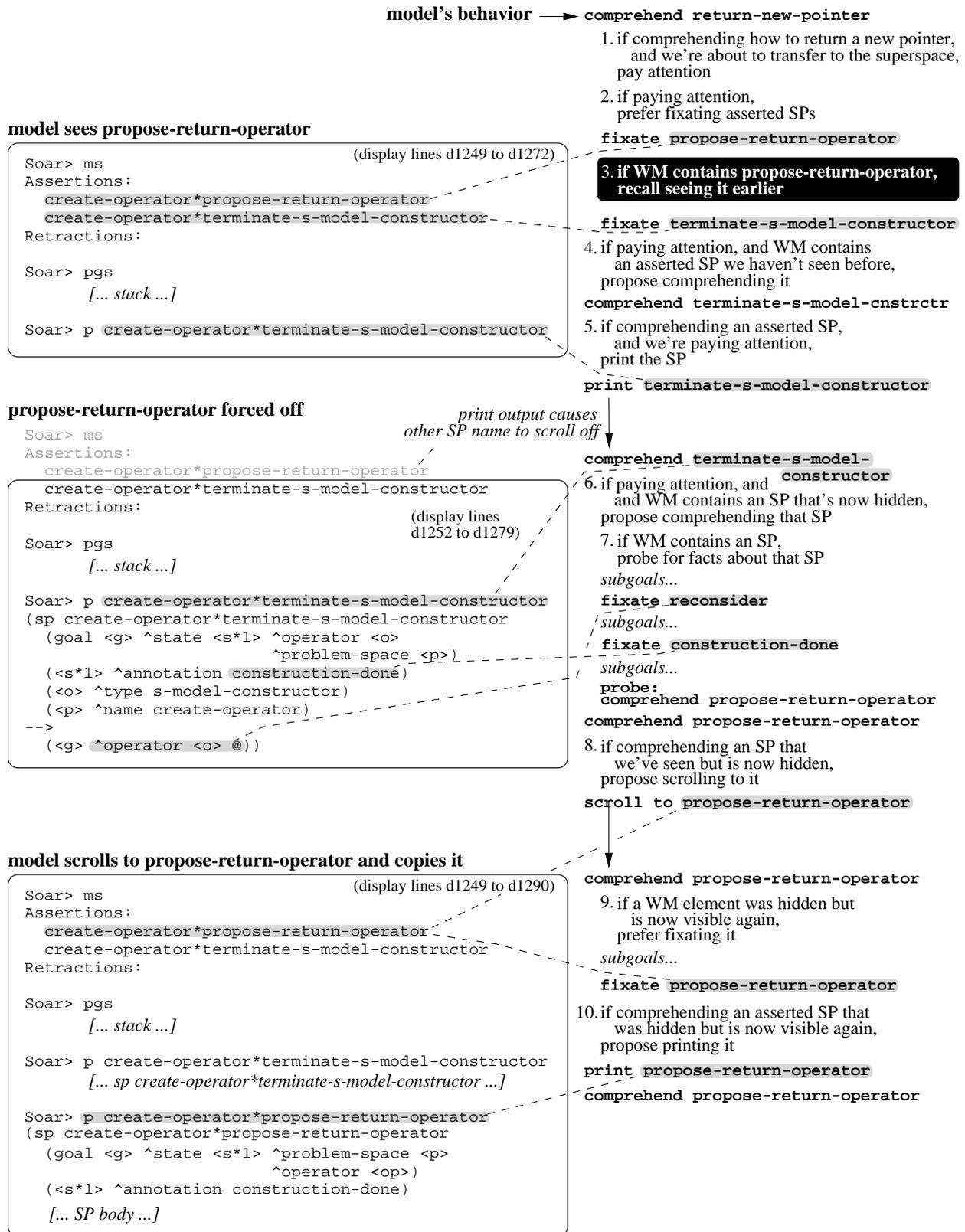


Figure 24: Scrolling event 4, model's behavior

current problem space to the superspace (`comprehend return-new-pointer`). This goal derives from the programmer's high-level goal for modifying the program. The model recognizes that transfer of control between problem spaces is about to happen, and goes into "pay-attention" mode (rule 1).

t745: propose return operator, I think we're getting to the right place

When paying close attention to the program's behavior, the model considers it important to look at SPs about to fire (rule 2). The model looks at both SPs about to fire (abbreviated `propose-return-operator` and `terminate-s-model-constructor`). The model has seen `propose-return-operator` before. The episodic memory encoded then activates now (rule 3).

The second SP, `terminate-s-model-constructor`, is new to the model. The model knows that when paying attention it should comprehend an SP it hasn't seen before (rule 4), and that in service of this goal it should print the SP body (rule 5).

`terminate-s-model-constructor, see what that's doing {print terminate-s-model-constructor}`

Recall episode (middle screen): Printing the body of `terminate-s-model-constructor` causes the `propose-return-operator` assertion to scroll off. However, the name of this asserted SP persists in WM, tagged as hidden, while the model is comprehending `terminate-s-model-constructor`.

Still paying attention, and having an asserted SP in WM that recently became hidden, the model proposes a goal to comprehend that SP (rule 6). This proposal acts as a mental note for the model to return to something important that it has temporarily lost sight of. The newly-proposed goal is not immediately selected, as no converging evidence supporting this goal has presented itself.

Having an SP in WM also causes the model to propose probing for facts about it (rule 7). SPs are important objects (like states; see rule 2 in scrolling event 2), and the model knows that it is generally useful to probe for facts about important objects. The newly-proposed probe is not immediately selected, as the model selects other information-retrieval subgoals first. Two of these other subgoals fixate on fragments of the `terminate-s-model-constructor` SP (still the current goal). One fixates on the action creating the *reconsider* control symbol (@; Laird et al., 1993), and the other fixates on the condition testing the `construction-done` attribute.

When the model finally does select the `propose-return-operator` probe (proposed by rule 7), the probe represents converging evidence for the `propose-return-operator` goal (proposed by rule 6, and still active). The model selects the goal to `comprehend propose-return-operator`. Because this asserted SP is still hidden, and tagged as such in WM, the model scrolls to it (rule 8).

Ok fine that looked for the construction done {scroll 1 line up} to put out the {scroll 1 line up}
reconsider {scroll 1 line up}

When the `propose-return-operator` assertion reappears, a heuristic applies that prefers fixating on something that was hidden and is now visible again (rule 9). This heuristic reflects the model's investment in scrolling

back to something. The goal is still propose-return-operator, and with this visible again as an asserted SP, the model proposes printing out its body (rule 10). The programmer uses the visible name to construct a print command by cut-and-paste. Given the length and complexity of SP names, this method is much faster and more accurate than trying to construct a print command from memory.

```
now what is this doing {print sp body}
```

Having printed the SP body, the programmer sees that the newly-printed SP shares a condition with terminate-s-model-creator ("that also looks for the construction-done"). The model's representation of this condition does not allow it to discriminate between the two instances. Because the model fixated on this condition in terminate-s-model-creator, it does not fixate on it again in propose-return-operator. A similar failing of the model's representation was described in Section 4.3.2.

Summary: The model's order of comprehending the two asserted SPs, with the second selected first, matches the programmer's order. The model's order is based on an episodic memory for having seen the first SP before. Because the model is paying close attention, it first examines the other, novel SP.

While the model is comprehending the novel SP, the familiar one persists, both in WM and as a proposed goal. When the model thinks about the familiar SP again, by probing for facts about it, the familiar SP becomes the goal and the model scrolls to it.

4.6. Scrolling event 5

Overview: The programmer scrolls back to chunks she printed out several minutes earlier, on a hunch about their conditions. The hunch seems to be triggered by information on display during the recall episode. The model scrolls because it recalls a condition of one of the chunks. The recollection occurs because the model sees a fragment of the condition in a different context, and imagines the fragment as a condition (Figure 25).

This scrolling event shows the model making a display-based inference (Larkin and Simon, 1987), putting two features together to guess a third based on knowledge that this third feature is likely to exist.

Encoding episode: The model has selected a goal to comprehend chunk-128. After some work on this goal it switches to comprehending right-hand sides of SPs. This goal is more general, being independent of any particular chunk, but also more specific, because it enables fixation knowledge that examines individual actions and conditions of chunk-128. The model selects a subgoal to fixate condition s-creator16, encoding an episodic memory for this feature (rule 1).

```
t379: Ok, this chunk is testing for s-creator16
```

Recall episode: The model is comprehending the new-operator attribute at the bottom of the screen, which points to the object identifier o24. The model already has information in WM that o24 identifies an operator. It checks for o24 in the run-time stack on display, to see if o24 identifies any operators in this

model sees condition s-constructor16

```

Soar> run 1 (display lines d1093 to d1109)
Build: chunk-128
Build: chunk-129
Soar> p chunk-128
(sp chunk-128
 :chunk
 (goal <g1> ^operator <o1> ^state <s1>)
 (<o1> ^name s-constructor16
 ^type s-model-constructor)
 (<s1> ^assigners <a1>)
 (<a1> ^n <n2>)
 (<n2> ^max <n1>)
 (<n1> ^head <ul>)
 (<ul> -^referent <r*1>)
 -->
 (<ul> ^referent <r1> + ^referent <r1> &)
 (<r1> ^referent-of <ul> + ^type s-model +))

```

model recalls seeing condition s-constructor16

```

: ==>G: G15 state no-change (display lines d1263 to d1302)
: P: P68 create-operator
: S: S15
: O: O24 s-constructor16
: ==>G: G16 operator no-change
: P: P85 s-construct
: S: S15
: O: O26 exhausted

Soar> p create-operator*terminate-s-model-constructor
(sp create-operator*terminate-s-model-constructor
 (goal <g> ^state <s*1> ^operator <o>
 ^problem-space <p>)
 (<s*1> ^annotation construction-done)
 (<o> ^type s-model-constructor)
 (<p> ^name create-operator)
 -->
 (<g> ^operator <o> @))

Soar> p create-operator*propose-return-operator
(sp create-operator*propose-return-operator
 (goal <g> ^state <s*1> ^problem-space <p>
 ^operator <op>)
 (<s*1> ^annotation construction-done)
 (<p> ^name create-operator)
 (<op> ^type { << u-model-constructor
 s-model-constructor >> <t*1> })
 -->
 (<g> ^operator <o> + ^operator <o> >)
 (<o> ^name return-operator + ^new-operator <op> +))

Soar> [... run commands ...]
80: O: O29 return-operator
Soar> p o29
(O29 ^name return-operator ^new-operator O24)

```

model's behavior

↓

comprehend chunk-128
subgoals ...

comprehend right-hand sides
subgoals that fixate actions and conditions...

fixate condition s-constructor16

encode:
1. if WM contains condition s-constructor16,
recall seeing it earlier

goals ...

comprehend condition s-constructor16

screen scrolls off;
more goals ...

WM contains id o24

comprehend new-operator
2. if display shows an operator with an
identifier already in WM,
propose fixating that operator

fixate operator s-constructor16
more subgoals...

comprehend left-hand-sides
fixate on individual conditions ...

fixate condition s-model-constructor

3. if WM contains an operator, and the
s-model-constructor operator type
as a condition,
propose imagining the operator as a condition

imagine condition s-constructor16

**1. if WM contains condition s-constructor16,
recall seeing it earlier**

4. if WM contains a condition
we recall having seen,
propose comprehending it

comprehend condition s-constructor16

5. if comprehending a condition
that we've seen but is now hidden,
propose scrolling to it

scroll to condition s-constructor16

Figure 25: Scrolling event 5, model's behavior

stack (rule 2). This would provide information about o24, and hence about the new-operator attribute that points to it. Identifier o24 exists in the execution stack, identifying operator s-structor16, and the model selects a subgoal to `fixate operator s-structor16`.

t821: Ok it says new-operator o24, which is in fact s-structor16

After more subgoals in service of the new-operator goal, the model eventually selects a goal to `comprehend left-hand sides`. Like the goal to comprehend right-hand sides, this enables fixating on individual conditions and actions. The model `fixates condition s-model-structor`, which is part of a printed-out SP. This condition specifies a type of operator.

The model now has in WM both an operator (s-structor16), and an operator type appearing as a condition (s-model-structor). These pieces activate an imagine subgoal that puts them together (rule 3). Given that the operator type appears as a condition, the model imagines that the operator itself does also. This is a reasonable guess, given the idioms used in the program (discussed in the footnotes on p. 49).

Imagining condition s-structor16 activates the episodic memory for having seen it (rule 1). Having recalled seeing a condition, the model selects a goal to comprehend it (rule 4). The model scrolls, because the condition is hidden (rule 5).

t834: I think all those chunks I built test for {scroll window up} test for operator six {scroll window up} uh, "s" whatever it is, I think they test for the s-structor {scroll window up} let's see shall we, yeah, s-structor16, there it is

Summary: The model imagines a feature based on its program knowledge about likely contexts for that feature to appear in. Two related elements of information on the display, the s-model-structor condition and the s-structor16 operator, trigger an inference that puts them together. The inference in turn retrieves an episodic memory for having seen an s-structor16 condition, which leads to the scrolling event.

4.7. Summary and discussion: images, episodes, and program state

The thesis is that access to external information is a function of both expertise in the task domain and episodic knowledge about what features have appeared on the display. This section reviews the three examples of how the model uses images generated from expert knowledge to search for memories of hidden features.

In scrolling event 1, the model deliberately imagines asserted SPs that it might have seen. These are a clue to the program's current execution state, because they might have fired, and can be examined for their effects. Scrolling to a recalled assertion is worthwhile if only because it gives access to the SP name, which in turn makes it much easier to print the SP to see what it does. As it happens, in this scrolling event the SP is already printed out, and is redisplayed when the model scrolls back to the assertion. In either case, navigating to an asserted SP is worthwhile. Searching for memories of assertions when those memories would be useful requires program knowledge about what SPs modify what data structures, and more general language knowledge about how state changes cause chunks to be built.

In scrolling event 3, the model is comprehending an object (an utterance model) and deliberately imagines a key attribute of that object (the referent). Like the existence of old assertions, the existence of a referent provides a clue to the program's current execution state. In this case, the model recalls having looked for a referent, and uses this information to emulate the scrolling action of the programmer. Trying to recall seeing pieces of an object under construction requires knowledge of what the complete object looks like, and roughly when the program builds it.

In scrolling event 5, the model sees first an operator (s-structor16) and then a condition testing an operator type (s-model-structor). It imagines whether the operator itself might appear as a condition somewhere. Like old assertions and attributes of objects under construction, chunk conditions are important state information. The syntax of chunks is often difficult to predict, and searching for memories of conditions is a way to stay aware of what the program's chunks actually look like. Generating likely conditions requires knowledge of both the chunk-building algorithm and the program-specific idioms that manipulate it, as well as tactical knowledge that generating such conditions is useful for monitoring chunks.

In all three scrolling events, the model deliberately tries to recall features it might have seen. The episodic information it searches for could be useful to an expert programmer trying to comprehend a program, both because it bears on the program's current state, and because it points to hidden contexts of relevant information. By telling us what we've seen, long-term episodic knowledge may help us make many kinds of inferences, beyond the inferences that might cause us to return to where the sighting occurred.

Chapter 5

Measures of the model

We have described a set of five navigation events, in which a programmer redisplay hidden information that was generated during her programming session. We then presented a model, and showed how it emulates the programmer's behavior on these five events. In this chapter we present and inspect some measures of the model. Section 5.1 examines the fit between programmer commands and model commands. Section 5.2 presents an accounting summary of the model's rules and rule firings, both pre-loaded and learned.

5.1. Fitting the keystroke protocol

This section evaluates the model's fit to the protocol data over the full lifetime of the model. Because verbal protocols are comparatively ambiguous and incomplete (Newell and Simon, 1972, Ericsson and Simon, 1992), we rely on the overt behavior reflected in the keystroke protocol. Matching this behavior provides a less detailed but much broader evaluation of fit than the comparison of model and programmer behavior carried out in Chapter 4.

5.1.1. The programmer's commands

The programmer issues 50 commands, consisting of commands either to the language interpreter (like `match-set`) or to GNU Emacs to scroll the window (like `scroll-window-up`). Programmer commands (PCs) are shown in right-of-middle column of Figure 26. They are categorized with respect to commands issued by the model, in terms of *hits* (46), *misses* (2), and *disregarded* commands (2).

The programmer makes only two slips (low-level errors) in the keystroke protocol. In both she typed an incorrect keystroke when starting a command, and had to backspace over the keystroke. These slips are identified at the right of Figure 26. They are not included in the total PC count, and we do not account for them in the model.

To simplify the protocol for the purposes of modeling, we grouped programmer commands into semantic clusters. We identified 7 clusters of 2 or 3 neighboring commands that appear to be unified by one identifiable purpose. To group commands by purpose, we examined them in the context of the protocol, the state of the display, and the state of the programmer's program.

unit commands (UC)		compound commands (CC)		model commands		programmer commands (PC)		disregarded slips			
v	v					misses	hits	v	v	v	v
1		match-set	after-selection	match-set		1					
2		print-sp	create-referent	print	create-referent	2					
3		print-operator	o25	print	o25	3					
4		print-object	u20	print	u20	4					
5		print-stack		print-stack		5					
6		run	where-was-i	run	1	6					
		-----		'm',	backspace					1	
7		print-chunk	chunk-128	print	chunk-128	7					
8		scroll	to-sp	scroll-window-up		8					
9		scroll	to-state	scroll-window-down		9					
		-----		goto	prompt						1
10		print-chunk	chunk-129	print	chunk-129	10					
11		match-set	asserted-sp	match-set		11					
12		run	to-builds	run	1	12					
13		match-set	after-builds	match-set		13					
		-----		run	1			1			
14		match-set	after-selection	match-set		14					
15		run	to-builds	run	1	15					
16		match-set	after-builds	match-set		16					
	1	run	to-op-after-builds	run	1	17					
		''		match-set		18					
		''		run	1	19					
		-----		run	1			2			
17		match-set	after-selection	match-set		20					
	2	run	to-end-of-space	run	1	21					
		''		match-set		22					
		''		run	1	23					
18		print-stack		print-stack		24					
	3	scroll	to-object	scroll-window-up		25					
		''		scroll-window-up		26					
		-----		goto-prompt						2	
19		print-object-fresh	u20	print	u20	27					
20		print-object	r9	print	r9	28					
21		match-set	after-selection	match-set		29					
22		print-sp	implement-exhsted	print	implement-exhsted	30					
	4	run	action-and-print	run	1	31					
		''		print	s15	32					
23		match-set	asserted-sp	match-set		33					
24		print-stack		print-stack		34					
25		print-sp	term-s-model-con	print	term-s-model-con	35					
	5	scroll	to-sp	scroll-up-one-line		36					
		''		scroll-up-one-line		37					
		''		scroll-up-one-line		38					
26		print	propose-return-op	print	propose-return-op	39					
	6	run	to-expected-op	run	1	40					
		''		match-set		41					
		''		run	1	42					
		-----		'o',	backspace					2	
27		print-operator	o29	print	o29	43					
	7	scroll	to-sp	scroll-window-up		44					
		''		scroll-window-up		45					
		''		scroll-window-up		46					

total model commands: 34

total programmer commands (w/o slips): 50

Figure 26: Model and programmer commands

We identified two kinds of semantic clusters. *Scroll-to* clusters (3 of the 7) consist of scrolling commands that together redisplay a particular hidden feature. For example, in scrolling event 2, the programmer issues two consecutive `scroll-window-up` commands (hits 25 and 26) that together redisplay the target data structure. Command sequences similar to *scroll-to* clusters were implemented in a previous Soar model that represented visual search in greater detail than ours (Peck and John, 1992).

Run-to clusters (4 of the 7) run the program up to an event apparently anticipated by the programmer. For example, at one point the programmer runs the program until it selects the next operator (hits 17, 18, and 19: `run 1, match-set, run 1`). The utterances indicate that she knows the current operator to be essentially complete, and that she wants to step the program to the next operator.¹¹

5.1.2. The model's commands

The model issues 34 commands, shown in the left-of-middle column of Figure 26. These 34 commands consist of 27 *unit commands* and 7 *compound commands*. Each unit command (UC) maps to one programmer command. Each compound command (CC) maps to one of the semantic clusters we identified among the programmer's commands.

The use of compound commands was a pragmatic choice for reducing the time to implement the model. The 7 compound commands account for 19 programmer commands, reducing by 12 the number of programmer commands to account for. The reduction in implementation effort may have been greater, because an existing command proposal (UC 8: `scroll to-sp`) transferred to two of the compound situations (CC 3 and 5).

5.1.3. Measures of the fit

The 34 model commands account for 46 out of 50 model commands, or 92%. Of the 34 model commands, each has a corresponding command or semantic cluster in the keystroke protocol (that is, there are no false alarms). The 50 programmer commands do not include the programmer's slips, which the model ignores as well.

The model fails to account for 4 programmer commands. These are identified as "misses" and "disregarded" at the right of Figure 26. They are discussed below.

¹¹The relevant utterances begin at t585: "And then I should get reconsiders, yup, and (`run 1, match-set, run 1`) ok fine, so it's going to add the next one". The "reconsider" preferences referred to indicate the end of the current operator. They only appear on the display as a result of the command sequence, meaning that the programmer was acting on an expectation when she referred to them. "It" in the last clause refers to the "next" operator, selected as a result of the command sequence.

5.1.3.1. Missed programmer commands

The model misses two commands. Both are commands to run the program one cycle (`run 1`). The model knows to run the program under some circumstances, but not these.

We have no explanation for miss 1 other than that the model's knowledge must be wrong. The model's knowledge is also wrong in miss 2, but for a more interesting reason. The data suggest a specific way in which the programmer's command knowledge is more contingent than the model's.

In miss 2, the programmer advances her program one more step than the model does before querying the interpreter for status information. Figure 27 shows the circumstances. The programmer has run her program up to an operator selection. She then runs the program an additional cycle (miss 2), and prints the match-set (hit 20). In contrast, the model issues a match-set command (UC 17) directly after the program selects the operator, without first advancing the program another cycle.

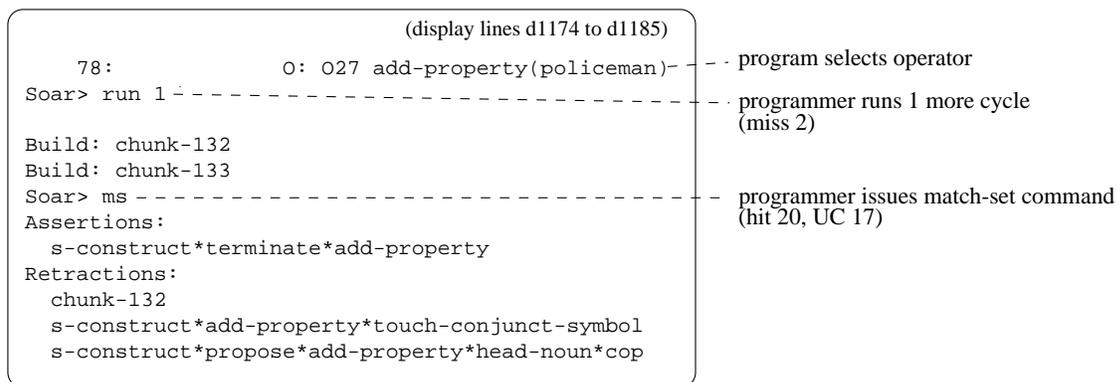


Figure 27: The model fails to issue a run command (miss 2)

The programmer may already know what the match-set looks like after this kind of operator, having seen the match-set before. The program selected the first `add-property` operator earlier, after which the programmer printed the match-set directly. When the program selects `add-property` again, the programmer waits to print the match-set, first running the program one more cycle. The protocol data are unfortunately silent as to whether LTM is involved in bringing knowledge about the first operator's match-set to bear on the second operator. The time between the appearance of the first match-set (t575) and the selection of the second operator (t598) is only 23 seconds.

Miss 2 occurs for at least two reasons. First, when the emulator selects the second `add-property`, the model remembers nothing about the match-set after the first. The asserted SPs from the first match set do not persist in WM, and the model does not retrieve feature memories for them, because existing feature memories for them are tied to a goal that the model does not select in connection with the second `add-property`. (Feature memories are always tied to a goal; see Section 6.2.1.) The second reason for miss

2 is that the proposal for the post-miss match-set command (UC 17) is too general, and would not be inhibited even if the match-set information were in WM.

After both misses, the display emulator compensates. It responds to each post-miss command (UC 14 and UC 17) by generating the output of the missed command, as well as that of the post-miss command.

5.1.3.2. Disregarded programmer commands

The model has no knowledge at all about one programmer command, `goto-prompt`. In the data, both instances of this command return the cursor to the prompt so that the programmer can immediately issue a print command there. The model has no representation of the prompt at all, which means it cannot have a representation of `goto-prompt` commands.

We omitted this representation again to simplify the implementation of the model. The `goto-prompt` command is simply an enabling step for the subsequent print command, and in omitting it we do not omit much by way of expert command knowledge.

Also, while `goto-prompt` is a kind of navigation, it involves a stable feature of the environment, rather than a dynamic one. In modeling `goto-prompt`, we would assume expert knowledge about the programming environment (rather than episodic memories) linking the prompt's absence from the display to its existence as a hidden feature.

5.1.4. Reuse: command-proposal rules

The model shows considerable reuse of knowledge between commands (Figure 28). There are 20 command proposals, of which 10 each account for more than one command being selected. These 10 account for 70% $((14+6+4)/34)$ of the commands selected.¹²

<u>number of command proposals</u>	<u>causing N selections each</u>	<u>total commands selected</u>
10	x 1	= 10
7	x 2	= 14
2	x 3	= 6
1	x 4	= 4
20		= 34

Figure 28: Reuse of command-proposal knowledge

¹²Section C.2 maps model commands to the rules that propose them, and enumerates how many commands each rule is responsible for.

Reused command knowledge is desirable because it points to general methods that the programmer might have, and because it adds constraint. A command proposal that applies in more than one situation requires the model's other knowledge to be able to represent the similarities in the situations.

5.1.5. Summary: constraints from fitting the data

A good fit between model and data reduces the chance that failures to account for some data undermine the model's hypotheses. A good fit to the programmer's command protocol is a pragmatic target that we were able to meet, and also able to measure. The model accounts for 92% of the programmer's commands, from the time that the first scrolled-to feature appears on display, to the last sequence of scrolling commands.

The fit we achieved is also a measure of the completeness of the model's display-command knowledge. For the activities that occur in the modeled interval — running and querying a program and navigating through hidden trace information — the model offers a specification for a core set of methods.

5.2. Rules and rule firings

Rules are a basic unit of representation in the model. They propose and select goals and subgoals, recall facts and episodic information from LTM, and generally represent all the knowledge and mechanisms in the model. This section presents an accounting summary of the model's rules and rule firings, both pre-loaded and learned. We also find reuse of knowledge throughout the model, reducing internal degrees of freedom and increasing confidence in the generality of the model's knowledge.

5.2.1. Rule counts by category of knowledge and mechanism

Figure 29 shows a breakdown of all the rules in the model. There are 1514 rules total (bottom right) when the model finishes executing. Of these, 194 are pre-loaded (knowledge and mechanisms) and 1320 are encoded during execution.¹³

The figure also shows how often rules in various categories fire during execution (rule *firing counts* are in italics). There are 17352 total rule firings, of which 15851 (91%) are from pre-loaded rules and 1501 (9%) are from encoded rules.

The rule categories reflect those introduced in Chapter 3 (see Figure 20, p. 41, and Figure 19, p. 40). Expert knowledge (126 rules) includes proposals for retrieving information from the display (attend and fixate) and from LTM (probe and imagine). It also includes facts about objects, proposals for important objects to comprehend, a small amount of specific knowledge for selecting fixate subgoals, and proposals for commands to change the display.

¹³The display emulator is implemented by a set of 103 pre-loaded rules, not counted as part of the model.

	<u>rule counts</u>	<u>firing counts</u>
Pre-loaded rules		
Expert knowledge		
Attend proposals	2	74
Fixate proposals	31	886
Probe proposals	7 ¹⁴	1094
Imagine proposals	10	228
Facts about objects	28	302
Comprehension-goal proposals	23	153
Fixate preferences	5	28
Command-goal proposals	20	83
Total (expert knowledge)	126	2848
Mechanisms		
Attend subgoal	5	160
Fixate subgoal	5	1419
Imagine subgoal	3	63
Working memory	12	3436
Comprehension-goal selection	5	388
Subgoal selection	25	6083
Command-goal selection	4	151
Shared rules	9	1303
Total (mechanisms)	68	13003
Total (pre-loaded rules)	194	15851 (91%)
Encoded rules		
Feature memories	407	354
Recoded facts	257	62
Image memories	68	6
Episodic memories for regions	62	680
Episodic memories for features	400	271
Goal selection	63	0
Episodic-retrieval/hidden-feature	63	128
Total (encoded rules)	1320	1501 (9%)
Total rules	1514	17352 (100%)

Figure 29: Distribution of pre-loaded and encoded rules, and *firing counts*

¹⁴See Figure 30 for an expanded count.

The model's knowledge shows a balance between knowledge for retrieving external information (33 proposals for attend and fixate) and knowledge for retrieving internal information (31 elements of knowledge, 10 imagine proposals and 21 elements for proposing probes, using the expanded count from Figure 30). One might expect expert performance to involve some balance between internal and external knowledge, when both are available.

Mechanisms (68 rules) implement the information-retrieval subgoals (attend, fixate, probe, and imagine) and the model's limited-persistence WM. The mechanism for selecting comprehension goals uses converging evidence of goal relevance, supplied by subgoals. The subgoal-selection mechanism is based on very general heuristics, like looking at features in regions on the display for the first time, and choosing indifferently among subgoals not discriminated by other knowledge. The mechanism for selecting command goals acts to make new external information available as soon as display-command knowledge deems it relevant.

Shared and other rules (9) contribute to multiple mechanisms. One rule creates a timestamp unique to each goal, used by the fixation mechanism to tag features (Section 3.4.2.1, by the attention mechanism to tag regions (Section 3.4.1), and by the imagining mechanism to tag images (Section 3.5.2). Other rules maintain the set of probes tried during each goal, which affects fixation and in turn goal selection (Section 3.6.3).

Mechanisms account for only 1/3 of pre-loaded rules (68/194), but account for 3/4 of total rule firings (13003/17352). This is consistent with their role as a layer of generic functionality through which the architecture manipulates the model's knowledge.

Of the model's encoded rules, most are feature and episodic memories arising from fixate subgoals (407 and 400, respectively). The model recodes many facts (257), and encodes some memories for imagined features (68). The model also encodes a rule whenever it selects a new comprehension goal (63). The model renders this rule effectively irretrievable, as otherwise the rule could immediately replace the current goal if it were selected again. This immediate replacement would prevent further information-retrieval in service of that goal, defeating the purpose of comprehension. Finally, the model encodes a rule when it retrieves an episodic memory or notices that a feature in WM has become hidden (63). We treat these rules as artifacts of the architecture's universal learning, and do not try to interpret their significance.

The rule-firing data for encoded rules suggest that information about the particular session plays a small but critical role in task performance. This information is captured in memories for the display — feature memories and episodic memories for regions and features. These session-specific memories account for 1305 (8%) of total rule firings.¹⁵

¹⁵354+680+271=1305; 1305x100/17352=8%

5.2.2. Reuse: firings compared to selections

Rules translate into behavior when the architecture selects proposed goals and subgoals. Figure 30 shows how many selections of each kind of subgoal and goal occur in the lifetime of the model (rightmost column). It also shows how many rules propose each kind of subgoal and goal (left-of-middle column). It relates these two columns by the *reuse factor* of the proposal rules, indicating on average how many selections one proposal rule accounts for (right-of-middle column).

Reuse of knowledge is considerable. Earlier we examined reuse of command-proposal rules (Section 5.1.4), finding that half of them accounted for more than one command selection each. Figure 30 shows that each command proposal accounts for 1.7 command selections, on average. The reuse factor is much higher for the rest of the model's knowledge.

For example, 7 probe-proposal rules account for 181 probe selections. This is somewhat misleading, because one of these proposal rules reads a 15-entry table describing what objects in WM to probe with (the table is created by a rule included in the detailed trace, on p. 183). Counting this table in place of the rule that interprets it, 21 elements of knowledge account for 181 selections, still leaving a reuse factor of 8.6. Aggregated, each proposal rule accounts for 5.4 goal or subgoal selections, on average.

The high degree of reuse throughout the model reduces degrees of freedom, by reducing the number of variables in the model that can be tailored to account for the programmer's behavior. This increases confidence that the model's knowledge is a meaningful approximation to that of the programmer.

<u>goal or subgoal</u>	<u># proposal rules</u>	<u>reuse factor</u>	<u># selections</u>
attend	2	x16.5	33
fixate	31	x4.3	134
probe	7 (21)	x25.9 (x8.6)	181
imagine	10	x2.5	25
comprehension goal	23	x4.0	92
command	20	x1.7	34
total	93	x5.4	499

Figure 30: Proposal rules vs. selections for goals and subgoals, showing reuse

5.2.3. Summary: numbers as confirmation

The numbers presented above help confirm properties that we have attributed to the model. Three of these properties are illustrated in Figure 31, which compares major categories of rule and firing counts after a run of the model. First, 87% of the model's total rule count is made up of learned rules, suggesting that learning is in fact pervasive. Second, mechanisms account for 5% of total rules but 75% of total rule firings, consistent with their role as providing a generic layer of cognitive functionality. Third, learned session-specific rules (feature memories and episodic memories for regions and features) account for 8% of firings, indicating a small but key effect of session on the model's behavior.

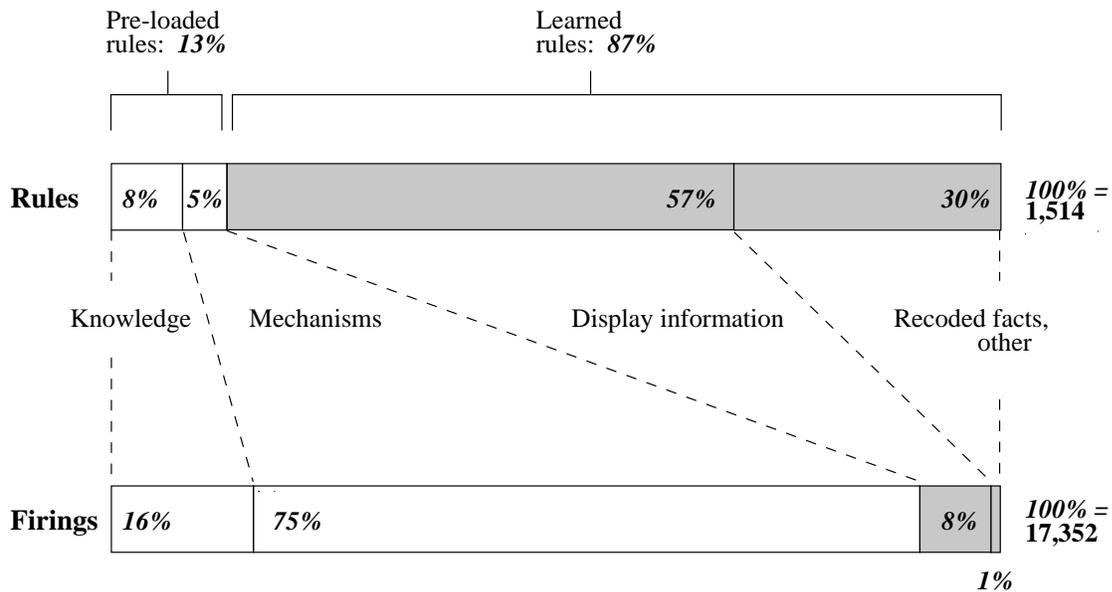


Figure 31: Comparison of main categories of rules and firings

The numbers also show that knowledge reuse throughout the model is high (Figure 30), which is evidence that the model contains a much better approximation of the programmer's expert knowledge than an arbitrary program that mimics her behavior.

Chapter 6

Further characterization of the model

The previous chapter characterized the model with quantitative data. This chapter touches on a number of different points of discussion. Section 6.1 reviews the various roles of knowledge in bringing about navigation. Section 6.2 revisits what the model actually learns. Section 6.3 reviews our motivation for limiting WM persistence to a duration of 2 goals. Section 6.4 examines the issues of purposefulness and flexibility in modeling an extending train of thought. Section 6.5 addresses the limitations of the model's submodel of comprehension, and reviews why the model does not learn to imagine features. Section 6.6 compares our model to another that learns about external features to discover how they are complementary. Section 6.7 speculates about how the model might be extended to have a more realistic world model, and about the implications for our hypotheses about episodic memory. Section 6.8 examines implications of the model for the design of complex interfaces. Finally, Section 6.9 summarizes the key lessons learned from each of these points of discussion.

6.1. The chain of knowledge that leads to scrolling

The model's episodic memories map a feature in WM to a timestamp. We argue that these simple memories are one key element of purposeful search for external information. The other elements of this search consist of domain knowledge, used to search internally for episodic memories and then to act on them. Here we examine where domain knowledge comes into play in the model's navigation, and speculate on how experts might differ from novices along this dimension.

Knowledge comes into play indirectly at encoding time. Fixate proposals determine what features the model will process, and hence what features it encodes episodic information about. They embody domain knowledge that perceives meaningful features, ranging from language-general features like SP conditions (po*fixate*condition, p. 157) to program-specific features like the absence of a key attribute (po*fixate*no-referent, p. 158).

Knowledge has two roles at retrieval time. First, the model must be able to imagine features. This is an act of recall, and requires greater familiarity with features than simple recognition. In implementation terms, the model must have rules whose actions propose imagine subgoals. Second, the model must be able to imagine a feature when memories about that feature would be relevant. For example, in scrolling event 3 (Section 4.4) the model imagines a referent attribute when comprehending a utterance model. The existence of a referent attribute on a utterance model is an index of how far the program has run. Some

understanding of the relevance of a feature is necessary to make imagining it worthwhile. In implementation terms, the imagine proposal rule must have appropriate conditions, and the model must be able to create a WM context in which those conditions are met. Later, in Section 7.2.2, we propose that model actions leading up to the activation of an imagine proposal constitute a *retrieval structure* consisting of domain knowledge.

Finally, knowledge plays a role in acting on a retrieved memory by proposing a scrolling command. Having recalled the existence of a feature, the model scrolls to reveal the context of that feature. Scrolling is thus based on knowledge about what contexts or *settings* (Tulving, 1983) are important under what circumstances. The model's scrolling knowledge may be underconstrained, an issue we return to in Section 6.1.2, which considers why the model does not scroll more often.

6.1.1. Where expert-novice differences might reside

We have no novice scrolling data to compare to our expert data, making it difficult to say specifically how different levels of expertise affect encoding, retrieval, and navigation. However, we can look to the model and the data at each of the stages discussed above and ask how specialized the knowledge is. At the fixation stage (continuing with the example of scrolling event 3), the programmer notices the absence of a referent on the utterance model. This perception seems to require considerable familiarity with the program.

At the retrieval stage, it seems plausible that someone not experienced with the specific symbols of a specific program could have trouble imagining them in a way that would trigger recognitional memories. The model says little, beyond specifying the need for imagine proposals and the ability to activate them. Its simple representation does not allow for partial match between imagined and fixated features, nor do we have an account for how it might acquire the ability to imagine a feature (an issue we return to in Section 6.5.2). There is a clearer case that expertise can be involved in determining the relevance of imagined features. It would require considerable familiarity with the program to know to measure the program's progress by imagining a referent attribute.

At the scrolling stage, the model has scrolling that seem general to interaction with a language interpreter in a buffer. For example, the model scrolls to a goal object when it recalls looking for an attribute of that object (po*display*scroll*to-object, p. 149). A novice to a given program or language might inherit this kind of rule from another language. On the other hand, this rule might well need more specific conditions if the model covered a wider range of behavior.

6.1.2. Why the model scrolls so little

If episodic knowledge is encoded as ubiquitously as the model claims, the question arises as to why the model does not also imagine more often, and why it does not scroll more often as a result. The best answer we can give is that the programmer may imagine much more often than the model does. However, the

protocol contains no clear evidence of imagining when this imagining fails to result in scrolling. We gave the model close to the minimum of imagining knowledge needed to emulate the programmer's scrolling behavior.

Figure 32 shows data on the model's imagining behavior. The model selects 25 imagine subgoals total (bottom right). Of these, 14 would retrieve an episodic memory were the model to fixate on the corresponding feature beforehand. The rest imagine features that the model has no knowledge to fixate.

Of the 14 imagine subgoals that could retrieve episodic memories, only 3 actually do (not shown in the table). In all 3 cases, the model scrolled. This suggests that the model's scrolling rules can be as weakly constrained as they are only because the model retrieves the minimum amount of episodic knowledge by imagining.

<u>rule</u>	<u>fixatable?</u>	<u>proposed</u>	<u>selected</u>
po*imagine*nil-object		11	4
po*imagine*postpone		4	1
po*imagine*operators	y	4	1
po*imagine*operator-targets		6	3
po*imagine*assertions	y	16	4
po*imagine*action	y	177	6
po*imagine*actions-refract		1	1
po*imagine*sp-causes-builds		4	2
po*imagine*referent	y	3	1
po*imagine*s-model-constructor	y	20	2
	total:	246	25
	total fixatable:	5 (50%)	14 (56%)

Figure 32: Imagine subgoals and their intersection with fixate subgoals

Imagining has a second purpose that is unrelated to retrieval of episodic memories. (This explains why there are images that cannot be fixated.) Images can provide converging evidence of relevance for a proposed goal (Section 3.6.1). For example, in scrolling event 1 (Figure 21, p. 45) the model imagines the superstate context. The model is aware of a chunk, but is not aware of the problem space in which that chunk was built. Under these circumstances, it imagines the superstate as a candidate (po*imagine*operator-targets). The evidence from the protocol is the programmer saying, "So it must have just changed it on the superstate?" (Figure 6, p. 14). The model has independent knowledge that says that superstates are important to comprehend, and has already proposed the superstate goal when the imagine subgoal is selected. The imagine subgoal matches this proposed goal, and the model selects the goal to comprehend the superstate.

6.1.3. Summary: knowledge mediates access to external information

There are four links in the chain of knowledge that leads the model to scroll. These are illustrated below with respect to scrolling event 3.

- Fixation: 1. Knows to look for referent
model encodes episodic memory
- Retrieval: 2. Able to imagine referent
retrieving episodic memory
 3. Knows to imagine referent now
- Scrolling: 4. Knows to visit referent's setting

This chain illustrates our hypotheses about episodic information. Encoding is passive, a side effect of noticing a relevant feature. Retrieval is deliberate, the effect of calling to mind a relevant feature. This chain also points at which domain knowledge mediates scrolling, and hence where experts may have an advantage in gaining access to external information.

6.2. What the model learns

The model operates under the constraint that all information retrieved to WM is also encoded in LTM. The constraint results in a lot of encoded rules, and raises questions about what the model really learns. This section analyzes three kinds of rules learned by the model: feature memories, recoded facts, and episodic memories.

6.2.1. Feature memories

A feature memory caches a feature, allowing the model to heed it more quickly in the future (Section 3.4.2.1). In machine-learning terms, it *operationalizes* knowledge for perceiving and fixating the feature (Mitchell et al., 1986). This leaves open the question of psychological plausibility.

Feature memories are *display-based* (Larkin, 1989), in that they are activated by external cues. The learning of such rules is not a Soar-specific artifact (Howes, 1994, Rieman et al., 1994, Kitajima and Polson, 1995). However, Soar itself makes strong claims about the irrevocable ties of display-based rules to the external world (Howes and Young, 1996a). In our model it is impossible to get such rules to fire using internal information only. This is equally true of fixation proposal rules themselves. Thus feature memories serve only to retrieve external information more efficiently, conferring no advantage over fixation proposals when the display is absent. In particular, feature memories are not how the model would learn to imagine features in its mind's eye. Feature memories are display-based in the most literal sense, contributing to the ability to recognize but not to the ability to recall.

The recognition capability of feature memories is itself limited. The model ensures that every feature memory is encoded to depend on a specific goal (the one current when the memory was encoded). This prevents the model from over-learning the display to the point where every feature enters WM in parallel as

soon as the display is generated. Tying a feature memory to the object for which the feature was retrieved retains a plausible bottleneck in the process of retrieving information from the display.

Feature memories contribute only a limited efficiency improvement to the retrieval of information. This improvement is critical, however, because it saves the model from having to start accumulating information from scratch whenever it selects a given goal. Feature memories thus enable incremental comprehension of objects. In Section 6.6.1 we compare this behavior to similar behavior in another model, in a domain where incremental comprehension is more directly tied to observable behavior.

6.2.2. Recoded facts

Recoding expert knowledge makes such knowledge more accessible the more it is accessed. A finding consistent with this in the area of programming psychology comes from experiment conducted by Davies (1994), who found that programming experience interacted with speed and accuracy in recognizing *focal lines* of code (Rist, 1995). A focal line directly reflects a programming goal. For example, in a program that requires a running total, $total := total + current$ would be a focal line. In Davies's experiment, recognition speed and accuracy were not significantly different between novices and intermediates. However, novices and intermediates were significantly slower and less accurate than experts.

Davies concludes from these findings that improved recognition for focal lines was a non-linear function of programming experience. The difficulty of controlling for expertise knowledge (Sheil, 1981) makes it difficult to determine whether the differences in levels of knowledge justify Davies's conclusion. However, his conclusion does seem consistent with our model, in which retrieving a fact links that fact directly and permanently to an additional goal, making it more likely that the fact will be retrieved again. Access to expert knowledge thus seems to reinforce itself automatically. We revisit the model's use of recoded facts in Section 7.2.4, with an example.

6.2.3. Episodic memories for features

The model's episodic memories, which associate features with a simple timestamp, are the minimum necessary to capture the knowledge that something was seen before. This knowledge in turn is the minimum needed to decide to navigate in a directed manner to hidden information.

A general episodic memory would allow recall of much more complex declarative information. For example, Jeffries et al. (1981), in their study of software design, found that experts used episodic memory to recall outstanding design questions. This is presumably a much more productive recollection than the existence of a hidden external feature. Similarly, the problem-solving method of *progressive deepening* (Newell and Simon, 1972) involves recreating previous cognitive states for the purpose of integrating new information. For example, as applied in algorithm design (Kant and Newell, 1984, Steier, 1987), progressive deepening involves mentally simulating a computation to determine where and how it is under-specified, then "resetting" the mental run with the new information integrated into the mental

representation of the program. To the extent that people recall previous states internally, without the use of external information, they must encode complex, temporally-coherent information in LTM.

The model's episodic memory is not so limited that it serves only the purposes of navigation. For example, in scrolling event 4 (Section 4.5), the model sees two SPs and chooses to comprehend the novel one first. The model knows which one is novel because an episodic memory tagged the other one as having been seen before. Thus an episodic memory activated incidentally (without deliberate search of LTM using images) to provide useful information.

6.3. Why a 2-goal persistence limit on working memory?

A model of LTM scrolling events depends for plausibility on a limited-span WM, one that can't simply store information about a hidden feature until the model has a use for it, whenever that might be. To be forced to avoid implausible uses of WM, we implemented a limitation on WM, making up for Soar's lack of architectural constraint (Section 3.5.4). While our limit on WM persistence is simplistic and probably artificially short, it is still worth examining in terms of constraints it does satisfy.

Two goals is sufficiently short, in that it forces the model to retrieve episodic information from LTM in cases where the data suggest that the programmer does (scrolling events 1, 3, and 5). Two goals is also maximally short, in that a 1-goal persistence would prevent the model from having any WM context for its current goal (Section 3.5.4).

Though maximally short, two goals is workable, at least for what our model does. The model continually reconstructs WM with information from the display and LTM. Moreover, by encoding feature memories and recoding its expert knowledge, the model is able to retrieve increasing amounts of information at once. The model thus expands its WM capacity through experience and the use of LTM (we elaborate on this point in Section 7.2.4).

A more specific test of whether two goals is long enough to be workable is whether it lets the model account for short-term scrolling events. In two scrolling events (2 and 4), the programmer could have maintained episodic information in WM, rather than retrieving it from LTM; the scrolled-to screen is hidden to the programmer for less than 15 seconds before scrolling. In both cases, the two-goal span is sufficient to allow the model to maintain in WM the feature it scrolls to.

It seems worth noting that a limit of two on elements of the same kind provides for local comparison, and that this limit surfaces elsewhere in models of cognitive processes. The weak method of hill climbing (Laird, 1984, Rich and Knight, 1991) maintains two states, using the current state to evaluate the next. Lewis's language-comprehension model (Lewis, 1993) produces accurate predictions using bounds of two. Sentences with two elements per syntactic role (for example, two embedded relative clauses) can be parsed, but those with more induce parsing breakdown. It may be that a limit of two is somehow adaptive in providing for essential local comparison. Our model has no processes that carry out comparisons between WM elements. However, a comparison process might be able to exploit the association between an

element and the goal during which it was retrieved, to keep otherwise-identical elements distinct. Thus a persistence of two might provide for essential local comparison in our model as well.

6.4. Goal selection for purposeful yet flexible behavior

One of the challenges in modeling an extended interval of interactive behavior is to emulate a coherent but flexible train of thought, guided both by internal and external knowledge. In our model, the difficulty lies in deciding when to select a new comprehension goal to replace the current one. Goal selection should be both purposeful and flexible at the same time.

To characterize the purposefulness of the model, we calculate the duration of the model's goals in terms of programmer time (Section 6.4.1), to get a sense of whether model goals span roughly the amount of time that people spend on mental operators. We characterize flexibility with respect to how the model's train of thought can be influenced by newly-retrieved knowledge (Section 6.4.2).

6.4.1. Goal duration in terms of programmer time

Part of purposefulness lies in adhering to a goal and not being too easily distracted. The converging-evidence requirement for goal selection is adapted for this purpose (Section 3.6.1). Without it, the model would select goals as soon as they are proposed, which is much too frequently. Another way to ask whether the duration of a goal is long enough is to compute how long each goal "lasts" in terms of programmer time.

The model's lifetime spans 629 seconds of programmer time (Figure 33). The model selects 499 goals and subgoals, or one every 1.3 seconds.¹⁶ This corresponds to the roughly one second that Newell allots to one complex cognitive operator (Newell, 1990). This suggests that the model behaves at a sufficiently fine grain for representing complex information processing.

The model selects 92 comprehension goals in total, or one every 6.8 seconds of programmer time. Newell and Simon (1972) cite an average duration of 8 seconds for mental moves in the Logic Problem. Based on this and observations of human behavior in other tasks, Newell (1990) places the duration of the *composed operations* at roughly 10 seconds. The level of composed operations is the level at which decisions are made about what step to take next. Goal selection falls into this level, and a 6.8-second span is within range. From this perspective, the model's goals on average last an appropriate length of time.

¹⁶The 499 total goals and subgoals accounts for all the model's deliberative acts, but the model actually runs for 569 Soar decision cycles. The difference of 70 (569-499) is because the decision-cycle count includes Soar impasses. The 70 impasses are operator no-changes that occur on the Soar operators that represent comprehension goals. Information-retrieval subgoals for a given comprehension goal occur in the impasse for that goal.

<u>goal or subgoal</u>	<u>selection counts</u>	<u>avg. programmer seconds between selections</u>
comprehension	92	6.8
command	34	18.5
attend	33	19.1
fixate	134	4.7
probe	181	3.5
imagine	25	25.2
total	499	1.3

duration of covered protocol: 629 seconds

Figure 33: Goal and subgoal frequency in terms of programmer time

6.4.2. Flexibility through goal-independent subgoals

Flexibility lies in the ability to leap to thinking about something only tenuously related. The model achieves flexibility in part through a combination of goal-independent knowledge for retrieving new information, and goal-termination criteria defined in terms of new information.

The model has goal-independent knowledge about what objects are important to fixate on, imagine, and comprehend. For example, almost all the model's fixation knowledge is goal independent to some degree. Of the 31 rules that propose fixate subgoals, 17 are goal independent, and 13 require only that a particular goal belong to the union of the current goal and the set of probes selected for the current goal (Section 3.6.3). The longer the model spends in a subgoal, the more probes it selects, and the further afield it will look for information. This systematically increases the number of subgoals proposed for a given goal, which also increases the number of possible next goals. The longer the model spends on one object, the more likely it will be to see another it wants to think about next.

In many AI domains the goal state is well-defined. In contrast, our model implicitly decides when the current goal is "done", by selecting the next one. This seems to depart from the formulation of the problem space computational model as containing an explicit goal state (Newell et al., 1991). However, it is consistent with the method of progressive deepening, in which the criteria for terminating a search path involve the generation of new information (Section 6.2.3). It is also consistent with observations that students show goal-selection patterns that are more flexible than the last-in first-out order specified by many problem-solving systems (VanLehn et al., 1989).

This implicit representation of goal completion may reflect the kind of activity that occurs in the segment of behavior we studied. The programmer wants to understand her program. Gaining understanding is a task with an ill-specified goal state, defined mainly in terms of relevant information accumulated. It seems appropriate that the goal selection methods that serve in modeling this behavior have no explicit goal-

success criterion. Extending the model to cover planning and generation of code may require methods for explicitly evaluating the match of intended situations, represented in the model's head, to actual situations in the environment. (We return to the model's limited representation in Section 6.5.1, below.)

Independent sources of knowledge have to converge on a goal before the model will select it. This keeps the model thinking about the current goal. Convergence is a function of general knowledge about what information to retrieve. This general knowledge, and the increasing tolerance for any new goal as work on the current goal continues, support flexibility in choosing the next goal.

6.5. Limitations of the model

The model makes its hypotheses in the context of an interactive programming task. As an actual model of programming and interaction, it is quite limited. This section examines two limitations in particular. First, the model's submodel of comprehension does not explain the memory structures used for other subtasks of programming. Second, though the model learns, it does not account for the learning involved in transforming familiarity with external features into the ability to imagine them in the mind's eye.

6.5.1. Comprehension limited to information-gathering

While the model selects goals that ostensibly "comprehend", its comprehension processes do not build the kind of structures generally associated with comprehension of programs (Pennington, 1987a, Pennington, 1987b, Brooks, 1983) or text in general (Lewis, 1993, Kintsch, 1988, van Dijk and Kintsch, 1983). The model's comprehension processes essentially retrieve unstructured information to WM.

The model's comprehension output is impoverished to the extent that it fails to allow for a plausible account for some utterances. For example, in scrolling event 2 (Section 2.3.2), the programmer appears to compare what she remembers of the now-hidden screen to the now-visible screen. She notes a similarity between the two screens ("that *also* tests for construction done", emphasis added). The model retrieved the construction-done attribute to WM from the now-hidden screen, but this prevents it from fixating the attribute again on the now-visible screen. The model's representation is too coarse to represent the differences between the two elements. Moreover, the model has no process for comparing arbitrary elements of information.

Apart from a few examples like this, the segment of behavior covered by the model provides little constraint that would help to specify richer comprehension structures. The behavior seems adequately modeled by a pattern of gathering information about an object, information which then leads to consideration of the next object. The data beyond this segment do contain coding and editing behavior. Extending the model to account for these data would not only broaden it to other subtasks of programming, but would also constrain the model's comprehension structures to be functional inputs to coding and editing and processes. The model would have to apply the information it accumulates.

Other computational models of programming could provide guidance for extending our model to generate

and use richer knowledge structures. Brooks's (1975, 1977) model generates code from structures posited as the output of task understanding and code planning. Brooks (1983) elaborates on this model, with an analytical theory of how program functionality and structure interact with expert knowledge to produce comprehension behavior. The theory predicts that comprehension proceeds by hypothesis refinement and revision, and produces knowledge structures that link task knowledge to programming constructs. Rist's (1995) *Zippy* model carries out both program design and coding. Its designs are guided variously by knowledge of control-flow constructs and program functions, means-ends analysis applied to focal lines of code, and opportunism.

6.5.2. Images limited to being pre-loaded

Given that Soar is a learning architecture, the question arises as to how the model could learn more of what it knows (how the *density* of learning in the model could be increased, relative to the *diversity* of potential learning targets; Altmann, 1993). In particular, the primary activity explained by the model involves fixation and imagining of features. A more compelling account of these processes would connect the two, explaining how familiarity with fixated features eventually leads to the ability to imagine them.

The general form of the process for converting features to recallable images is dictated by the architecture and its learning mechanism. In Soar, learning to recall knowledge requires reconstructing the to-be-recalled element in WM from other recalled knowledge.¹⁷ The difficulty lies in the recursion inherent in this process. This appears to be an instance of the *symbol-grounding problem* (Harnad, 1990) of how to map external features to internal symbols that allow for fully-general cognitive processing. Soar models have not progressed beyond initial investigations of the use of internal symbols and internal-external mappings (Mertz, 1995).

A model of how we acquire images could have important implications, for example in the design of interfaces. The model predicts that one factor in effective access of external information is the ability to recall features. This in turn means that environments should be designed in order to make features easy to learn to imagine, a process that a model could shed light on.

6.6. Comparison to IDXL

As we mentioned in Chapter 1, there are other computational cognitive models that learn about hidden information in a computer interface. These include IDXL (Rieman et al., 1996), which models exploratory learning of interfaces; a predecessor of IDXL called Ayn (Howes, 1994); and several models that learn task-action mappings (Howes and Young, 1996b, Kitajima and Polson, 1995, Mannes and Kintsch, 1991, Lewis, 1988). IDXL in particular shares two kinds of behavior with our model: incremental comprehension, and the encoding and retrieval of episodic memories for external information. We review these two similarities here and examine how the models complement each other.

¹⁷Section B.5, p. 120, gives a simple example of the Soar process of *data chunking*, by which Soar learns to recall symbols. The example illustrates the ideas referred to in this paragraph. Section 8.1.1, p. 108, speculates on how data chunking may be used to model the encoding phase of skilled memory.

6.6.1. Incremental comprehension

The first similarity is *incremental comprehension*. Each model comprehends an external object by returning to it on successive occasions, recalling at each occasion what it learned previously and then comprehending something new about it. Our model incrementally examines the features that make up an object (Section 6.2.1). IDXL incrementally comprehends menu items. For a given menu item, it tries on successive occasions to predict whether that item carries out a specified task. The incremental comprehension in our model seems plausible, and arises naturally from other mechanisms in the model, but there is no direct evidence in the data as to whether the programmer comprehended objects in this manner. In contrast, incremental comprehension in IDXL reflects users' behavior directly.

In the data modeled by IDXL, users were given simple tasks to carry out in a given menu-based software package. Menu structures were familiar to users, but the software was not. Users were found to iterate their search through menu items, spending more time per item on successive passes, until they found a menu item they considered appropriate for the task. Processing on each subsequent visit to a menu item was apparently deeper, implying some recollection of knowledge retrieved during past visits. In these data, the evidence for incremental comprehension is in the motor commands that effect the menu search. In our data, motor commands are too sparse to reveal any patterns that might suggest incremental comprehension.

IDXL is also implemented in Soar, and inherits the same architectural provisions of passive, pervasive, and recognitional learning that our model does (Section 3.3.1). When IDXL visits a menu item, it caches the results of comprehension in *recognition chunks*. These correspond to our feature memories in being tied to the presence of cues on the display. When a recognition chunk fires in the future, it might tell IDXL to "think harder" about a menu item — to attempt comprehension again, starting with the knowledge retrieved by the chunk.

For example, in one run of IDXL, its task is to draw a chart. The interface contains no `chart` item, but does contain a `line` item (in the menu for drawing graphs). The first time IDXL sees `line`, it rejects `line` as not equal to `chart`. From this it encodes a rule that recognizes `line` and retrieves `not chart`. The second time it comes across `line`, having failed to find `chart` elsewhere, the recognition chunk fires immediately, saving IDXL the step of comparing `line` to `chart`. However, like our model, IDXL also knows to do some deliberation in service of a comprehension goal, and thus thinks a little harder about `line`. It probes its memory, recalling that `chart` and `line` are synonyms. It uses this recollection to decide to select the `line` item. The benefit of incremental comprehension as a strategy is that it first applies low-cost, high-value assessments, like comparing a menu item to symbols in the task description.

Our model engages in a very different form of behavior than IDXL. It spans one long, continuous interval of subject behavior, rather than an abstraction of aggregate behavior. It selects many goals and fixates many features in ways that are not well-defined by the kinds of methods that govern interface use. The model also processes a lot of knowledge, both internal and external. The data is mostly verbal protocol, which lacks the precision that would be necessary to discriminate between increments of comprehension. These differences all reflect differences in the data, which in our case is from an instance of knowledge-intensive problem solving rather than aggregation of comparatively knowledge-lean interface explorations.

The effect of these differences is to make it much harder for us to measure how well incremental comprehension reflects our subject's behavior. In contrast, the task environment for IDXL ties incremental comprehension directly to motor behavior and screen changes. The user returning to a menu item is an observable event. Thus the case for incremental comprehension in IDXL's task environment is quite strong.

Because incremental comprehension in the two models is quite similar, its success in IDXL provides some support for its role in our model, and hence in accounting for comprehension during knowledge-intensive display-based problem solving. It may be that one way people concentrate on external information is by cycling back to it, each time retrieving increasing amounts of knowledge about it. In Chapter 7 we discuss other ways in which long-term memory mediates working information.

6.6.2. Encoding and retrieval of episodic knowledge

In Chapter 1, we said that models of interface exploration left open the question of what people might learn about external information more dynamic than menus. Comparing IDXL with our model suggests how the contents of encoded memories might vary with how dynamic the information is: the more numerous features are, and the less directly related to the task at hand, the less information will be encoded about them. It also suggests how the ability to retrieve episodic memories with internally-generated cues could account for directed search through a menu hierarchy.

In our task environment, the model cannot know which of the many features it fixates on might be relevant later and therefore merit more elaborate encodings. The least amount of processing that leads to a retrievable memory seems an appropriate investment of cognitive effort. Consonant with this, the model encodes only a symbol denoting the extant goal. This seems in some sense minimal, denoting no more than the occurrence of an event at a given time.

In contrast, in the interface-exploration task environment, there are comparatively few features; each is a potential solution to the current task; and each is a persistent part of the environment that may be a solution to some future task. These differences imply a greater cognitive investment per feature. Consonant with this, IDXL stores, for example, an assessment that a menu item (`line`) does not identically match the task symbol (`chart`). While this is not a complex assessment, it seems closer to what we might think of as semantic. Thus in these two models, degree of semantic processing of features, and hence semantic richness of the encoded memories, reflects the potential importance of a feature with respect to the task. In Chapter 7 we return to the influence of task demands on how information is encoded, as we examine our model as an instance of long-term working memory (Ericsson and Kintsch, 1995).

Our model is unable to compare features on display with each other (Section 6.5.1), or with internally-generated expectations. Were it extended with a symbol-matching process like that of IDXL, it would encode rules that captured the resulting assessments. Such rules might have semantic content comparable to IDXL's recognition chunks.

The contribution of our model relative to IDXL is that it can retrieve episodic knowledge with both internal and external cues. IDXL must encounter an external feature in order to retrieve knowledge about it. Our model can imagine features beyond the narrow range of what is visible, and thus gain access to a much broader field of external information. In terms of interface use, our model would account for behavior in which a user recalls seeing some menu item that seems relevant now. Recalling a relevant hidden item may lead the user to search in a directed manner for that item, complementing the exploratory search modeled by IDXL.

6.7. A speculative extension to a more detailed world model

One direction in which our model simplifies drastically is in its world model. Beyond the ability to distinguish hidden from visible, it has no spatial knowledge to speak of. For example, where the programmer issues directional scrolling commands — up or down — the model simply issues scrolling commands. Also, these scrolling commands do not specify stopping criteria, nor does the model have perceptual processes that detect when a command has succeeded. These simplifications are consonant with others in the model, for example the subsumption of perception in the conditions of fixate proposal rules. However, they remove our model somewhat from more detailed HCI models, in particular other Soar models like Rieman et al.'s IDXL, Bauer and John's video-game model, and Nelson et al.'s model of the NASA Test Director. Extending our model in this direction could give us a more detailed picture of how episodic memory functions in human-computer interaction. Here we speculate about such an extension, with the additional purpose of assessing whether the model's account of episodic memory somehow depends on its simplifications.

In scrolling event 3 (Section 2.3.3), the programmer is reminded of the utterance model being constructed by the program. She then scrolls to a hidden copy of this utterance model in an earlier state of construction. She notices the identifier of the scrolled-to copy, then returns directly to the prompt to print a fresh copy, using the identifier.

In the model's account, scrolling to an old copy of the utterance model is incidental to trying to determine how much of the utterance model has been constructed (Section 4.4). Figure 34 shows the steps in greater detail, beginning with the model's behavior just after the model has been reminded of the utterance model being constructed by the program. The model selects the goal of comprehending the utterance model (line 1 in the figure). In service of this goal, it imagines a referent on the utterance model (line 2). The explanation for this that is consonant with the model is that an episodic memory for having seen the referent would mean that the referent exists, providing information about the progress of construction on the utterance model. The model then scrolls, based on a heuristic that says to visit the setting of a feature that we've invested the time to imagine (line 3). After scrolling, the model reselects the comprehension goal from before scrolling (line 4), and fixates on the scrolled-to identifier (line 5), consistent with the reason for scrolling. It then infers that the scrolled-to utterance model is old, because it was scrolled to, and issues a command to print a fresh copy (line 6).

There is an alternative explanation of the programmer's behavior. She may intend to print a current copy

1. **comprehend u-model**
2. **imagine referent on u-model**
 retrieves episodic memory
3. **scroll to u-model**
4. **comprehend u-model**
5. **fixate u-model identifier**
6. **print u-model fresh**

Figure 34: Model as it stands, scrolling to u20

of the utterance model as soon as she begins thinking about the utterance model. This is illustrated in Figure 35, on lines 1 and 2. In this scenario, scrolling to the hidden utterance model would have the more direct purpose of retrieving the identifier needed for printing a current copy of the utterance model. This scenario introduces nested command goals, with the scrolling as a subgoal of printing. Currently the model represents every command as complete and atomic.

One proposal for representing command subgoals would be as analogs of existing subgoals, but selected in service of command goals. For example, the model might select a goal to print the utterance model, but find that this goal cannot be achieved, because the identifier is missing from WM (line 2, Figure 35). This would cause the architecture to generate a new context in which to select subgoals to try to achieve the goal. One of these subgoals might be to probe for possible ways to retrieve an identifier (line 3). This might retrieve several methods, including scrolling to an old copy of the to-be-printed object, if such a copy exists (the next section describes alternative methods). The model might then try to recall whether it had seen an old copy of the utterance model that it could now revisit. To try to recall this, the model might imagine attributes of the utterance model, to trigger any episodic memories for having fixated them. The referent candidate is again a likely candidate (line 4), as the attribute being constructed by the program at this point. Having recalled looking for a referent, the model might then scroll to the utterance model where it looked for the referent (line 5), fixate on the identifier (line 6), and finally issue the completed print command (line 7), all in service of the persistent command goal to print a current copy of the utterance model.

1. **comprehend u-model**
2. **print u-model**
3. **probe for methods for retrieving identifier**
 recall: scroll to an old copy
4. **imagine referent on u-model**
 retrieves episodic memory
5. **scroll to u-model**
6. **fixate u-model identifier**
7. **issue print command**

Figure 35: Model as speculated, scrolling to u20

The model makes some very specific claims about the programmer's behavior. These compete with other plausible accounts, such as the alternative account of scrolling event 3 presented above, and an earlier

account of scrolling event 5 (Altmann et al., 1995). This competition, and the simplifications of the world model, seem to leave the essential hypotheses of the model intact. There continues to be a role for low-overhead encoding and knowledge-based retrieval of episodic memory.

6.7.1. Alternative methods for retrieving an identifier

Above we described one method for retrieving an identifier, namely scrolling to an old copy. This is just one of the methods that were available to the programmer, and that would also have to be represented in a more detailed analysis of her command choices.

A second identifier-retrieval method might be to try to recall the identifier directly. The programmer may attempt this, as suggested when she refers to "u'-something", but if so then this method fails, as suggested when she scrolls instead. This failure is consistent with the model's inability to imagine identifier symbols. These symbols are generated by the program at runtime to link data structures. Their purpose is to be unique, rather than mnemonic, and they may change between runs if the programmer modifies the data-structure configuration. For example, the identifier for this same utterance model, now u20, could become u21. The model has no meta-generator for images of identifiers; u20 would have to be well-learned, as it is by now for us as analysts, before it could be represented in the model as an imaginable feature. This again points again to the importance of asking, in the context of a minds-eye-based theory of retrieval, how people acquire the ability to imagine a particular feature (Section 6.5.2).

A third identifier-retrieval method would be to print data structures, beginning with the s15 data structure that is visible on the display when the programmer begins thinking about the utterance model (Figure 23, p. 59). The utterance model is several levels deep. Its identifier could be accessed in three print commands, starting with a command to print s15. Each print command would have to be followed by a visual search for the next appropriate identifier to print out. Thus the cost of this method might outweigh the cost of scrolling to a hidden object, if the hidden object is known to exist and scrolling is very likely to succeed.

A fourth identifier-retrieval method would be to use the editor's search command to find a hidden copy of the utterance model. This could be applied without knowing of the existence of a hidden copy, but would then require knowing an effective search criterion. The string 'u', which the programmer apparently recalls or infers about the utterance model identifier, is probably not distinct enough to be effective. Another possibility would be to try to recall having seen other features of the utterance model as strings to search on. However, the symbol *referent* (for example) appears elsewhere on the display, in SPs and chunks that the programmer printed out, so it may also fail to make searching cost-effective.

The selection between these methods could be done by selection rules operating in service of the print command, in the context beneath the corresponding command goal (after line 2, Figure 35). Selection might require the retrieval of *survey knowledge* (Golledge, 1991, Mantei, 1982, Thorndyke, 1981), which is map-like knowledge that enables spatial inferences. For example, a selection rule might take into account the distance to a hidden object, when computing the cost of getting to that object to compare with other identifier-retrieval methods.

Such selection rules could help address the question of why the programmer doesn't scroll more often, if episodic knowledge is as pervasive as we claim it is (Section 6.1.2). It may be that the programmer is often prepared to scroll, having gone through the first three links in the chain of knowledge that leads to scrolling (Section 6.1.3), but that other methods for retrieving the same information are less costly and are selected instead.

6.7.2. Discussion: implications of more detailed motor behavior

Above we offered a speculative extension of the model that incorporates a more detailed account of motor behavior. The model as it stands seems as if it would accommodate GOMS-like methods and selection rules, if it decomposed commands into separately-achievable subgoals. An actual implementation would allow an analysis of learning over command goals, a behavior much closer in kind to the other HCI models mentioned earlier. Previous analyses have identified constraints on how commands must be represented in Soar in order to preserve GOMS-like behavior as Soar learns (John, 1996). Comparing detailed motor behavior in our model, which is based on data reflecting complex problem solving, to similar behavior in other models might point toward a common set of mechanisms that would support integrated models of problem solving and skilled interface use.

The speculated extension suggests that the model's simplifications do not affect its hypotheses about episodic memory. In its behavior there is still a plausible role for effortful retrieval of episodic knowledge. Unable to recall the identifier, the speculated model tries to recall other features of the utterance model that it might have seen. The referent is a plausible choice to imagine, given the programmers surrounding references to a referent, but other attributes would have done as well. More importantly, there appear to be roles for imagining in methods other than scrolling, for example to generate strings for use with search commands.

Two other points come out of this section, related to alternative accounts of the encoding process. First, it might be that there is an abstract symbol in the programmer's mind that symbolizes the entire utterance model. There is no lexical symbol on display that identifies the utterance model as such directly, but this leaves open the possibility that the programmer retrieves this symbol when she recognizes for what it is. Thus fixation, and the encoding of fixation events, may involve more complex perceptual processing than the model represents. The second point concerns the possible storage of richer knowledge at encoding time. As discussed above, a selection rule for choosing the quickest way to retrieve an identifier could make use of a measure of the distance of a hidden object. A faraway object, for example, might bias the choice against scrolling. To glean this information at retrieval time would impose demands on what kind of information would be stored at encoding time, and what kind of processing it affords. A model could link a fixation event to a map-like spatial knowledge, or could store richer episodic information that would allow estimates about amount of output generated in the intervening interval. Some external constraint would have to be found to ensure that a model with a more detailed world model carries out plausible cost estimates for accessing hidden information.

Our purpose in this section was to see whether the model would admit a more realistic world submodel, and

whether this would affect the model's hypotheses about episodic knowledge. We proposed how the model might make use of command subgoals to provide an alternative account of one of our scrolling events, with minor changes to the representation of command goals. This identified a role for the same kind of episodic knowledge the model uses now. The role in this example would be to help select methods for achieving command subgoals — searching for a hidden object is a better prospect if we can establish that such an object probably exists.

6.8. Hypotheses concerning screen clutter

We would like to be able to use complex models to make predictions useful in designing artifacts. Though our focus was on modeling a particular memory phenomenon, our model is rich in detail that may have many implications for interface design. In this section we examine what the model might say about *screen clutter*, defined here as an excess of possible directions to choose from when intending to navigate to a setting of interest.

As we discussed in Section 6.7, our model's spatial knowledge is limited to distinguishing what is hidden from what is visible. Supposing its world model were extended to represent multiple locations for features, such as different buffers or windows, how would it know where to look?

The model could infer direction from survey knowledge that allowed for inferences from feature to location. For example, programming environments often deposit different kinds of trace output into different windows. Good survey knowledge of a well-structured programming environment would make it easy to infer the containing window, or the direction in which to scroll, from the to-be-located feature. Acquiring such knowledge would take time, introducing a penalty for novice users. Learning time might increase with the complexity of the feature-location mapping.

A novice using a cluttered interface might retreat to a strategy of encoding and retrieving location information, as the model now encodes and retrieves episodic information. To retrieve such location information, the model would have to imagine potential locations, asking itself where it might have seen a feature of interest. This process would be deliberate, and would come at a cost in terms of selecting imagine subgoals. This in turn would interact with the model's goal selection mechanism. The longer the model spends retrieving information for a goal, the more likely it is to select a new goal. Thus generating candidate locations, already time-consuming, could also distract the model from its goals.

To summarize this analysis, the model makes three hypotheses about clutter. First, when navigating to hidden features, there can be a penalty for having many locations to choose among. Second, the penalty lies in the cost of generating images of locations, and in a greater chance of losing track of the goal while generating them. Third, survey knowledge mapping features directly to locations is the key to reducing this penalty.

6.9. Summary

Navigation in the model comes about through a chain of events involving knowledge. Knowledge says when to look for certain features. When the model fixates on something, it encodes the event. Later, knowledge says when to imagine a feature, and what form the feature takes. Imagining retrieves any memories the model might have of seeing that feature. Finally, knowledge says to visit the setting of a feature that the model has remembered the existence of.

The model learns many rules other than episodic ones. Feature memories are display-based rules that let the model incrementally comprehend an object, and recoded facts make frequently-accessed knowledge more accessible. On the other hand, episodic memories are only the simplest of the kinds of episodic knowledge that problem-solvers exhibit.

Some constraint on WM is a necessary part of accounting for LTM scrolling. A maximum 2-goal persistence meets the functional needs of forcing retrieval of episodic information from LTM where the data suggest this is necessary, while letting the model account for short-term scrolling events by keeping hidden-feature information in WM.

The model supports a purposeful yet flexible mode of thinking. It delays selection of a new comprehension goal by waiting for a hint that supports a new goal. After a rough conversion to programmer time, the resulting goal duration is consistent the duration of mental moves observed in other tasks. As information about a goal accumulates in WM, so do potential hints for which goal to choose next. This places a functional bound on the duration of a goal, and allows for a seemingly far-flung choice for the next goal.

The model is limited with respect to programming in general and comprehension in particular, though the two limitations are related. Modeling a broader range of programming behavior would require comprehension to produce a richer functional representation. Also, the model is not able to learn to imagine features. The architecture provides a framework in which to pursue the question of how to model this.

Like our model, the IDXL model of Rieman et al. (1996) also emulates the comprehension of external information, some of which is hidden. We draw support from IDXL for our model's hypothesis that people cycle back to external information, accumulating knowledge in increments stored in LTM. IDXL encodes more semantic information in its recognition chunks than our model does in episodic memories. This may reflect a difference in depth of processing per feature in the two task environments.

Our model seems to admit extensions to a more detailed world model, and to correspondingly detailed motor behavior. Such extensions might change the specific role of episodic memory in mediating commands, but do not seem to undermine the model's hypotheses about encoding and use of episodic memory.

Given a world model representing different locations where a feature might be, the model might have to ask itself where it saw something, in addition to asking itself whether it saw that thing. The cost of this additional imagining process could be reduced if the environment were structured to allow inferences from feature to location.

Chapter 7

Episodic long-term working memory

We claim that access to external information is a function both of episodic memories for having seen features, and of knowledge used to retrieve those memories. These claims appear to be related to the theory of *long-term working memory* (LT-WM) proposed by Ericsson and Kintsch (1995). According to their proposal, experts in a particular domain can expand their WM in that domain through the use of knowledge stored in LTM. This long-term knowledge provides “hooks” for storing and retrieving working information that arises during a task. This is a parsimonious account of expanded WM, in that the independent variable in determining WM capacity is acquired knowledge, rather than inherent differences in STM proposed ad hoc.

This chapter compares our claims to Ericsson and Kintsch’s proposal. The purpose of this comparison is two-fold. First, we want to propose *episodic long-term working memory* as a variant of Ericsson and Kintsch’s proposal (which we refer to simply as *LT-WM*). To do this we need to examine the similarities and the differences. Second, our model represents a model of episodic LT-WM in use, and we want to examine this model for what it says about the associated phenomena. A mechanistic model is a rigorous response to the call to action with which Ericsson and Kintsch conclude their paper:

It is clear that our analyses of skilled performance must probe deeply into the organization of knowledge and its encoding and retrieval processes if they are to fully describe the operation of LT-WM. Only if we are willing to dissect complex cognitive skills and fully describe them will we ever ascertain the real limits of cognition and create a theoretical framework for working memory that encompasses the full range and complexity of cognitive processes (Ericsson and Kintsch, p. 240).

Our comparison is organized around the components of LT-WM as proposed by Ericsson and Kintsch: (1) the ability to encode memories in LTM on-line (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. Section 7.1 describes whether and how these components manifest themselves in our data. Section 7.2 maps these components to characteristics of the model. Section 7.3 discusses how external constraints shaped the two proposals differently, and Section 7.4 summarizes the similarities and differences.

7.1. Evidence for episodic LT-WM in our data

Evidence for rapid and reliable encoding comes from LTM-scrolling events in our protocol (Section 7.1.1). Evidence for rapid and flexible retrieval comes from semantic coherence between cues on the display and scrolled-to information (Section 7.1.2). Expert knowledge about the program and the language seems to act as a retrieval structure that connects visible information to episodic memories for related hidden information. Evidence for interference-resistant encodings is lacking in our data, but this may be a function of the task environment (Section 7.1.3).

7.1.1. Rapid and reliable encoding

By "rapid and reliable" encoding, Ericsson and Kintsch appear to mean encoding that occurs in the course of regular cognitive activity. They refer to incidental memory as "the most direct method of assessing experts' storage of information in LTM during regular cognitive activities" (p. 214). Incidental memory is the most specific measure provided for determining when encodings in LTM occur rapidly enough to support LT-WM.

In tests of incidental memory, the experimenter unexpectedly asks the subject to recall information related to task performance, once that is complete. If the subject can recall information after it has fallen out of STM, without having been told ahead of time that such recall would be tested, then the information must have been stored in LTM as part of his or her regular cognitive activities.

What we have termed LTM-scrolling events (Section 2.1.2) seem to reflect the same memory phenomena as incidental memory. First, there is enough time and cognitive activity between a feature becoming hidden and scrolling to that feature to rule out its maintenance in STM. Second, we posed no retrieval demands ahead of time. When the programmer scrolls based on a memory for a hidden feature, the retrieved information must have been stored in LTM as part of her regular cognitive activities.

We found no specific evidence that the programmer anticipates future retrieval demands on her own. She might have said "I need to remember this" (or the equivalent) about some feature, but did not. To the extent that the programmer does anticipate her retrieval demands, the processes involved are never vocalized.

7.1.2. Rapid and flexible retrieval

Skilled-memory theory, on which Ericsson and Kintsch base their theory of LT-WM, proposes that improvements in recall are due to *retrieval structures* (p. 216). These constitute specialized structures that experts possess and novices do not, located in LTM (Richman et al., 1995). The purpose of these structures is to maintain close and reliable associations in LTM between easily-generated cues and to-be-retrieved elements. To retrieve a particular element at a specific time, the expert follows links in the retrieval structure from the cue to the to-be-retrieved element.

As an example of the use of retrieval structures, chess experts can encode a meaningful board in such a way as to make the contents of individual squares readily accessible. Cued with individual squares, a chess master could recall the contents at a rate of roughly 1 per second (Ericsson and Staszewski, 1989). Ericsson and Kintsch review studies that suggest that time to access LTM is around 300ms (p. 215). By this estimate, the chess master's retrieval structure required roughly 3 consecutive retrievals to connect a given square presented as cue to the piece it contains (if any).

Retrieval structures are adaptive to the memory demands of the task. In our task, the programmer needs to remember the existence of hidden information, when that information could inform what she is trying to comprehend. Our protocol data lack quantitative evidence on time to get from cue to result. However, our data do provide some insight into how recall of episodic knowledge supports the programmer's comprehension activity.

In our data, expert knowledge appears to act as the retrieval structure. Knowledge makes the connection between some current goal and a relevant hidden feature. For example, in scrolling event 1, the programmer scrolls to hidden information that has a causal connection to the chunk she is comprehending. While comprehending chunk-128, she notices a missing problem-space condition, which leads to thinking about the superstate, and then to noticing a shared state, and finally to scrolling (Section 2.3.1). Scrolling reveals the SP that fired to cause the chunk (the causal relationship is described in Section 4.3.2). The relationship between chunks and the SPs that cause them leads from chunk-128 to information about the scrolled-to feature.

This expert-knowledge retrieval structure appears to enable rapid access. In scrolling event 3, the programmer scrolls to retrieve an identifier from an old, hidden copy of an object, so she can print a new copy (Section 2.3.3). Another method that did not involve hidden features was available for retrieving the identifier; the programmer's decision to scroll instead suggests that she had ready access to her memory for the hidden feature.

7.1.3. Interference-resistant encoding?

Ericsson and Kintsch propose that a key component of LT-WM is interference-resistant encoding of working information. However, interference-prevention mechanisms are adaptive to the needs of the task. For example, in mental-abacus calculation (p. 233), the expert uses LT-WM to maintain access to certain intermediate results. A succession of intermediate results arise in the course of a given problem, but only the most recent is necessary. Experts were in fact found to have poor incidental memory for all intermediate results but the most recent.

In the scrolling events we studied in detail, the scrolled-to feature is the only instance of that feature. We had no opportunity to assess interference between memories for multiple instances of a feature.

However, the programmer's task environment may not demand interference-resistant encodings even when there are multiple instances. Trace information generated by the program becomes invalid or irrelevant

with time. A code fragment or data structure either changes, in which case the newest hidden copy supersedes older ones, or it does not, in which case the newest copy is sufficient.

When people search for strings (rather than scroll to them), they may recall the string's existence as a hidden feature, but may not distinguish different instances from memory. Searchers often evaluate the context of a hit, continuing to search if the context fails some criterion. This kind of interaction was observed in a study of help-text browsing (Peck and John, 1992). It is also supported by short-cut methods for finding the next instance, which appear in interfaces ranging from Emacs to the Macintosh. In searching tasks in general, it may be sufficient to recall that a feature is out there somewhere. Preventing interference between memories for different instances may be unnecessary, because the continued external existence of different instances means that the distinction will not be lost if they are confused internally.

Thus in keeping with the adaptive nature of LT-WM, it may be that interference-resistant encodings in LT-WM are unnecessary in some domains. Our programmer appears to make use of LTM to store and gain access to working information, and it seems appropriate to consider this activity as an instance of LT-WM.

7.2. Mechanisms for episodic LT-WM in our model

This section relates the episodic-memory processes in our model to Ericsson and Kintsch's components of LT-WM. The model's underlying architecture encodes memories without deliberation, for all features fixated on by the model (Section 7.2.1). Retrieval structures for episodic memories consist of sequences of goals and subgoals that end in a retrieved episodic memory (Section 7.2.2). One such memory is sufficient to indicate that a hidden feature exists (Section 7.2.3).

7.2.1. Rapid and reliable encoding

The LTM-scrolling events in the protocol imply that the programmer stores information about features in LTM as part of her regular cognitive activities. To account for this, the model encodes episodic information about all features that the model fixates on.

Encoding is rapid in that it involves no deliberation. The model has no knowledge of a newly-encoded memory until it retrieves the memory later. This is a constraint imposed by the architecture (Section 3.3). Also, the model itself invests no additional effort in encoding more general memories through inductive problem-solving. Thus the model's only episodic memories associate a timestamp with a specific feature. This specificity places the burden on the retrieval process, which must be able to generate features in WM as cues.

Encoding is reliable in that it captures every feature that the model fixates on. This is also a constraint imposed by the architecture. Because learning is a side effect of information entering WM, there is no opportunity for the model to select what features to encode, separately from what features to fixate on.

The consequence of these architectural constraints on the encoding process is that the model can retrieve

episodic information about any feature it can imagine. Generating imagined features is both necessary and sufficient to make use of episodic LT-WM.

7.2.2. Rapid and flexible retrieval

Our data show evidence for retrieval structures that embody expert knowledge. The programmer has ready access to information about hidden features relevant to what she is thinking about. Semantic knowledge seems to relate her pre-scrolling cognition to the scrolled-to features.

In the model, a retrieval structure for episodic information is a sequence of steps leading from a comprehension goal to an imagine subgoal, and thereby to an episodic memory for the corresponding feature. For example, in scrolling event 1 (Figure 21, page 45) the following goals and subgoals lead from comprehending chunk-128 to retrieving a memory for the hidden create-referent assertion.

comprehend chunk-128	chunk-128 is on display
comprehend right-hand sides	chunks have right-hand sides
comprehend operator condition	conditions affect actions (pieces of right-hand sides)
fixate no problem space	notice missing context condition
imagine superstate target	guess context that if modified causes a chunk
comprehend superstate	think about this context
imagine create-referent asserted	guess asserted SPs that could modify context
retrieves episodic memory for create-referent assertion	

Retrieval can be rapid, depending on knowledge. For example, in scrolling event 3 (Figure 23, page 59), the model has program-specific knowledge that links a particular feature (the referent attribute) to a particular goal object (the u-model). Whether the referent attribute has been created yet is a measure of the progress of the program. Having selected the u-model goal, the model moves directly to imagining the referent attribute.

comprehend u-model	think about an object constructed by the program
imagine referent on u-model	imagine an attribute of that object
retrieves episodic memory for having looked for the referent attribute	

Retrieval is flexible, in that the path from initial goal to retrieved memory is interruptible and contingent. It can be interrupted by other goals and subgoals. It is contingent in that with different knowledge, or different subgoal selections (Section 3.6.2), the model may arrive at another outcome, perhaps not involving retrieval of an episodic memory. This flexibility is appropriate for the task. There are potentially many features that could be relevant to a given goal object, and the model should not be bound to follow a fixed path from a given goal object to a particular feature.

7.2.3. No interference-resistant encoding

Our data contain only one instance of each scrolled-to feature, but this may not matter. Even with multiple instances, interference between memories for them might not be a problem. As discussed in Section 7.2.1, the task environment is such that the most recent instance generally supercedes others. It might be sufficient to recall that at least one instance exists as hidden information.

When the model imagines a feature it has seen before, it retrieves memories for every occasion it fixated on that feature. It does not discriminate between these occasions. Instead, if there is at least one memory, the model infers that it has seen the feature before, and that because the feature was imagined now, the external instance must be hidden. Absent constraining data, we did not include mechanisms in the model for handling interference.

7.2.4. Broader evidence for episodic LT-WM in the model's behavior

The analysis in this chapter has been limited to LT-WM as a factor in episodic memory for features, because of the strong evidence in the data. However, the data on the model itself implicate LT-WM more broadly. The firing counts of encoded rules, presented in Section 5.2 (p. 74), suggest LT-WM is a factor in mediating storage of many kinds of information.

The model encodes and uses many rules other than episodic memories for features. The model encodes 407 feature memories, of which 152 fire a total of 354 times. Scrolling event 2 contains an example of a feature memory being retrieved (Section 4.3.2). The model also recodes its expert facts into 257 new memories, of which 44 fire a total of 62 times. Below we give an example of a recoded fact in use, and discuss the implications for LT-WM.

7.2.4.1. Re-encoding and retrieval of a fact

Figure 36 shows a fact being recoded and the new memory being retrieved.¹⁸ At the top of the figure, the model issues a command to print out the run-time stack, and then selects a goal to comprehend the current execution context. Probe-proposal knowledge knows that operators are useful to probe with, so the model probes with an operator it already has in WM (the exhausted operator). The probe recalls two facts related to this operator. The first fact (1a) is the high-level goal of returning a new pointer to an existing data structure. The second fact (2a) is a connection between the high-level goal and the exhausted operator. This connection is important because the exhausted operator is one of the last to occur in the current execution context, indicating that control is about to transfer to an older context. This is the time to think about returning a new pointer to the older context. Thus the probe both reminds the model of its high-level goal, and makes the model aware of the important cue that led to this reminding.

The result of retrieving these facts is to encode new rules (1b and 2b) that associate the facts with the current goal. The new rules include another condition as well, a result of the architectural learning mechanism tracing through all WM elements that play a role in retrieving information (Section 3.3). The additional condition requires that WM contain the exhausted operator, which is the information that led to the probe that retrieved the facts.

¹⁸Pointers to details of the behavior in Figure 36 are as follows. The proposal for `print_stack` is `po*display*print-stack`. The proposal for the goals to comprehend the current execution context is `po*comprehend*current-context-when-exhausted`. Facts 1a and 2a correspond to the rule `f:high-level-goal-cues`. In the detailed model trace, the encoding episode occurs between decision cycles 86 and 90, and the recall episodic occurs at decision cycle 377.

7.3. Discussion: cognitive architecture as constraining theory

Skilled-memory theory (Ericsson and Staszewski, 1989, Chase and Ericsson, 1982) plays an important role in Ericsson and Kintsch's proposal for LT-WM. It provides a pre-existing set of mechanisms on which to build, adding necessary constraint and increasing the coverage of the final theory.

Skilled-memory theory posits elaborate knowledge used at encoding time to associate to-be-retrieved elements with easily-generated cues. For example, a subject DD achieved exceptional digit span (106 digits) by mapping sets of digits to mnemonic codes (long-distance running times, dates, ages, etc.) of which he could recall a large number (Ericsson and Staszewski, 1989). He grouped these codes in turn into supergroups, and those into clusters of supergroups. According to skilled-memory theory, there are two principles underlying this kind of ability. The *meaningful-encoding principle* states that the subject uses prior knowledge (e.g., mnemonic codes) to represent information meaningfully during encoding. The *retrieval-structure principle* states that experts associate these meaningful elements in LTM with easily-generated cues (e.g., the indices "first", "middle", and "last" for elements of a supergroup).

In our model, the constraining theory is the underlying cognitive architecture. Soar encodes simple, recognitional memories ubiquitously (Howes and Young, 1996a, Newell, 1990). Retrieving information stored in such memories requires a two-process search of LTM, which first generates candidate elements that are then recognized by encoded memories. This generate-and-recognize approach is consistent with other two-process models of free recall (e.g., Anderson and Bower, 1972, and Kintsch, 1970). Soar models can apply this generate-and-recognize process to encode more general memories using the default recognitional ones (Huffman, 1994, Miller, 1993, Vera et al., 1993, Bauer and John, 1995). However, this kind of elaborative encoding requires deliberate cognitive activity. In the absence of evidence of such activity in the data, the architectural constraint speaks the loudest.

Our protocol data, the task environment, and the model's contingent retrieval processes are all consistent with the default architectural constraint on learning. The protocol data show no evidence of elaborative processes geared toward encoding more general memories. The task environment is such that the model cannot know ahead of time what information will be useful to retrieve later. In contrast with skilled-memory tasks, there are no explicit retrieval demands that can be anticipated. Yet the programmer seems to have ready access to episodic memory about seemingly arbitrary features. For the model to achieve the same kind of access, it must encode large amounts of information non-selectively in LTM. Finally, the model's retrieval structures consist of highly-contingent paths between comprehension goals and relevant features. There may be many such paths, and anticipating specific ones at encoding time to make retrieval more direct may not be worth the effort.

The architecture and the task environment specify a different role for knowledge than skilled-memory theory, shifting the role of expertise from encoding time to retrieval time. We can illustrate this difference in terms of the meaningful-encoding principle, which states that the subject uses prior knowledge to represent information meaningfully during encoding. In the model, the units of meaning encoded in LTM are symbols that represent only a rudimentary sense of time. The model's encoding process contains no

analog of imposing specialized domain knowledge on a feature, as digit-span subjects imposed running times on digit-strings. Elementary timestamps are sufficient to account for the navigation in our data. The model's retrieval process has the opportunity to gain access to all encoded information, but doing so takes deliberate effort and knowledge for generating cues.

The model and the task environment suggest that the retrieval structures for LT-WM are more flexible and perhaps less controllable than those of skilled-memory theory. The retrieval-structure principle of skilled-memory theory is abstracted from tasks with specific, predictable retrieval demands. Consequently, it states that experts associate to-be-retrieved elements with cues that can be generated in a controlled manner from an elaborate LTM structure geared toward those demands. In contrast, the contingent, interruptible retrieval structures in our model reflect the dense interconnections between objects and relevant features in our real-world task domain.

Our model also differs from skilled-memory theory with respect to how retrieval structures are used at encoding time. The skilled-memory subject generates cues in STM at encoding time in order to associate them with to-be-retrieved elements. The retrieval structure presumably plays a role in generating these cues systematically. In our model, only one element of the retrieval structure plays a role during both encoding and retrieval, namely the feature itself. The model must both know a feature when it sees it, and know how and when to imagine it.²⁰

7.4. Summary: episodic LT-WM compared to LT-WM

Episodic LT-WM seems both related to and distinct from the theories of skilled memory and LT-WM. The main similarities are, first, that both use LTM to store working information that arises in the course of a task. Both thus meet the basic criterion for a working memory, that it provide "rapid and reliable access of a particular piece of information at a specific time" (Ericsson and Kintsch, 1995, p. 215). The second main similarity is that retrieving this working information requires long-term knowledge of the kind that experts acquire.

The main differences seem to be, first, that retrieval structures in our phenomena are less explicit than the ones identified for skilled-memory subjects; and, second, that little work is done at encoding time to link to-be-retrieved elements into retrieval structures, or to encode them in ways that resist interference. These differences are consistent with the task environment, in which the retrieval demands are not known ahead of time. There are many features, some small number of which may be relevant in the future, depending on how the task evolves and what knowledge is evoked. Under these circumstances, it seems appropriate to invest minimal per-feature effort at encoding time, and to rely on retrieval based on relevance to task behavior.

Our model predicts that episodic LT-WM is ubiquitous. The architecture learns passively and pervasively.

²⁰This and the other kinds of knowledge that eventually lead to the overt behavior of scrolling are discussed in Section 6.1.

The model adds only a rudimentary sense of time applied to the noticing features, and a mind's eye for imagining features. Neither this sense of time nor the mind's eye are particular to programming. In any complex environment with many more relevant features than we can see at one time, access to external information is likely to be mediated by episodic LT-WM.

Chapter 8

Contributions and future work

Our model provides a detailed account of how a programmer might have navigated to hidden external information. This account conforms to strong constraints imposed by the data and the underlying cognitive architecture. This chapter examines the contributions of this work and directions for future work. Section 8.1 reviews how constraints shape the model, and presents our hypotheses about the use of episodic long-term working memory. Section 8.2 discusses the place of our model in programming psychology. Section 8.3 summarizes the contributions.

8.1. Cognitive science: Episodic long-term working memory

A line of research in psychology has examined how people store dynamic information in LTM. Skilled-memory theory (Chase and Ericsson, 1982, Ericsson and Staszewski, 1989) describes specific examples of how people acquire sophisticated mnemonic structures for remembering meaningless information. Digit-span subjects, for example, were able to remember long sequences of random digits by finding meaningful patterns in them. These digit sequences, like the features in our data, arise dynamically, but subjects encoded them deliberately, using knowledge tailored for the task of memorization.

Long-term working memory (Ericsson and Kintsch, 1995) broadens the account of skilled-memory theory to domains where the memory task is not explicit. The proposal is that expertise includes the acquired mnemonic structures that allow the encoding and retrieving of dynamic task-related information. Experts understand a domain well enough that such information is meaningful to them, making it memorable.

We propose a further broadening, namely that people make use of an *episodic long-term working memory*. The stored dynamic information concerns the event of seeing a feature, and is associated in LTM with no more sophisticated a structure than the feature itself. The encoding of this information is passive, a side effect of normal interaction with the environment rather than the result of deliberate memorization. However, the retrieval of this episodic information is still deliberate, and still depends on the use of domain knowledge. People decide to search for memories about a feature, and carry out this search by imagining the feature in their mind's eye. This requires knowing when it would be useful to remember having seen the feature, in addition to the ability to call to mind an image accurate enough to trigger the appropriate memories.

8.1.1. Directions for future work

Our proposal for passively encoded episodic information derives directly from the constraints of Soar's learning mechanism, as used to model our data. By default, Soar learns rules that only fire when elements of the encoding context are in WM as cues (Section 3.3.1). Other proposals for how people encode dynamic information emphasize more elaborate encoding processes, which associate to-be-remembered information with well-known or otherwise easily-generated cues (Chapter 7). In the same vein, memory improves with depth of processing at encoding time (Anderson, 1994). What can we extrapolate from our model and from Soar about the memory improvements that come with applying knowledge at encoding time?

Soar's answer to the question of how to learn generalized rules is *data chunking* (Howes and Young, 1996b, Rosenbloom et al., 1991, Newell, 1990, Rosenbloom et al., 1987).²¹ By this process, a model implemented in Soar can learn rules that associate a to-be-retrieved element (the "data") with cues other than the element itself. This requires deliberate problem solving on the part of the model. Having first learned to recognize the to-be-retrieved element, the model must reconstruct the element in WM from prior knowledge in LTM, recognizing when this reconstruction process has succeeded. At this point the model can learn a new rule associating the reconstructed element with cues present at reconstruction time. Data chunking is thus a process of deliberate, knowledge-based encoding.²²

We can speculate about the digit-span task to see how Soar might model skilled memory with data chunking. Highly-skilled digit-span subjects found patterns in presented digit sequences, and then linked patterns into higher-level mnemonic structures (Chase and Ericsson, 1982, Ericsson and Staszewski, 1989). To encode a short span of digits, a Soar model might reconstruct the span from its knowledge of patterns, and then learn a rule associating the reconstructed span with a cue that is part of an higher-level organizing structure. The ability of a Soar model to memorize a contiguous sequence of such spans rapidly and reliably would depend directly on the amount of pattern knowledge it had available, and on its ability to retrieve its organizing structures. Thus the constraints of Soar's learning mechanism appear to predict key aspects of skilled-memory phenomena.

Following through with an implemented Soar model of skilled-memory data would contribute to research in unified theories of cognition (Newell, 1990), by bringing more important phenomena under the roof of one such theory. Also, as a powerful descriptive tool such a model would contribute to our understanding of the phenomena themselves:

Only if we are willing to dissect complex cognitive skills and fully describe them will we ever ascertain the real limits of cognition and create a theoretical framework for working memory that encompasses the full range and complexity of cognitive processes (Ericsson and Kintsch, 1995, p. 240).

²¹Section B.5 steps through a simple example.

²²The imagine mechanism is similar to data-chunking, in that it reconstructs a WM context for the purpose of activating rules that recognize the imagined feature. It differs in that it is not oriented toward learning new, more general rules.

8.2. Programming psychology: A (limited) model of real work

Previous computational cognitive models of programming have, like ours, focussed on specific subtasks of programming. Modeled subtasks include design and planning (Rist, 1995, Steier, 1989), coding (Rist, 1995, Brooks, 1975), comprehension (Green et al., 1987), and learning and transfer (Wu, 1992, Spohrer and Soloway, 1989, Katz, 1988). Of these the most comprehensive is Rist's *Zippy* model, which separates cognitive mechanisms for planning and coding from the knowledge required to carry out a particular task. For the subtask of program comprehension, which dominated the programming behavior we studied, there are also many non-computational models (Von Mayrhauser and Vans, 1995, Davies, 1994, Gellenbeck and Cook, 1991, Wiedenbeck, 1991, Detienne and Soloway, 1990, Gray and Anderson, 1987, Pennington, 1987a, Letovsky, 1986, Brooks, 1983). In all these models, the behavior on which the models are based involves experimenter-specified programs. These are usually simple and small, compared to the kind that an experienced programmer typically works on. Also, in most cases the subjects were novices.

Our model is of an experienced programmer working on her own program in her accustomed programming environment. This led to discoveries that might have been difficult to achieve in a laboratory setting. For example, the volume of external information generated by the programmer's program was great enough to force the question of how the programmer gained access to it. This volume was partly a function of the complexity of the program, which required the programmer to print out considerable state information. It was also a function of the programming environment, which provided a language interpreter in which to run the program and print the necessary state information. The programmer's ability to navigate through this large volume of information, as we have proposed, is a function of her knowledge about the domain. Thus the opportunity to study access to external information arose from the factors that make up authentic work.

A second example of how authentic work influenced the model is the goal-selection mechanism. The programmer thought about a lot of different objects during the 10.5-minute interval we studied. This raised the question of how to model the selection of the next object to think about. The architecture is largely silent on this question.²³ The pattern we observed in the protocol was that newly-generated information led to the next goal. This seems to follow from the use of a language interpreter to comprehend a program interactively — running the program a step generates new output that activates knowledge and goals that in turn lead to stepping the program again. The converging-evidence criterion, in which some visual or other cue prompts the model to select from a set of proposed goals (Section 3.6.1), emerged as our solution to having new information mediate the direction of the train of the thought.

Another way in which our study differs from previous work is in the programmer's language, which was a production system as opposed to the procedural languages used in other studies. The two paradigms differ broadly, for example in how they represent control flow. In production systems, control flow has little explicit representation and is often a challenge to determine even in a running system (like our model itself; Section 4.1). In addition, our programmer's program generated and executed new code at run-time, making

²³See Section B.3.1, p. 118 for a discussion of this in Soar terms.

its behavior that much harder to track. In contrast, procedural languages have explicit control flow constructs and may be much more amenable to plan-structured internal representations (Rist, 1995, Pennington, 1987b, Soloway and Ehrlich, 1984). To understand the large-scale effects of different programming paradigms on internal representations, we need to understand those internal representations in the kind of rigorous detail that computational models can supply.

Our primary contribution to the psychology of programming is an additional point in the space of computational cognitive models, one which branches out from previous approaches in taking as its starting point the behavior of authentic work in all its complexity. The model accounts for this behavior in limited fashion (Section 6.5.1), but enough of the original task environment shows through to lead to discoveries characteristic of real work.

8.2.1. Directions for future work

Our data leave us in a good position to broaden the model's coverage of the tasks of programming, given that the programmer switches to coding soon after the interval covered by the model. Extending the model to account for coding activity would impose multiple new constraints. For example, comprehension would have to produce knowledge structures that represent functionally-increased understanding. Such structures presumably implicate a more sophisticated use of LTM, as the programmer's understanding appears to grow over a period of tens of minutes, and might take us in the direction of existing models of text comprehension (Kintsch, 1988, Lewis, 1993). Modeling the code-generation process would allow us to interpret the model's memory structures in terms of classical planning-coding conceptions of programming. This would be an important unification between our analysis of authentic work and existing laboratory studies.

8.3. Concluding summary

Our model offers an explanation of how people encode and retrieve simple episodic memories for features of the environment. This explanation is constrained both by human data and by the learning mechanism of the underlying cognitive architecture. Our first hypothesis is that people passively encode memories about what they see while performing a task. Each such memory stores the event of seeing a feature. To be retrieved, this episodic memory requires that the feature itself be represented in WM as a cue. This implies our second hypothesis, which is that retrieval is a function of domain knowledge. The syntactic component of this knowledge must be capable of generating the feature in the mind's eye, as a cue. The semantic component of this knowledge must know when the cue is relevant — that is, when it might be useful to recall having seen that feature.

Studying real work helped to reveal these episodic-memory phenomena. The need to manage access to large, hidden information spaces was a function of both the amount of information needed to describe the behavior of a complex program, and the programmer's deep knowledge of how elements of this external information were related. Episodic long-term working memory may be ubiquitous in complex task environments with large amounts of task-related external information.

Appendix A

Model diagram

The model's behavior has three top-level components: knowledge, an underlying cognitive architecture, and mechanisms that let the architecture manipulate the knowledge. This appendix reviews the interaction of these components, by tracing through a figure showing the complete model (Figure 37).

The model and display emulator are embedded in Soar, represented by the two full-width boxes in the figure. Soar's WM encloses the upper components of the figure. Soar's WM contains the model's WM, selected goals and subgoals, proposed goals and subgoals, and the emulated display. The model and the display are partitioned into different areas of Soar's WM. The only communication across the partition is through attend subgoals, fixate subgoals, and the feature memories that Soar encodes from fixate subgoals.

Soar's LTM encloses the lower components in the figure. Soar's LTM contains all the model's long-term knowledge, as well as rules that implement the model's mechanisms and the display emulator (the display emulator rules are not shown). The model's WM is implemented so that any element retrieved by an information-retrieval subgoal causes Soar to encode a new rule. Any number of rules can fire in parallel, though goals and subgoals (both represented as Soar operators) are selected one at a time.

The upper left of the figure contains a legend for the arrows that connect parts of the model. Grey-outline arrows represent the use of WM elements, goals, and subgoals to activate knowledge in LTM. Thin solid arrows represent retrieval of information to WM, and to the buffers that contain proposed and selected goals and proposed and selected subgoals. Dashed arrows, which occur only in WM, represent the addition of new elements to WM, which triggers encoding of new memories in LTM. Finally, heavy solid arrows represent the encoding of memories in LTM.

We trace the functioning of the model starting with the left part of the figure, which shows the retrieval and encoding of information (Section A.1). We then turn to the right part of the figure, which shows the selection of comprehension and command goals (Section A.2). Lastly, we look at the center of the figure, which shows how the model uses working memory (Section 3.8).

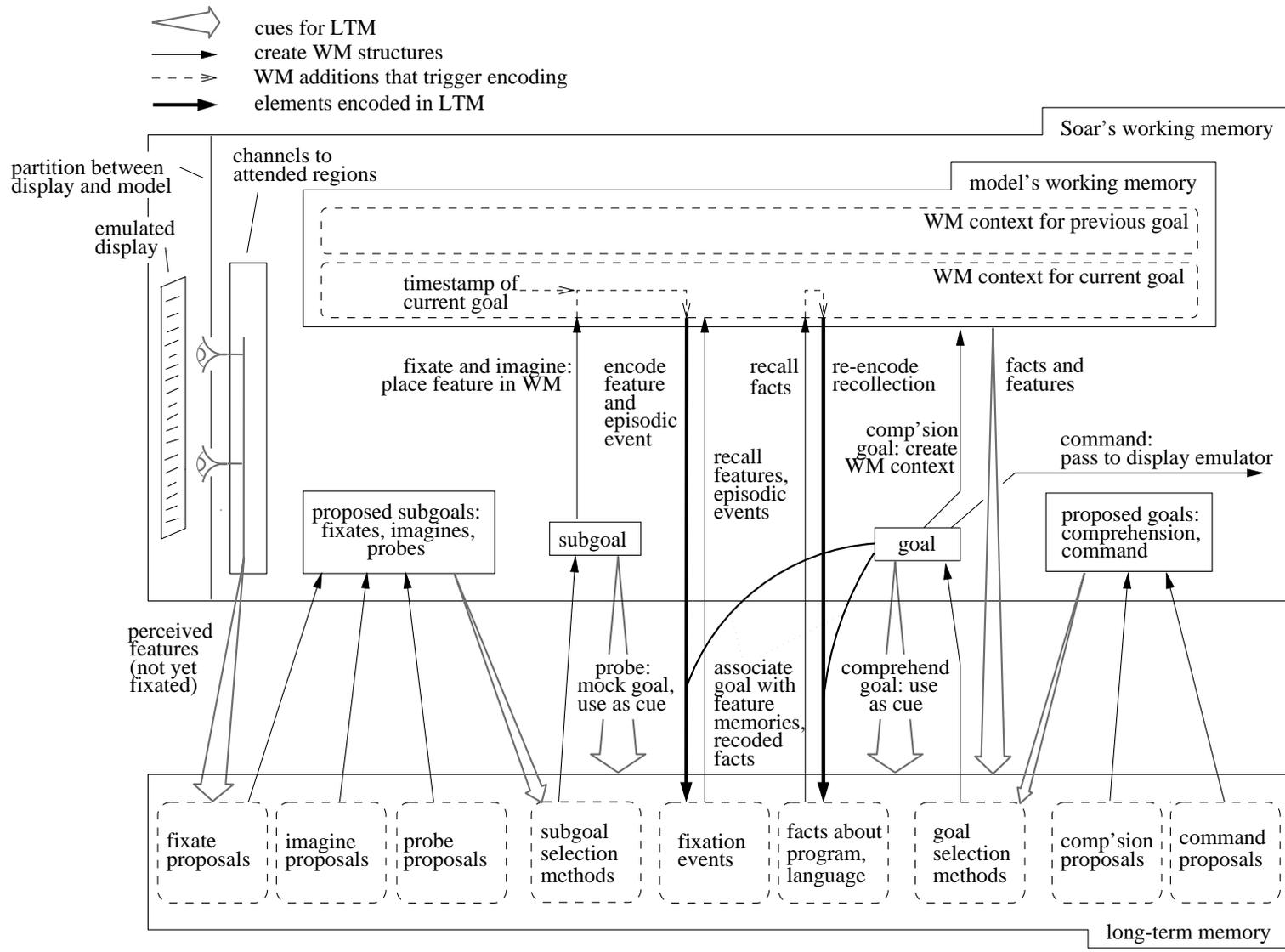


Figure 37: Complete diagram of the model

A.1. Retrieving information from display and LTM

Beginning at the left of the figure, the model's fixation knowledge (Section 3.4.2) perceives features on the emulated display, through pointers (eyes) created by attend subgoals (Section 3.4.1). The model proposes fixation subgoals based on these features, as well as on the current comprehension goal, retrieved information in WM, and knowledge in LTM about what features are important to look at given the language and the program. The model also proposes probe and imagine subgoals (Sections 3.5.1 and 3.5.2), based on the current goal, information in WM, and knowledge in LTM.

From the set of proposed information-retrieval subgoals, the model selects one using its subgoal-selection methods. These methods consist of both general heuristics and feature-dependent preferences (Section 3.6.2).

If the selected subgoal is a fixate, the model places the feature into WM, adding the timestamp of the current comprehension goal. The model encodes the memory for the feature so as to index it to the current goal. Whenever the feature is on display and the same goal is selected, the model will place that feature in WM without having to select a fixate subgoal. The model encodes the memory for the timestamp so as to index it to the feature. If the model fixates or imagines that feature in the future, it will recall all previous occasions on which it saw that feature.

An imagine subgoal is like a fixate subgoal, in that it places a feature in WM. It differs in that it retrieves a feature from LTM rather than the display. Also, the model imagines a feature not to retrieve the feature itself, but to retrieve episodic information about the feature available. Such information tells the model that it saw the feature before.

Selecting a probe causes the model to retrieve knowledge about the probe object, as well as feature memories indexed to that object. The probe object mimics a goal, serving to retrieve facts and features that would be retrieved were it selected as a goal. The model re-encodes the retrieved elements, creating a new link from the current goal to the retrieved element and increasing the semantic connectedness of the model's knowledge about objects.

When one subgoal is complete, the model selects another, until the goal-selection methods select a new comprehension or command goal.

A.2. Selecting comprehension goals and commands

At the right of the figure, the model proposes objects to comprehend and commands to change the display. The model selects the next goal from this proposed set. It immediately selects any proposed command (Section 3.7.1), and the display emulator responds by representing the changes that occurred on the programmer's display. After a command, the model re-selects the comprehension goal that preceded the command goal. This lets the model continue to accumulate information about the object for which it generated the new external knowledge.

The model selects a new object to comprehend when a new object is proposed as a goal, and when converging evidence supports the relevance of this object (Section 3.6.1). Converging evidence consists of an information-retrieval subgoal (fixate, imagine, or probe) that corresponds to the proposed object.

A.3. Constructing and reconstructing working memory

When the model selects a new object to comprehend, two important events occur with respect to WM. First, the model creates a new context in WM to accumulate information about that goal. Second, the object becomes a cue for knowledge in LTM about that object. The model recalls all the expert knowledge it has about that object, as well as memories that associate that object with any features examined previously when that object was the goal.

WM is the union of the two most recent contexts in WM (Section 3.5.4). Older contexts drop out of WM, taking the elements in those contexts with them. The model continually reconstructs its WM by retrieving information from the display and from LTM, using goals and information-retrieval subgoals.

The model learns continually, increasing the number of paths of access to its knowledge. This means that the model gains quicker access to its knowledge over time. When an object is selected as a goal or probe, the model recalls immediately what it had to search for previously. The more the model returns to thinking about an object, the more information about that object is immediately retrieved to WM.

Appendix B

Mapping the model to Soar

This appendix describes some key elements of the model in Soar terms. Section B.1 describes the motivation behind our vocabulary for describing the model, given that vocabulary departs from standard Soar terminology. Section B.2 describes in general what a Soar operator trace looks like and what parts of it correspond to model goals and subgoals. These sections provide prerequisite information for interpreting the detailed traces for the 5 scrolling events (Appendix D) and the rule-firing trace for the entire life of the model (Appendix F).

Section B.3 describes how the goal-selection mechanism maps onto the selection of Soar operators, and discusses in detailed terms how the architecture fails to constrain goal selection. Section B.4 is a guide to the implementation of the fixation and imagining mechanisms for readers interested in the code.

B.1. The vocabulary problem in describing the model

The model we describe is a Soar model of someone comprehending a Soar model of comprehension. The need to describe both levels of Soar model in detail introduced a vocabulary problem. Hence, our model is the *model*, and the programmer we studied works on a *program*. The protocol data “uses up” much of the standard Soar vocabulary, including *problem space*, *state*, and *operator*, so we avoided these terms in our descriptions of the model.

The programmer does not use *goal* or *subgoal*, leaving these terms available for describing the model. We chose them because their usage in the context of the model is consistent with general usage in AI and cognitive science. A goal is something to achieve, and a subgoal contributes to achieving a goal. The model tries to achieve comprehension goals, by retrieving information with subgoals. The model achieves command goals directly. The next section describes how goals and subgoals map to Soar operators.

B.2. Goals, subgoals, and Soar operators

The model uses the top and second levels of the Soar goal stack. (Models that use the Soar goal stack to represent *universal subgoaling* (Laird, 1984) treat the stack as arbitrarily deep.) The model's goals (comprehension and command) correspond to top-level Soar operators. The model's subgoals (attend, fixate, imagine, and probe) correspond to second-level Soar operators. A second-level context arises in response to a Soar *impasse* (Laird et al., 1993) on a comprehension operator.

A goal-stack depth of two is the minimal depth that implicates learning. Soar learns automatically when a newer context generates information visible in an older context (Laird et al., 1993, Altmann and Yost, 1992). The model's information-retrieval subgoals, which occur in second-level contexts, add information to the model's WM. The model's WM is also visible in Soar's top context. This causes Soar to encode any information added to the model's WM. An index to the rules that implement the model's WM appears on page 125.

Below are three operators from a Soar operator trace (taken from the first detailed trace Appendix D, p. 179). The first operator (`display`) represents a command goal. Attached to this operator is the time of the corresponding programmer command in the protocol (`t225`; the protocol appears in Appendix E). The second operator (`comprehend`) represents a comprehension goal. The third operator (`fixate`) is a fixate subgoal selected in service of the comprehend goal. Subgoals are indented to the right of goals.

```
13 0: 028 (display match-set:after-selection (t225))
...
14 0: 026 (comprehend create-referent :selected-op)
...
17   0: 040 (fixate create-referent :argument cop)
```

B.3. The goal selection mechanism

The model's goal-selection mechanism depends on Soar's distinction between proposing and selecting an operator. We first need to clarify what we mean by an operator. In Soar terms, *operator* can refer to both a *type* and a *token*. An operator type is what is specified by the right-hand side of an operator-proposal rule. An operator token is an object that finally ends up in Soar's WM. If an operator-proposal rule fires multiple times, each firing proposes the same type but different tokens. Below *operator* means *operator token*, unless otherwise noted.

An operator has two representations in Soar's WM: *proposed* and *selected*. Soar selects the next operator by choosing from the set of proposed operators, taking into account any preferences asserted for them by search-control rules. An operator stays proposed as long as the conditions of the proposal rule continue to be satisfied.

Both representations are necessary for various kinds of functions. For example, search-control rules need access to proposed operators to be able to compare them. On the other hand, many rules depend on what the currently-selected operator is. For example, the model's semantic knowledge (facts about objects) tests the `^goal` attribute of the selected top-level operator (the comprehension goal).

The model's goal-selection mechanism selects a proposed goal when Soar selects a matching information-retrieval subgoal (we now refer to our goal/subgoal vocabulary for describing the model). Below is an example.

Suppose that fixating feature1 causes object2 to be proposed as a goal. As long as feature1 is in WM, object2 will remain proposed:

```
O: O1 (comprehend object1)
==>S: S2 (operator no-change impasse)
  O: O2 (fixate feature1)
proposed goals include object2
  O: O3 (fixate feature2)
proposed goals include object2
  . . .
```

Also, suppose that feature1 causes object2 to be proposed as a probe subgoal. Object2 may be an "important object" that the model knows it may be useful to probe with. (The rule a*state*important-objects, p. 172, creates a table that describes what objects are important. This rule is shown in action in the recall episode of scrolling event 2, followed by another rule that looks up a specific object to see whether it's important; see Section D.2, p. 183. These two rules together make up rule 3 in the corresponding abstract trace, p. 53.) As long as feature1 is in WM, and as long as the current goal persists, the object2 probe will stay proposed:

```
O: O1 (comprehend object1)
==>S: S2 (operator no-change impasse)
  O: O2 (fixate feature1)
proposed goals include object2
  proposed subgoals include object2
  O: O3 (fixate feature2)
proposed goals include object2
  proposed subgoals include object2
  . . .
```

Eventually the model may select the object2 probe from the set of proposed subgoals. The model's subgoal-selection heuristics exercise some control over this, but often Soar has to make a random choice from an equivalent set of subgoals. The model makes this possible by generating indifferent preferences automatically for all subgoals. Soar processes indifferent preferences after it processes all others. If preferences declare one subgoal "better" than another, the worse one will never be chosen while the better one is proposed. But the asserted "better" (and "best") preferences may only generate a partial order, and leave any number of subgoals in the most-preferred equivalence class. This would cause an impasse in Soar (specifically, an operator-tie impasse) if the model didn't make the members of this class "indifferent".

Now the model has selected the object2 probe subgoal. This triggers the model's goal-selection rules, because the selected subgoal matches a proposed goal. Object2 being selected as a probe counts as "converging evidence" (Section 3.6.1, p. 35) that object2 is relevant to the model's train of thought, so the model selects object2 as the goal. The goal-selection rules (indexed on page 125) issue the necessary preferences to replace object1 (operator O1) with object2 (operator O15):

```

O: O1 (comprehend object1)
==>S: S2 (operator no-change impasse)
      O: O2 (fixate feature1)
      proposed goals include object2
      proposed subgoals include object2
      O: O3 (fixate feature2)
      proposed goals include object2
      proposed subgoals include object2
      ...
      O: O14 (probe object2)
O: O15 (comprehend object2)

```

When Soar selects a new comprehension goal, a number of things happen. It de-selects the old goal, so all rules that proposed subgoals for the old goal retract and the corresponding subgoals vanish. All memories conditional on the new goal have an opportunity to fire, if their other conditions are also met. This includes all semantic knowledge about the goal, as well as feature memories encoded when that goal was selected previously. The retrieved information accumulates on the new goal, and the working-memory mechanism (also indexed on p. 125) shifts the 2-goal window (contained on the `^applied` attribute) to replace the older goal with the new one.

B.3.1. Discussion: Goal selection constrained by data, not architecture

A comprehension goal and its information-retrieval subgoals are related by the Soar *impasse* mechanism. Abstractly, an impasse represents a lack of knowledge, and resolving it requires generating the knowledge that is lacking. More concretely (Laird et al., 1993), an impasse means that Soar cannot apply the current operator in one decision cycle (an *operator no-change* impasse), or cannot make a unique choice for a next operator (an *operator tie* impasse).

A comprehension goal results in an operator no-change impasse. By the model's definition of comprehension, knowledge is always lacking about the goal object, and hence an impasse always occurs. (Comprehension does not proceed like this in all Soar models. For example, in NL-Soar (Lewis, 1993)), comprehension of most words occurs immediately, reflecting the competent reader's ability to comprehend recognitionally.)

The architecture and the cognitive theory behind it (Newell, 1990) give little specific guidance about how to resolve an impasse. An operator no-change impasse (for example) merely reflects the absence of a preference signalling that the current operator is complete and that its selection should be reconsidered (the preference is called *reconsider*). An impasse is resolved when some rule reconsiders the operator on which the impasse occurred. Soar says little about the conditions of these rules. In particular, it says little about how much new information is "enough" to call an object comprehended and move on to the next one, and how to decide what object to select next (Section 6.4).

The data shaped the model's goal-selection mechanism. In the volume of data our model emulates, the programmer thinks about a lot of different objects, but her selection of these objects is not random. The general pattern we observed in the protocol is that one thing leads to another — thinking about one object

leads to retrieval of information about that object which in turn inspires the next object to think about. A model mechanism reflecting this pattern had to allow retrieved information to inspire the next goal. This inspiration is reflected by the "converging evidence" criterion, in which an information-retrieval subgoal leads to the selection of the next goal.

B.4. Guide to fixate and imagine mechanisms

The fixation and imagining mechanisms are central to our hypotheses about encoding and retrieval of episodic memory. This section gives a brief Soar-level guide to how they are implemented, focussing on how it's done rather than why it's done that way. Here we abandon the goal/subgoal terminology used elsewhere to describe the model, and speak in terms of operators.

Fixation and imagine operators are proposed by rules that represent domain knowledge. This knowledge determines when the fixated or imagined feature would be relevant. In the case of fixation proposals, the rules also perceive elements of information on the display, by following a display pointer in WM ($\wedge_{wm}.dp$). Fixate and imagine proposals are both contained in the file `fixate.soar`, which begins on page 156 with a contents listing.

When Soar selects a fixate operator, the model's fixation mechanism applies the operator in two phases. (All mechanisms are contained in the file `mechanism.soar`, beginning on page 165 with contents listing.) Phase 1 of fixation (`ao*fixate`, p. 167) brings the feature into a special region of WM called *fixation memory*. This consists of tuples attached to the $\wedge_{wm}.fixated$ attribute. Each tuple contains the feature and the region it came from. (The region information is perceived by every fixate proposal and is a parameter of every fixate subgoal. It is used by the novel-region heuristic; `p*fixate*newest*interleave-best`, 168. It is also used by the attention mechanism to determine when the region a feature came from is no longer in display; `ao*fixated-recently`, p. 170.) Phase 1 encodes a new rule for every parameter of the fixate operator that makes up the feature. These new rules are feature memories. An example of a feature memory appears on page 180, in the context of a detailed trace of scrolling event 1.

Phase 2 of fixation (`ao*fixate*unpack-fixation-object`, p. 167) carries out two tasks. It "unpacks" the feature from fixation memory into WM proper, making the feature available as a cue for knowledge in LTM. At the same time, it tags the tuple with a timestamp. These two activities create one new rule, which is the episodic memory. An example of an episodic memory also appears on page 180.

Whenever a feature appears in fixation memory, whether through a fixate subgoal, the firing of a feature memory, or the selection of an imagine subgoal, the episodic memory will fire, simultaneously making the feature available in WM and timestamping the tuple. Thus episodic memories for features fire much more often than the model imagines features (271 times compared to 25 times; see Figures 29, p. 75, and Figure 32, p. 81).

The model makes use of the episodic timestamp only if it was an imagine operator that cause the timestamp to be retrieved. An imagine operator places a feature into fixation memory (`ao*imagine`, p. 170). If this

activates an episodic memory, that tuple in fixation memory will be immediately timestamped with a time that is not the current time. If such a timestamp appears and the same feature was also imagined, the model places the feature on the `^imagined-but-seen` attribute (`ao*imagine*imagined-but-seen`, p. 170). This attribute is a cue for the model's scrolling knowledge (`po*display*scroll*to-sp`, p. 143; `po*display*scroll*to-state`, p. 143; and `po*display*scroll*to-object`, p. 149).

The timestamp used during fixation is unique to the current comprehension operator, generated automatically after each new impasse arises (`a*substate*create-time`, p. 171). This timestamp reflects an inherent ability in Soar to denote new events, by generating unique symbols in response to those events (with the `make-constant-symbol` function call on the right-hand side of a rule). To refer to these denotations as timestamps may suggest that they carry more information than they do. Soar's condition-matching syntax affords only a not-equal test (used in `ao*imagine*imagined-but-seen`, introduced above). Any richer episodic structure would require the implementation of other cognitive mechanisms using Soar rules.

B.5. Data chunking: deliberate learning of generalized rules

By default, Soar learns rules that, in order to fire, require cues from the encoding context to be present in WM. These rules are recognitional in that they recognize aspects of the encoding context, such as an external stimulus. Recognitional learning is thus easy for Soar, as it is easy for people. Recall learning is more difficult for Soar, as it is more difficult for people. Soar uses a generate-and-recognize process called *data chunking* to learn recall rules — rules that associate the to-be-retrieved element with a cue other than the element itself. A detailed illustration of data chunking appears in Rosenbloom, Laird, and Newell (1987). Here we present a simple example, to ground the concept. Below is a specification for how a recall model, which we call R, might be implemented in Soar.

Suppose that R is presented with the task to recall a binary string 11, on the cue `recall`.

```
recall → 11
```

R must first learn to recognize the stimulus, for which it would invoke Soar's inherent capability to denote a new event. It would invoke this capability by calling a built-in Soar function called `make-constant-symbol` on the right-hand side of a rule. This function generates a new, unique symbol and places it into WM. For example, our programmer model calls this function to generate a new name for every comprehension goal (`a*substate*create-time`, p. 171). This name constitutes the episodic timestamp assigned to each feature fixated in service of that comprehension goal.

To learn to recognize the stimulus 11, R would learn the rule below, where `name-1` was a symbol generated by `make-constant-symbol`. The string 11 could be the name of a Soar operator that impasses. In the impasse context, R would call `make-constant-symbol` and return the resulting name as the result of the operator. This would cause Soar to learn a rule of the form:

```
11 → name-1
```

To learn to recall the stimulus, R would next have to use prior knowledge to reconstruct (or generate) the stimulus in WM, using the recognition rule above to tell when it had succeeded. It might select an operator called `recall`, which would impasse. In the impasse context, the model would reconstruct the stimulus. One approach to this reconstruction would be to use prior knowledge of the components of the stimulus. So, for example, R might first recall the binary digits:

```
recall → 0
recall → 1
```

The model might then combine them in all possible strings of length two. When it finally generated 11, the recognition rule from above would fire, retrieving the name for that string. This firing constitutes recognition of the to-be-learned element.

```
0 → 00          no recognition rule fires
0 1 → 01        no recognition rule fires
0 1 → 10        no recognition rule fires
1 → 11          recognition rule fires on cue 11, retrieving name-1
```

At this point R knows what it needs to know to learn a recall rule. It knows that 11 is the stimulus it was asked to learn to recall. If it returns 11 as the result of the recall operator, Soar will learn a rule of the form:

```
recall → 11
```

This is the desired recall rule. However, learning it required prior knowledge of the components of the stimulus, begging the question of how the model might have acquired this component knowledge. This recursion seems to be one instance of the symbol-ground problem (Harnad, 1990): at what point is this prior knowledge grounded in something fundamental, that the model is either born with, or presented with as a raw signal from the environment?

In the context of our model, the question arises how this process might be used to learn to imagine. Currently, the imagine operator, like all information-retrieval operators, applies immediately and never results in an impasse. To learn to imagine a feature, the model could select an imagine operator that results in an impasse (as the recall operator did in R, above). In the impasse, the model would have to reconstruct to-be-imagined feature, using knowledge of the constituent components of features (whatever those might be). To recognize the reconstructed feature, the model could use a recently-constructed episodic memory.

Appendix C

Model code and indices

This appendix contains all the code for the model, as well as several ways to index it and get an overview. Section C.1 (p. 124) lists rules and page numbers by category of knowledge and mechanism. The categories are those of Figure 29 (p. 75), which presented rule and firing counts in each category.

Section C.2 (p. 126) maps the commands actually issued by the model (Figure 26, p. 70) to the rules that propose them, and lists how many commands are selected due to each rule.

Section C.3 (beginning on p. 127) is the model source code. The organization of the code does not exactly parallel the categories of mechanism and knowledge used throughout the thesis. There are four files:

- `comprehend.soar` (p. 127)
Contains comprehension-goal proposals (prefix `po*comprehend`, for *propose operator*) and facts (`f:`).
- `display.soar` (p. 136)
Contains command-goal proposals (`po*display`) and most of the display emulator (`a*display` rules *augment* display operators that are hits, `ao*display` rules *apply* display operators, and `d*display` rules respond by changing the emulated *display*).
- `fixate.soar` (p. 156)
Contains fixate proposals (`po*fixate`), imagine proposals (`po*imagine`), and specialized fixate search-control rules (`p*fixate`, for *preference*).
- `mechanism.soar` (p. 165)
Contains everything else, which is primarily mechanistic, but also includes (for example) the knowledge for attend proposals.

Each file begins with a table of contents.

Finally, Section C.4 contains an alphabetical index of all rule names.

C.1. Table of rules by knowledge and mechanism category (Figure 29, p. 75)

Expert knowledge

Attend proposals

po*attend.....	166
po*attend*old-regions.....	167

Fixate proposals

po*fixate*condition.....	157
po*fixate*action.....	157
po*fixate*binding-attribute*target.....	157
po*fixate*binding-context.....	157
po*fixate*current-context.....	158
po*fixate*no-referent.....	158
po*fixate*where-was-i*chunks-built.....	159
po*fixate*where-was-i*assertions.....	159
po*fixate*builds.....	159
po*fixate*chunk.....	159
po*fixate*no-problem-space.....	159
po*fixate*shared-state.....	160
po*fixate*current-context*shared-state.....	160
po*fixate*chunk*second.....	160
po*fixate*chunks-retracting.....	161
po*fixate*no-chunk.....	161
po*fixate*selected-operator.....	161
po*fixate*argument.....	161
po*fixate*previous-argument.....	161
po*fixate*state-id.....	162
po*fixate*id-of-imagined-object.....	162
po*fixate*augmentation.....	162
po*fixate*two-valued-attribute.....	163
po*fixate*superstate-id.....	163
po*fixate*annotations.....	163
po*fixate*proposal-context.....	163
po*fixate*assertion.....	163
po*fixate*binding-attribute*param.....	164
po*fixate*bind-op*space.....	164
po*fixate*selected-id.....	164
po*fixate*bind-op*id.....	165

Probe proposals

po*probe*where-was-i.....	172
po*probe*with-previous-goal.....	172
a*state*important-objects.....	172
po*probe*with-important-object.....	172
po*probe*with-high-level-goal.....	172
po*probe*with-attribute.....	173
po*probe*with-part.....	173

Imagine proposals

po*imagine*nil-object.....	158
po*imagine*postpone.....	158
po*imagine*operators.....	158
po*imagine*operator-targets.....	159
po*imagine*assertions.....	160
po*imagine*action.....	160
po*imagine*actions-refract.....	161
po*imagine*sp-causes-builds.....	162
po*imagine*referent.....	162
po*imagine*s-model-constructor.....	165

Facts about objects

f:apply-sps.....	128
f:recognize-u-model-object.....	128
f:recall-for-part-of-u-model.....	128
f:high-level-goal-cues.....	129
f:select-exhausted.....	129
f:postpone-return-new-pointer.....	129
f:shared-states-build-chunks.....	130
f:superstate-is-shared-too.....	130
f:sp.....	130
f:abstract-sp.....	131
f:operator-condition*part-of-lhs.....	131
f:operator-condition*lhs-means-rhs.....	131
f:terminate-create-referent.....	131
f:sps-propose-add-property.....	132
f:add-property-target.....	132
f:nil-chunk-is-abstract.....	132

f:apply-add-property-target.....	132
f:terminating-sps.....	132
f:recall-u-model.....	132
f:u-model.....	133
f:exhausted-builds-proposal.....	133
f:operator.....	133
f:sp-parts.....	134
f:propose-return-operator.....	134
f:return-operator-target.....	134
f:proposal-build-action.....	134
f:pay-attention-when-building-proposal.....	134
f:s-construct-builds-chunk.....	135

Comprehension-goal proposals

po*comprehend*init.....	128
po*comprehend*selected-operator.....	128
po*comprehend*current-context-when-exhausted.....	129
po*comprehend*actual-context.....	129
po*comprehend*where-was-i.....	130
po*comprehend*superstate-target.....	130
po*comprehend*imagined-chunk-cause.....	130
po*comprehend*chunk.....	130
po*comprehend*sp-parts.....	131
po*comprehend*operator-condition.....	131
po*comprehend*builds.....	131
po*comprehend*building-agent.....	132
po*comprehend*u-model.....	133
po*comprehend*high-level-goal.....	133
po*comprehend*assertion*real.....	133
po*comprehend*assertion*imagined.....	133
po*comprehend*superstate.....	134
po*comprehend*proposal-context.....	134
po*comprehend*assertion*pay-attention.....	134
po*comprehend*assertion*pay-attention*fixated-recently.....	135
po*comprehend*new-operator.....	135
po*comprehend*objects-attribute.....	135
po*comprehend*recalled-condition.....	135

Fixate preferences

p*fixate*superstate.....	159
p*fixate*u-something.....	162
p*fixate*nil-assertion-worst.....	164
p*fixate*assertion*pay-attention.....	164
p*fixate*operator-id.....	164

Command-goal proposals

po*display*match-set*after-selection.....	138
po*display*print-sp*applies-operator.....	139
po*display*print-operator.....	139
po*display*print-object.....	140
po*display*print-stack.....	141
po*display*run*where-was-i.....	141
po*display*print-chunk.....	142
po*display*scroll*to-sp.....	143
po*display*scroll*to-state.....	143
po*display*match-set*asserted-sp.....	144
po*display*run-to-builds.....	145
po*display*match-set*after-builds.....	145
po*display*run*to-op-after-builds.....	147
po*display*run*to-end-of-space.....	148
po*display*scroll*to-object.....	149
po*display*print-object*fresh.....	150
po*display*run*action-and-print.....	152
po*display*print-sp*when-paying-attention.....	153
po*display*print*high-level-goal.....	154
po*display*run*to-expected-op.....	155

Mechanisms

Attend subgoal

ao*attend.....	166
ao*attend*previous-not-newest.....	166
ao*attend*mark-locally.....	166
a*wm*newest-from-not-newest.....	166
ao*attend*old-regions.....	167

Fixate subgoal

a*state*fixate-meta-attributes.....	167
ao*fixate.....	167
ao*fixate*unpack-fixation-object.....	167
ao*fixate*mark-imagined-object.....	168
ao*comprehend*unpack-fixation-object.....	168

Imagine subgoal

ao*imagine.....	170
ao*imagine*imagined-but-seen.....	170
ao*fixated-recently.....	170

Working memory

ao*comprehend*create-dp-on-om.....	174
ao*comprehend*cleanup-naked-region-pointers.....	174
ao*comprehend*applied*first.....	174
ao*comprehend*applied*second.....	174
ao*comprehend*applied.....	174
a*state*applied-newer.....	175
a*wm*unpack-applied-om.....	175
a*dp*unpack-applied-om.....	175
a*subgoal*wm-pointer.....	175
a*subgoal*hold-back-wm-pointer-until-attend.....	175
ao*probe*unpack-probe-om-to-superop-om.....	175
ao*probe*unpack-probe-om-to-superop-om*fixated.....	175

Comprehension-goal selection

a*goal-select*proposed-during.....	169
ao*goal-select*select-now.....	169
ao*goal-select*comprehend*create-token.....	169
ao*goal-select*new-goal*fixate/imagine.....	170
ao*goal-select*new-goal*probe.....	170

Subgoal selection

p*attend*old-regions*reject.....	167
a*fixate*dont-interleave-best.....	168
a*fixate*newest.....	168
p*fixate*interleave-best.....	168
p*fixate*newest*interleave-best.....	168
p*fixate*top-down.....	168
p*fixate*bottom-up.....	168
p*fixate*invariant-feature*dont-interleave-best.....	168
p*fixate*fixated-recently-in-view*best.....	169
p*imagine*refract.....	170
p*imagine*worst-after-new-output.....	171
p*generic*indifferent.....	171
p*probe*repeated-goal*worst.....	173
p*probe*new-important-object*best.....	173
p*probe*new-attribute*best.....	173
p*probe*non-important-after-imagine*worst.....	173
p*probe*fixated-recently*best.....	173
p*probe*fixated-recently*worst.....	173
p*probe*where-was-i*best.....	173
a*state*new-important-object*best.....	173
p*probe*retrieved-by-probe*best.....	174
p*probe*new-high-level-goal*best.....	174
p*probe*rhs-when-sp.....	174
p*probe*lhs-best.....	174
p*probe*rhs-better-when-apply-sp.....	174

Command-goal selection

ao*comprehend*remember-display-command.....	169
p*comprehend*best*when-for-last-displayed-region.....	169
p*display*dunk-comprehend.....	169
p*display*reject-duplicates.....	169

Shared rules

a*substate*create-time.....	171
a*substate*initialize-goal-set.....	171
ao*probe*goal.....	171
p*generic*reject.....	171
p*generic*terminate-and-reject.....	171
a*topstate*create-display-wm-dp-time-dummy.....	171
a*topstate*clean-up-old-comprehends-and-displays.....	171
ao*substate*count-first.....	171
ao*substate*count-second.....	171

C.2. Table of model commands mapped to proposal rules

unit commands			
compound commands			
slips/disregarded/missed			
v	v v	model commands	proposal rule (page)
1		match-set after-selection	po*display*match-set*after-selection 138
2		print-sp create-referent	po*display*print-sp*applies-operator 139
3		print-operator o25	po*display*print-operator. 139
4		print-object u20	po*display*print-object. 140
5		print-stack	po*display*print-stack 141
6		run where-was-i	po*display*run*where-was-i 141
	s		
7		print-chunk chunk-128	po*display*print-chunk 142
8		scroll to-sp	po*display*scroll*to-sp. 143
9		scroll to-state	po*display*scroll*to-state 143
	d		
10		print-chunk chunk-129	po*display*print-chunk 142
11		match-set asserted-sp	po*display*match-set*asserted-sp 144
12		run to-builds	po*display*run-to-builds 145
13		match-set after-builds	po*display*match-set*after-builds. 145
	m		
14		match-set after-selection	po*display*match-set*after-selection 138
15		run to-builds	po*display*run-to-builds 145
16		match-set after-builds	po*display*match-set*after-builds. 145
	1	run to-op-after-builds	po*display*run*to-op-after-builds. 147
	m		
17		match-set after-selection	po*display*match-set*after-selection 138
	2	run to-end-of-space	po*display*run*to-end-of-space 148
18		print-stack	po*display*print-stack 141
	3	scroll to-object	po*display*scroll*to-object. 149
	d		
19		print-object-fresh u20	po*display*print-object*fresh. 150
20		print-object r9	po*display*print-object. 140
21		match-set after-selection	po*display*match-set*after-selection 138
22		print-sp impl't-exhausted	po*display*print-sp*applies-operator 139
	4	run action-and-print	po*display*run*action-and-print. 152
23		match-set asserted-sp	po*display*match-set*asserted-sp 144
24		print-stack	po*display*print-stack 141
25		print-sp term-s-model-con	po*display*print-sp*when-paying-attention. 153
	5	scroll to-sp	po*display*scroll*to-sp. 143
26		print propose-retrn-op	po*display*print*high-level-goal 154
	6	run to-expected-op	po*display*run*to-expected-op. 155
	s		
27		print-operator o29	po*display*print-operator. 139
	7	scroll to-sp	po*display*scroll*to-sp. 143
Proposal		po*display*match-set*after-selection	4 No. command goals
rule		po*display*scroll*to-sp	3 selected due to
		po*display*print-stack	3 proposal rule
		po*display*match-set*after-builds	2
		po*display*match-set*asserted-sp	2
		po*display*print-chunk	2
		po*display*print-object	2
		po*display*print-operator	2
		po*display*print-sp*applies-operator	2
		po*display*run-to-builds	2
		po*display*print-sp*when-paying-attention	1
		po*display*print-object*fresh	1
		po*display*print*high-level-goal	1
		po*display*run*action-and-print	1
		po*display*run*to-end-of-space	1
		po*display*run*to-expected-op	1
		po*display*run*to-op-after-builds	1
		po*display*run*where-was-i	1
		po*display*scroll*to-object	1
		po*display*scroll*to-state	1

C.3. Model code

C.3.1. comprehend.soar

```
po*comprehend*init..... 128
po*comprehend*selected-operator..... 128
f:apply-sps..... 128
f:recognize-u-model-object..... 128
f:recall-for-part-of-u-model..... 128
f:high-level-goal-cues..... 129
f:select-exhausted..... 129
po*comprehend*current-context-when-exhausted..... 129
po*comprehend*actual-context..... 129
f:postpone-return-new-pointer..... 129
po*comprehend*where-was-i..... 130
po*comprehend*superstate-target..... 130
f:shared-states-build-chunks..... 130
f:superstate-is-shared-too..... 130
po*comprehend*imagined-chunk-cause..... 130
po*comprehend*chunk..... 130
f:sp..... 130
f:abstract-sp..... 131
po*comprehend*sp-parts..... 131
po*comprehend*operator-condition..... 131
f:operator-condition*part-of-lhs..... 131
f:operator-condition*lhs-means-rhs..... 131
po*comprehend*builds..... 131
f:terminate-create-referent..... 131
f:sps-propose-add-property..... 132
f:add-property-target..... 132
po*comprehend*building-agent..... 132
f:nil-chunk-is-abstract..... 132
f:apply-add-property-target..... 132
f:terminating-sps..... 132
f:recall-u-model..... 132
po*comprehend*u-model..... 133
f:u-model..... 133
f:exhausted-builds-proposal..... 133
po*comprehend*high-level-goal..... 133
f:operator..... 133
po*comprehend*assertion*real..... 133
po*comprehend*assertion*imagined..... 133
f:sp-parts..... 134
po*comprehend*superstate..... 134
f:propose-return-operator..... 134
f:return-operator-target..... 134
po*comprehend*proposal-context..... 134
f:proposal-build-action..... 134
```

```
f:pay-attention-when-building-proposal..... 134
po*comprehend*assertion*pay-attention..... 134
po*comprehend*assertion*pay-attention*fixated-recently..... 135
po*comprehend*new-operator..... 135
po*comprehend*objects-attribute..... 135
f:s-construct-builds-chunk..... 135
po*comprehend*recalled-condition..... 135
```

```
;; -*- Mode: Sde -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; File           : comprehend.soar
;; Author        : Erik Altmann
;; Created On    : Mon Jun 20 16:08:49 1994
;; Last Modified By: Erik Altmann <altmann@electro.soar.cs.cmu.edu>
;; Last Modified On: 02 Jul 1996, 16:44:08
;; Update Count  : 1434
;;
;;
;; CONTENTS:
;;
;; This file contains two kinds of productions: (1) proposals for the
;; comprehend operator, and (2) "facts" that are like the
;; apply-operator chunks built by the model beneath comprehend
;; operators, but are written by me.
;;
;; Both comprehend proposals and facts represent knowledge we suppose
;; our programmer brought to her task (including language and program
;; expertise).
;;
;; Comprehend proposals ("po*..."):
;;
;; Comprehend proposals represent knowledge about what program
;; objects and abstractions are important to think about in various
;; contexts. The proposals are general in that they test very few
;; constant values in WM. They fire in the top context only.
;; (Goals that occur as probes in the subgoal arise by other means;
;; see mechanism.soar.)
;;
;; In a comprehend subgoal, the model can look at the display (see
;; fixate.soar for the knowledge about what's important to look at
;; and how to encode it) and probe LTM (see mechanism.soar for the
;; probing mechanism). If looking at the display or probing LTM
;; retrieves knowledge not already in WM, the model builds new
;; chunks. A goal terminates when there is alignment between some
;; new goal proposed in the top context and the train of thought in
;; the subgoal. When a probe or fixate operator in the subgoal has
;; this new proposed goal as a parameter, then the goal-selection
;; mechanism switches to the new goal. The model thinks about
;; something new when that thing is both relevant in the current
;; context (ie, proposed in the top context) and the model attends
;; to it specifically (through the probe or fixation).
;;
;; Facts("f:..."):

```

```

;;
;; Facts represent associations between specific symbols. The cues
;; on the LHS retrieve the knowledge on the RHS. Like the chunks
;; built during comprehension subgoals, they are specific, in that
;; they test only constant values in WM. They fire when comprehend
;; goals are selected in the top context and as probes in the
;; subgoal.
;;
;; Many facts test only the goal, and nothing in WM. By default
;; Soar classifies these as elaboration sps. The ":o-support" flag
;; on these productions overrides this.
;;
;; ORGANIZATION:
;;
;; Productions occur roughly in firing order. They are grouped into
;; emacs pages (^L), also roughly, with an excerpt from the
;; corresponding vicinity of the protocol.
;;
;; Many productions, mostly comprehend proposals, fire often. These
;; appear where they first fire.
;;
;; Knowledge is also grouped according to similarity, which may mean
;; that it appears before it fires. For example, the fact
;; "f:high-level-goal-cues" maps each of a set of cues to a
;; high-level-goal. In the file it occurs where the first of these
;; cues occurs.
;;
;; Finding productions from trace information:
;; . for comprehend operators, search here for the parameter, or
;; the ":" tag from the operator trace if there is one (eg,
;; :selected-op)
;;
;; Notes on domain terms:
;; . s-model = referent, but u-model = u-model
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; -----
;;; t223 - "there's the operator" - assume she's attended to the
;;; result of t223, a run l that causes the create-referent operator
;;; to be selected.

(sp po*comprehend*init
 (state <s> ^superstate nil
        ^current-display tinit)
 -->
 (<s> ^operator <o> +)
 (<o> ^name comprehend ^om <om>
      ^goal cinit))

;; 10-8-95 - think about an operator when it's actually been selected (we
;; haven't imagined it).
;; 9-17-95 - only do this when there's no ambiguity about the selected
;; operator. [allows current-context to be selected after t621,
;; when exhaustion takes over after a series of add-property's.]

```

```

(sp po*comprehend*selected-operator
 (state <s> ^superstate nil
        ^wm.selected <op>
        ^wm (- ^imagined.operator* <op>)
        ^wm.fixated.<id> <op>
        ^wm (- ^selected <> <op2> <op2> ) ; 9-17-95 [checked 10-8-95]
        )
 -->
 (<s> ^operator <o>)
 (<o> ^name comprehend ^om <om>
      ^goal <op>
      ^trace :selected-op + &))

;; 10-8-95 - knowledge about which sps apply operators. [could also be
;; derived perceptually from the naming of the sp.]
;; 10-12-95 - [missing; caused :print-sp to fret; see today in display.soar]

(sp f:apply-sps
 :o-support
 (state <s> ^operator.goal
        { <apply-sp> <<
          apply-create-referent
          apply-add-property ; 10-12-95
          implement-exhausted
          >> }
        ^operator.om <k>)
 -->
 (<k> ^apply-sp <apply-sp> + &))

;;; -----
;;; t255 - "the for argument is the profile in the u-model of the
;;; thing you're creating the referent for"

;; 9-19-95 - recognize a u-model object, when thinking about one of
;; its features (having fixated on it, typically).
;; 2-18-96 - feature list excludes "referent", which occurs in other
;; contexts; these attributes don't.

(sp f:recognize-u-model-object
 :o-support; tests nothing off state
 (state <s> ^operator.goal { = <goal>
        << head left-edge right-edge bar-level
        word-id category annotation empty-node
        spec zero-head >> }
        ^operator.om <k>)
 -->
 (<k> ^object u-model + &
      ^u-model <goal> + &))

;; 1-12-96 - recall that the "for argument" is part of the u-model.
;; [the extra knowledge that motivates printing out this object.]

(sp f:recall-for-part-of-u-model
 :o-support ; tests nothing off state
 (state <s> ^operator.goal for

```

```

      ^operator.om <k>)
-->
(<k> ^for u-model + &))

;;; -----
;;; t292 - "so the question is how long do i want to leave it on
;;; there"

;; 10-9-95 - when thinking about one of these objects (when it's a
;; goal or a probe), recall the high-level-goal of returning a new
;; pointer. (we want a new pointer from the state to the s-model.
;; the only pointer to the s-model now is from the u-model.)
;;
;; also add to WM the association between the cue and what was
;; retrieved. fixations use this to decide when an object (the cue)
;; is important to look at.

(sp f:high-level-goal-cues
 :o-support ; tests nothing off state
 (state <s> ^operator.goal { <goal> <<
                          u-model
                          create-referent
                          proposal-build
                          exhausted
                          return-operator
                          propose-return-operator
                          rhs
                          >> }
      ^operator.om <om>)
-->
(<om> ^high-level-goal return-new-pointer + &
 ^return-new-pointer <goal> + &))

;;; -----
;;; t315 - "i could leave it on, until, exhaustion in this space?"

;; 8-17-95 - given the "plan" to postpone something, suppose
;; postponing until the end of processing in this space.
;; [{"^operator.goal exhausted" doesn't work because the "plan" isn't
;; in WM except as a goal.]}

(sp f:select-exhausted
 (state <s> ^operator.goal postpone
          ^wm.operator* exhausted
          ^operator.om <om>)
-->
(<om> ^selected exhausted + &))

;;; -----
;;; t322 - "what space am i in? {pgs^M} i'm in s-construct"

;; 10-9-95 - if we're at the point of exhaustion in a space, and don't
;; know what space, find out.

(sp po*comprehend*current-context-when-exhausted

```

```

(state <s> ^superstate nil
          ^wm
          (^selected exhausted
           - ^current-context))
-->
(<s> ^operator <o>)
(<o> ^name comprehend
   ^goal current-context
   ^select-now ok
   ^trace :sno
   ^om <om>))

;; 8-25-95 - propose finding out more about the current context, if we
;; don't know anything about it.

(sp po*comprehend*actual-context
 (state <s> ^superstate nil
          ^wm.current-context <s-construct>
          ^wm (- ^<s-construct>))
-->
(<s> ^operator <o>)
(<o> ^name comprehend
   ^goal <s-construct>
   ^trace :actual-context
   ^om <om>))

;;; -----
;;; t330 - "i'm going to run a little bit longer, letting it build
;;; the thing on the u-model on the profile in the u-model because i
;;; don't think i want it sitting around on the state because if i do
;;; that i'm going to have to keep going through this thing anyway
;;; following pointers to get to it so i think i'll, i think i want to
;;; let it build the whole thing and then ... and then take it off
;;; this and put it on the state but i'm not sure, in any event i have
;;; to run some more, so, where was i"

;; 10-11-95 - if we're thinking about returning a new pointer (more
;; generally, "new structure"), consider postponing, when we're only
;; now building the pointed-to structure. (the s-construct space
;; builds a complete s-model; creating the referent is the first
;; step.)

(sp f:postpone-return-new-pointer
 (state <s> ^operator.goal return-new-pointer
          ^wm.current-context s-construct
          ^wm.for create-referent
          ^operator.om <om>)
-->
(<om> ^postpone return-new-pointer + &))

;; 8-29-95 - if we've now decided to postpone some change, and know
;; something about what we're going to postpone, then figure out
;; where we were in the execution [and resume].
;; 9-13-95 - [this fires the sp more often, apparently usefully.]

```

```

(sp po*comprehend*where-was-i
  (state <s> ^superstate nil
    ^wm.postpone <something>
    ^wm.<something> <something-else> ; 9-13-95
  )
-->
(<s> ^operator <o>)
(<o> ^name comprehend
  ^goal where-was-i
  ^select-now ok
  ^trace :sno
  ^om <om>))

;;; -----
;;; t416 - "so it [apply-create-referent] must have just changed it
;;; on, on the superstate?"

;; 8-31-95 - if we think the target is the superstate but there's no
;; problem space test to confirm this, think about the superstate.

(sp po*comprehend*superstate-target
  (state <s> ^superstate nil
    ^wm.problem-space nil
    ^wm.target superstate
    ^wm (- ^superstate))
-->
(<s> ^operator <o>)
(<o> ^name comprehend
  ^goal superstate
  ^om <om>))

;;; -----
;;; t421 - "is that -- oh these are shared states i see"

;; 10-11-95 - recall that shared-states can result in
;; operator-application chunks ("apply-chunks" is an abstract
;; chunk), and also that shared-state is a kind of state.

(sp f:shared-states-build-chunks
  (state <s> ^operator.goal shared-state
    ^operator.om <om>)
-->
(<om> ^shared-state apply-chunks + &
  ^state shared-state + &))

;; 10-11-95 - recall, when thinking about a shared state, that it's
;; also the superstate.
;; 9-9-95 - [knowing something about the superstate inhibits
;; po*comprehend*superstate-target, above]

(sp f:superstate-is-shared-too
  (state <s> ^operator.goal shared-state
    ^wm.state shared-state
    ^operator.om <om>)
-->

```

```

(<om> ^superstate shared-state + &))

;;; -----
;;; t447 - "so i'm in the s-construct space, i'm going to slap that
;;; thing on there, and lo and behold, i get this chunk, because
;;; they share the state"

;; 9-9-95 - think about something we've seen before that causes a
;; chunk.

(sp po*comprehend*imagined-chunk-cause
  (state <s> ^superstate nil
    ^wm.<< fixated-recently imagined-but-seen >> <shared-state>
    ^wm.<shared-state> apply-chunks)
-->
(<s> ^operator <o>)
(<o> ^name comprehend
  ^goal <shared-state>
  ^trace :imag-chunk-cause
  ^om <om>))

;; 10-11-95 - think about a real chunk (vs an abstract one).
;; 9-13-95 - [have to care about the build set, which we don't when
;; chunk-130/1 appear. this is a token representation of "let's
;; look at the chunks, shall we"?]

(sp po*comprehend*chunk
  (state <s> ^superstate nil
    ^wm.chunk <chunk-128>
    ^wm.builds <builds> ; 9-13-95
    ^wm (- ^abstract-chunk <chunk-128>))
-->
(<s> ^operator <o>)
(<o> ^name comprehend
  ^goal <chunk-128>
  ^trace :chunk
  ^om <om>))

;; 10-10-95 - identify sps as sps. the list includes all 10 sps and 4
;; chunks that appear in the protocol (see also f:abstract-sp). "+"
;; points to other knowledge where where an sp is identified as such.

(sp f:sp
  :o-support
  (state <s> ^operator.goal
    { <sp> <<
      apply-create-referent
      implement-exhausted
      apply-add-property
      terminate-add-property
      chunk-128
      chunk-129
      touch-conjunct-symbol
      terminate-create-referent ; +f:terminate-create-referent
      head-noun-cop

```

```

        head-noun-cop-animate
        chunk-130
        chunk-131
        propose-return-operator ; + f:propose-return-operator
        terminate-s-model-constructor
        >> }
    ^operator.om <om>)
-->
(<om> ^sp <sp> + &))          ; sps have parts

;; 1-30-96 - abstract sps [except nil] are also sps [only one left].

(sp f:abstract-sp
 :o-support
 (state <s> ^operator.goal
  { <abs-sp> <<
    apply-chunks
    >> }
  ^operator.om <om>)
-->
(<om> ^sp <abs-sp> + &))

;;; -----
;;; t463 - "i don't know why it's testing for the ... operator?"

;; 10-11-95 - think about an sp matching and firing. [this fires
;; early on, but it and related sps below were conceived here when
;; this was the beginning of the model.]
;; 9-24-95 - except when we're being careful.

(sp po*comprehend*sp-parts
 (state <s> ^superstate nil
  ^wm.part { << lhs rhs >> <part> }
  ^wm (- ^high-level-goal pay-attention))
-->
(<s> ^operator <o>)
(<o> ^name comprehend
 ^goal <part>
 ^trace :sp-parts
 ^om <om>))

;;; -----
;;; t469 - "why is it testing for the operator?"

;; 2-26-95 - if we know that there's a condition that's an operator,
;; but we don't know anything about any operator*, seek to understand.

(sp po*comprehend*operator-condition
 (state <s> ^superstate nil
  ^wm.condition operator*)
-->
(<s> ^operator <o>)
(<o> ^name comprehend
 ^goal operator*
 ^trace :op-cond

```

```

    ^om <om>))
;; 3-27-95 - an operator condition is part of a lhs. [lesser-known
;; conditions don't generate this reminder.]

(sp f:operator-condition*part-of-lhs
 (state <s> ^operator.goal operator*
  ^wm.condition operator*
  ^operator.om <om>)
-->
(<om> ^part lhs + &))

;; 3-27-95 - from lhs we know that there's a rhs.

(sp f:operator-condition*lhs-means-rhs
 :o-support
 (state <s> ^operator.goal lhs
  ^operator.om <om>)
-->
(<om> ^part rhs + &))

;;; -----
;;; t492 - "alright rick which means i now have, i now have it sitting
;;; there ... and the second chunk i know is just going to be testing
;;; for ... um the bead right yeah there's the conjunct symbol fine"

;; 9-17-95 - care about what chunks were built as a consequence, but
;; only when we've recently heeded what sps were about to fire.
;; [heads off comprehending chunk-130-1.]

(sp po*comprehend*builds
 (state <s> ^superstate nil
  ^wm.builds <builds-initial>
  ^wm.match-set <ms>) ; 9-17-95
-->
(<s> ^operator <o>)
(<o> ^name comprehend
 ^goal <builds-initial>
 ^trace :builds
 ^om <om>))

;;; -----
;;; t513 - "ok, terminate-create-referent, fine"

;; 10-10-95 - when thinking about the referent, and we've done the
;; bead stuff, the referent is created, so "expect" that the
;; terminate sp matches.

(sp f:terminate-create-referent
 (state <s> ^operator.goal referent
  ^wm.conjunct-symbol conjunct
  ^operator.om <om>)
-->
(<om> ^sp terminate-create-referent + &
 ^terminate-create-referent lhs + &))

```

```

;;; -----
;;; t538 - "what's going to happen when it does this it's going to
;;; build some more chunks, should build two more chunks, which means
;;; that in both spaces it will have, added the properties to"

;; 9-11-95 - recall that these sps cause the add-property operator to
;; be selected. [in f:select-exhausted, exhausted is already an
;; ^operator*.]

(sp f:sps-propose-add-property
 :o-support
 (state <s> ^operator.goal { << head-noun-cop-animate
                          head-noun-cop >> <sp> }
          ^operator.om <om>)
 -->
 (<om> ^operator* add-property + & ; [gives us a probe]
  ^selected add-property + &))

;; 10-10-95 - when we're thinking about add-property, and we have the
;; referent (which receives the property), recall that
;; add-property changes the superstate (the referent is on the
;; state, which is also the superstate).

(sp f:add-property-target
 (state <s> ^operator.goal add-property
          ^wm.action referent
          ^operator.om <om>)
 -->
 (<om> ^add-property target + &
  ^target superstate + &))

;; 9-13-95 - when we think something modifies the superstate, and we
;; see a modifying agent, think about that agent [to verify the
;; connection?]

(sp po*comprehend*building-agent
 (state <s> ^superstate nil
          ^wm.target superstate
          ^wm.<add-property> target
          ^wm (- ^imagined.<att> <add-property>) ; [10-11-95 - need this]
          )
 -->
 (<s> ^operator <o>)
 (<o> ^name comprehend
  ^goal <add-property>
  ^trace :building-agent
  ^om <om> ))

;;; -----
;;; t554 - "hm! why didn't it build a chunk"

;; 9-13-95 - recall that an absent chunk is an abstraction.
;; [means we won't try to display it. we "see" the absence of the
;; chunk, in a fixation.]

```

```

(sp f:nil-chunk-is-abstract
 (state <s> ^operator.goal nil
          ^wm.chunk nil
          ^operator.om <om>)
 -->
 (<om> ^abstract-chunk nil))

;;; -----
;;; t580 - "add the properties so i should get two more chunks, yes"

;; 9-13-95 - recall that the apply-add-property sp modifies the
;; superstate [cf f:add-property-target].

(sp f:apply-add-property-target
 :o-support
 (state <s> ^operator.goal apply-add-property
          ^operator.om <om>)
 -->
 (<om> ^apply-add-property target + &
  ^target superstate + &))

;;; -----
;;; t585 - "and then i should get reconsiders, yup"

;; 10-10-95 - recall that a termination sp issues a reconsider
;; preference. these are all the termination sps that appears in
;; the protocol.

(sp f:terminating-sps
 :o-support
 (state <s> ^operator.goal
          { <term-sp> <<
            terminate-add-property ; 10-11-95 - began with this
            terminate-s-model-creator ; 10-12-95 - [missing]
            terminate-create-referent
            >> }
          ^operator.om <om>)
 -->
 (<om> ^<term-sp> reconsider + &))

;;; -----
;;; t638 - "ok were am i, s15 now has an utterance model object"

;; 10-10-95 - when at exhaustion in the s-construct space, recall the
;; u-model object.
;;
;; . the display contains no more obvious cue to remind us of the
;; u-model
;; . this might be episodic knowledge encoded earlier.

(sp f:recall-u-model
 (state <s> ^operator.goal s-construct
          ^wm.selected exhausted
          ^operator.om <om>)
 -->

```

```

(<om> ^object u-model + &))

;; 10-10-95 - when we know that the state has a u-model but we don't
;; know anything about the model, think about the u-model.

(sp po*comprehend*u-model
 (state <s> ^superstate nil
  ^wm.state <s15> ; 9-21-95
  ^wm.object u-model
  ^wm (- ^u-model))
 -->
 (<s> ^operator <o>)
 (<o> ^name comprehend
  ^goal u-model
  ^om <om>))

;; 10-10-95 - recall that the u-model is associated with a referent.

(sp f:u-model
 (state <s> ^operator.goal u-model
  ^operator.om <om>)
 -->
 (<om> ^referent t + &))

;;; -----
;;; t662 - "we're about to come out of the s-constructor space to
;;; build a proposal"

;; 9-21-95 - recall that exhausted returns a new pointer. [expanded
;; version: exhausted marks the beginning of returning from the
;; subgoal, which is related to our high-level-goal for this goal.
;; [don't have a principle reason to distinguish between
;; ^<object> return-new-pointer" and ^high-level-goal
;; return-new-pointer ^return-new-pointer <object>"]

(sp f:exhausted-builds-proposal
 (state <s> ^operator.goal exhausted
  ^wm.selected exhausted
  ^operator.om <om>)
 -->
 (<om> ^exhausted return-new-pointer))

;;; -----
;;; t673 - "here's the big question, do i want to let it return this
;;; thing"

;; 3-6-95 - when a high-level-goal enters wm, think about it.

(sp po*comprehend*high-level-goal
 (state <s> ^superstate nil
  ^wm.high-level-goal <goal>)
 -->
 (<s> ^operator <o>)
 (<o> ^name comprehend
  ^goal <goal>

```

```

  ^trace :high-lev-goal
  ^om <om>))

;; 10-10-95 - recall that these are operators. these are all the
;; operators in the protocol.

(sp f:operator
 :o-support
 (state <s> ^operator.goal
  { <op> <<
   create-referent
   add-property ; + f:sps-propose-add-property
   exhausted
   return-operator
   s-constructor16
   >> }
  ^operator.om <om>)
 -->
 (<om> ^operator* <op> + &))

;;; -----
;;; t686 - "implement exhausted what does that do"

;; 9-1-95 - comprehend an assertion either if we saw it or if we
;; imagined it and recognize the image. [this also began here but
;; fires much earlier.]
;; 9-24-95 - defer to other methods when moving slowly.

(sp po*comprehend*assertion*real
 (state <s> ^superstate nil
  ^wm.assertion <sp>
  - ^wm.imagined.assertion <sp>
  ^wm (- ^high-level-goal pay-attention) ; 9-24-95
  )
 -->
 (<s> ^operator <o>)
 (<o> ^name comprehend
  ^goal <sp>
  ^trace :assertion
  ^om <om>))

(sp po*comprehend*assertion*imagined
 (state <s> ^superstate nil
  ^wm.assertion <sp>
  ^wm.imagined-but-seen <sp>)
 -->
 (<s> ^operator <o>)
 (<o> ^name comprehend
  ^goal <sp>
  ^trace :imagined-assertion
  ^om <om>))

;; 11-29-94 - sps have conditions and actions. [cf
;; po*comprehend*sp-parts]

```

```

(sp f:sp-parts
  (state <s> ^operator.goal <sp>
    ^wm.sp <sp>
    ^operator.om <k>)
  -->
  (<k> ^part lhs + &, rhs + &))

;;; -----
;;; t696 - "ok it puts this marvelous construction done flag, on, the
;;; superstate"

;; 10-11-95 - when an sp action is the superstate, think about the
;; superstate. [this is a specific sp action that we fixated on,
;; vs. the more abstract "target".]

(sp po*comprehend*superstate
  (state <s> ^superstate nil
    ^wm.action superstate)
  -->
  (<s> ^operator <o>)
  (<o> ^name comprehend ^goal superstate ^om <om>
    ^trace :superstate + &))

;;; -----
;;; t726 - "s-construction done, u-constructor applied, u-model
;;; success, this is going to let me propose the return operator"

;; 9-25-95 - when we've seen the construction-done annotation, while
;; thinking about a supercontext, know that we're about to propose
;; the return operator (which does things to the supercontext). [cf
;; f:terminate-create-referent.]

(sp f:propose-return-operator
  (state <s> ^operator.goal superstate
    ^wm.annotation construction-done
    ^operator.om <k>)
  -->
  (<k> ^sp propose-return-operator + &
    ^propose-return-operator lhs + &))

;; 9-24-95 - know that the return operator returns new structure and
;; that it modifies the superstate (more generally, the supercontext).

(sp f:return-operator-target
  :o-support
  (state <s> ^operator.goal return-operator
    ^operator.om <om>)
  -->
  (<om> ^return-operator return-new-pointer
    ^target superstate + &))

;;; -----
;;; t734 - "where ... create, in the create operator space."

;; 2-20-95 - when propose-return-operator is asserted, find out the

```

```

;; proposal context, if we don't know it.
;; 10-11-95 - and when we're not in careful mode.

(sp po*comprehend*proposal-context
  (state <s> ^superstate nil
    ^operator.goal propose-return-operator
    ^wm (- ^proposal-context)
    ^wm (- ^high-level-goal pay-attention) ; 10-11-95
  )
  -->
  (<s> ^operator <o>)
  (<o> ^name comprehend
    ^goal proposal-context
    ^select-now ok
    ^trace :sno
    ^om <om>))

;;; -----
;;; t750 - "propose return operator, i think we're getting close to
;;; the right place"

;; 10-11-95 - recall that s-construct builds propose-operator
;; chunks, as part of proposing an operator in another context [an
;; "abstract" action]

(sp f:proposal-build-action
  (state <s> ^operator.goal s-construct
    ^wm.proposal-context create-operator
    ^operator.om <om>)
  -->
  (<om> ^action proposal-build + &))

;; 9-24-95 - when the goal is to return a new pointer, and we're
;; about to build stuff, make the high-level-goal be to pay
;; attention. [this "careful mode" inhibits (some) other methods.]

(sp f:pay-attention-when-building-proposal
  (state <s> ^operator.goal return-new-pointer
    ^wm.action proposal-build
    ^operator.om <om>)
  -->
  (<om> ^high-level-goal pay-attention + &
    ^pay-attention proposal-build + &))

;;; -----
;;; t753 - "terminate s-model constructor, see what that's doing"

;; 9-24-95 - when we're in careful mode, think about new assertions.
;; [pay-attention inhibits po*comprehend*assertion*real, so we think
;; about assertions breadth-first. this may be appropriate: if in the
;; course of thinking about one the other scrolls off, want to
;; increase the chance that we can remember the other. see also
;; below.]

(sp po*comprehend*assertion*pay-attention

```

```

(state <s> ^superstate nil
  ^wm.high-level-goal pay-attention
  ^wm.assertion <sp>
  ;; 10-12-95 - actually saw <sp>, and for the first time:
  ^wm.fixated (^assertion <sp> ^time <now> - ^time <> <now>))
-->
(<s> ^operator <o>)
(<o> ^name comprehend
  ^goal <sp>
  ^trace :assertion-pay-attn
  ^om <om>))

;;; -----
;;; t773 - "now what is this [propose-return-operator] doing"

;; 9-24-95 - if we're paying attention, and an assertion just
;; disappeared, go find it and think about it.

(sp po*comprehend*assertion*pay-attention*fixated-recently
 (state <s> ^superstate nil
  ^wm.high-level-goal pay-attention
  ^wm.assertion <propose-return-operator>
  ^wm.fixated-recently <propose-return-operator>
  )
-->
(<s> ^operator <o>)
(<o> ^name comprehend
  ^goal <propose-return-operator>
  ^trace :assertion-fix-recent
  ^om <om>))

;;; -----
;;; t810 - "so let's do that"

;; 3-21-95 - when propose-return-operator is asserted, and when we
;; know the return-operator's argument, think about the
;; return-operator. [elsewhere this kind of expectation is
;; represented as a fact.]

;; 12-23-95 - "^new-operator" binds the operator we're going to
;; return; find out what that is.

(sp po*comprehend*new-operator
 (state <s> ^superstate nil
  ^wm.new-operator <s-constructor16>
  ^wm.operator* <s-constructor16>)
-->
(<s> ^operator <o>)
(<o> ^name comprehend
  ^goal <s-constructor16>
  ^trace :new-operator
  ^om <om>))

;;; -----
;;; t821 - "ok, it says"

```

```

;; 10-12-95 - when we know an id, think about the attribute of which
;; it's a value, but only if we already know something about this
;; attribute. [notice what's a little familiar; this seems
;; appropriately opposite of the breadth-first approach to
;; comprehending assertions, when in pay-attention mode.]

(sp po*comprehend*objects-attribute
 (state <s> ^superstate nil
  ^wm.id <id> ; u20, r9
  ^wm.<att> <id> { <> <id> <other-k> }
  ^wm.attribute <att>)
-->
(<s> ^operator <o>)
(<o> ^name comprehend
  ^goal <att>
  ^trace :objects-att
  ^om <om>))

;;; -----
;;; t843 - "i think all those chunks i built {^[v] test for operator
;;; six {^[v] uh s whatever-it-is, i think they test for the
;;; s-constructer {^[v]"

;; 1-30-96 - the operator "s-model-constructer" is implemented by
;; chunks. "s-construct" is the problem space in which the operator
;; is applied.

(sp f:s-construct-builds-chunk
 (state <s> ^operator.goal <<
  s-model-constructer ; 1-13-96
  s-construct
  >>
  ^operator.om <k>)
-->
(<k> ^chunk apply-chunks + &
  ^abstract-chunk apply-chunks + &))

;; 10-12-95 - if we've recalled a condition (imagined it and recognized
;; the image), think about it.

(sp po*comprehend*recalled-condition
 (state <s> ^superstate nil
  ^wm.imagined-but-seen <s-model-constructer>
  ^wm.condition <s-model-constructer>)
-->
(<s> ^operator <o>)
(<o> ^name comprehend
  ^goal <s-model-constructer>
  ^om <om>
  ^trace :recalled-condition))

```

C.3.2. display.soar

ao*display*po*init.....	137	ao*display*print*t564.....	145
a*display*tinit.....	138	d*print*t564.....	145
ao*display*tinit.....	138	a*display*t575.....	146
d*print*tinit.....	138	ao*display*print*t575.....	146
po*display*match-set*after-selection.....	138	d*print*t575.....	146
a*display*t225.....	138	ao*display*t582.....	146
ao*display*t225.....	138	a*display*t582.....	146
d*tinit*t225.....	138	d*initial*t582.....	146
po*display*print-sp*applies-operator.....	139	a*display*t587.....	147
a*display*t236.....	139	ao*display*print*t587.....	147
ao*display*print*t236.....	139	d*print*t587.....	147
d*print*t236.....	139	po*display*run*to-op-after-builds.....	147
po*display*print-operator.....	139	a*display*t593.....	147
a*display*t245.....	139	ao*display*t593.....	147
ao*display*print*t245.....	140	d*run-ms-run*t593.....	147
d*print-id*t245.....	140	a*display*t618.....	148
po*display*print-object.....	140	ao*display*t618.....	148
a*display*t252.....	140	d*initial*t618.....	148
ao*display*print*t252.....	140	po*display*run*to-end-of-space.....	148
d*print-id*t252.....	140	a*display*t621.....	148
po*display*print-stack.....	141	ao*display*t621.....	148
a*display*t323.....	141	d*run-ms-run*t621.....	149
ao*display*print*t323.....	141	a*display*t639.....	149
d*print*t323.....	141	ao*display*print*t639.....	149
po*display*run*where-was-i.....	141	d*pgs*t639.....	149
a*display*t364.....	141	po*display*scroll*to-object.....	149
ao*display*print*t364.....	142	a*display*t646.....	150
d*print*t364.....	142	ao*display*print*t646.....	150
po*display*print-chunk.....	142	po*display*print-object*fresh.....	150
a*display*t373.....	142	a*display*t649.....	150
ao*display*print*t373.....	142	ao*display*print*t649.....	150
d*print*t373.....	142	d*print*t649.....	150
po*display*scroll*to-sp.....	143	a*display*t654.....	150
a*display*t431.....	143	ao*display*print*t654.....	150
ao*display*print*t431.....	143	d*print*t654.....	151
po*display*scroll*to-state.....	143	a*display*t685.....	151
a*display*t445.....	143	ao*display*print*t685.....	151
ao*display*t445.....	143	d*print*t685.....	151
a*display*t503.....	143	a*display*t691.....	151
ao*display*print*t503.....	143	ao*display*print*t691.....	151
d*print*t503.....	144	d*print*t691.....	151
po*display*match-set*asserted-sp.....	144	po*display*run*action-and-print.....	152
a*display*t513.....	144	a*display*t720.....	152
ao*display*print*t513.....	144	ao*display*print*t720.....	152
d*print*t513.....	144	d*print*t720.....	152
po*display*run-to-builds.....	145	a*display*t730.....	152
a*display*t552.....	145	ao*display*print*t730.....	152
ao*display*print*t552.....	145	d*print*t730.....	153
d*print*t552.....	145	a*display*t738.....	153
po*display*match-set*after-builds.....	145	ao*display*print*t738.....	153
a*display*t564.....	145	d*print*t738.....	153
		po*display*print-sp*when-paying-attention.....	153
		a*display*t754.....	154
		ao*display*print*t754.....	154
		d*print*t754.....	154

```

a*display*t770..... 154
ao*display*print*t770..... 154
po*display*print*high-level-goal..... 154
a*display*t774..... 154
ao*display*print*t774..... 154
d*print*t774..... 155
po*display*run*to-expected-op..... 155
a*display*t811..... 155
ao*display*print*t811..... 155
d*print*t811..... 155
a*display*t817..... 155
ao*display*print*t817..... 155
d*print*t817..... 156
a*display*t845..... 156
ao*display*print*t845..... 156

;; -*- Mode: Sde -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; File      : display.soar
;; Author    : Erik Altmann
;; Created On : Tue Jun 21 11:28:01 1994
;; Last Modified By: Erik Altmann <altmann@electro.soar.cs.cmu.edu>
;; Last Modified On: 30 Jun 1996, 20:29:35
;; Update Count : 2110
;;
;; This file contains both the proposals for issuing commands to
;; change the display (generate new output or scroll) and most of
;; the display emulator that interprets commands and modifies the
;; emulated display accordingly.
;;
;; The file is organized into rule groups, each group corresponding
;; to a unit command or semantic cluster issued by the programmer.
;; The names of command-proposal rules are prefixed "po*" (propose
;; operator). Emulator rule names are prefixed "a*" (augment),
;; "ao*" (apply-operator), and "d*" (display).
;;
;; The first of the three emulator productions (prefixed "a*")
;; augments a proposed display operator if the operator is a hit
;; (and not a false alarm). Two elements of context information
;; usually suffice to decide this: (1) the timestamp previous
;; command, (2) the comprehension goal in service of which the
;; intention was proposed. To mark the operator a hit , the
;; emulator uses the timestamp of the current command. (Thus an
;; un-timed "display" operator occurring in the trace is an false
;; alarm.)
;;
;; The second of the three emulator productions (prefixed "ao*")
;; helps apply a display operator. (The rest of the application
;; occurs in the mechanism.soar.) It models the buffer-management
;; actions taken by the editor to handle the output of the command.
;; Where a new region appeared at the bottom of the buffer in the
;; protocol, the buffer-management sp adds the timestamp of that
;; region to the set on display. Where old regions scrolled off the

```

```

;; top in the protocol, the buffer-management sp removes the
;; timestamps for those regions.
;;
;; (The first and second sps do different things conceptually, but
;; there is another reason that they are separate. In order for the
;; timestamp to appear in the trace, it has to appear on the
;; operator before the operator is selected. On the other hand,
;; manipulating the set of on-display timestamps takes o-support.)
;;
;; The last sp in each group ("d*") generates the contents of the
;; output region for the command. It fires when the timestamp for
;; the command appears in the on-display set. It retracts when its
;; timestamp is removed from the on-display set, and the content of
;; the region disappears with it (the production is i-supported).
;;
;; The proposals ("po*") and last emulator rule ("d*") are optional,
;; because they transfer. The proposal appears where it first
;; fires. There's one "d*" for every command that generates new
;; output at the bottom of the buffer. For scroll commands, the
;; second emulator rule ("ao*") adds the timestamps of the
;; scrolled-to screen and removes those of the scrolled-from screen,
;; and the "d*" rules respond automatically.
;;
;; Other tricks and techniques are documented where they first
;; appear.
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; -----
;;; initial-screen - "you've got something out there that's a noun but
;;; doesn't have a referent, create the operator, so that's cool
;;; {run_1^Mms^M [t218, produces empty match set]} now what's it
;;; going to do, ok so that's the end of that decision cycle there's
;;; {run_1^M [t223, produces a selected operator]} the operator"
;;;
;;; 10-4-95 - generates that part of the screen above t225 which is
;;; still visible after t236. t225 itself is separate.
;;; 10-4-95 - throughout, represent only regions that are completely on
;;; the screen. so t197 is left off [removing it didn't hurt.]
;;; 6-6-96 - the awful name is to make this filter out with emulator
;;; mechanism SPs.

(sp ao*display*po*init
  (state <g> ^superstate nil ; all intentions in top context
    - ^current-display)
  -->
  (<g> ^operator <o>))
  (<o> ^name display
    ^goal tinit ; command's timestamp in protocol
    ^val init ; usually the current comprehension goal
    ^terminate-and-reject t)) ; meta-stuff

;; the first emulator production marks a proposed intention, if the
;; context is right for that intention.

```

```

(sp a*display*tinit
  (state <s> ^superstate nil
    ^operator <o> +)
  (<o> ^name display ^val init)
  -->
  (<o> ^time tinit))

;; the second emulator production adds the timestamp to the set on
;; display, once the intention is selected.

(sp ao*display*tinit
  :o-support
  (state <s> ^superstate nil
    ^operator.name display
    ^operator.goal tinit)
  -->
  (<s> ^on-display tinit + &))

(sp d*print*tinit
  (state <s> ^superstate nil
    ^on-display tinit
    ^display <d>)
  -->
  (<d> ^tinit <r1> + &, <r2> + &))

(<r1> ^previous no-display          ; t218 (run 1, ms)
  ^match-set ms-t218
  ^assertions nil
  ^retractions-name retractions-t218
  ^retractions nil)
(<r2> ^previous no-display          ; t223 (run 1)
  ^decision-cycle 76
  ^operator-id o25
  ^selected create-referent
  ^argument cop))

;; 1044 Soar> run 1
;; 1045
;; 1046 Soar> ms
;; 1047 Assertions:
;; 1048 Retractions:
;; 1049
;; 1050 Soar> run 1                [^:218]
;; 1051 76:                        O: 025 create-referent(cop)
;; 1052 Soar>                      [^:223]

;;; -----
;;; t225 - "there's {t223} the operator {ms^M} ok create referent,
;;; touch conjunct symbol, that's fine"

;; 7-9-95 - when trying to comprehend a recently-selected operator,
;; find out what sps will apply it. [get recency from the selection
;; being in the newest region.]

```

```

(sp po*display*match-set*after-selection
  (state <g> ^superstate nil
    ^operator.goal <create-referent>
    ^wm.selected <create-referent>
    ^wm.fixated (^selected <create-referent>
      - ^imagined-at)
    )
  -->
  (<g> ^operator <o>)
  (<o> ^name display
    ^goal match-set:after-selection
    ^val <create-referent>
    ^terminate-and-reject t))

;; 7-19-95 - put the name of the display region on the display
;; operator when the goal ("^val" on the display operator) in its
;; temporal context corresponds to the programmer's behavior. this
;; allows for false positives in the display intentions.

(sp a*display*t225
  (state <s> ^superstate nil
    ^current-display tinit
    ^operator <o> +)
  (<o> ^name display ^val create-referent)
  -->
  (<o> ^time t225))

(sp ao*display*t225
  :o-support
  (state <s> ^superstate nil
    ^operator.name display
    ^current-display t225)
  -->
  (<s> ^on-display t225 + &))

(sp d*tinit*t225
  (state <s> ^superstate nil
    ^on-display t225
    ^display <d>)
  -->
  (<d> ^t225 <r1> + &))
(<r1> ^previous tinit
  ^spatial <spatial>
  ^match-set ms-t225
  ^assertions apply-create-referent + &,
  touch-conjunct-symbol + &
  ^retractions-name retractions-t225
  ^retractions nil)
(<spatial> ^apply-create-referent touch-conjunct-symbol))

;; 1052 Soar> ms
;; 1053 Assertions:
;; 1054 s-construct*create-referent*touch-conjunct-symbol
;; 1055 s-construct*create-referent
;; 1056 Retractions:

```

```

;; 1057
;; 1058 Soar>

;;; -----
;;; t236 "that's fine, that's just going to string beads, and this is
;;; the question, where is it sticking this
;;; [p__s-construct*create-referent^M] so it's sticking ..."
;;;
;;; ["this" = the referent in the actions of the printed sp]

;; 1-28-96 - print an apply sp.  have to know of an operator (being
;; applied), and also have to have actually seen the sp asserted.

(sp po*display*print-sp*applies-operator
  (state <g> ^superstate nil
    ^operator.goal <apply-create-referent>
    ^wm.apply-sp <apply-create-referent>
    ^wm.operator* <op>
  )
  -->
  (<g> ^operator <o>)
  (<o> ^name display
    ^goal print-sp:applies-operator
    ^val <apply-create-referent>
    ^trace <apply-create-referent>
    ^terminate-and-reject t))

(sp a*display*t236
  (state <s> ^superstate nil
    ^current-display t225      ; previous command, & part of context
    ^operator <o> +)
  (<o> ^name display ^val apply-create-referent)
  -->
  (<o> ^time t236))

(sp ao*display*print*t236
  :o-support
  (state <s> ^superstate nil
    ^current-display t236)
  -->
  (<s> ^on-display t236 + &))

;; 7-31-95 - made the conditions and actions point to the conditions
;; and actions they point to on the display, as context info.  also
;; added the ^attribute and ^constant meta-attributes.  see
;; fixate*binding-context*condition.

(sp d*print*t236
  (state <s> ^superstate nil
    ^on-display t236
    ^display <d>)
  -->
  (<d> ^t236 <sp> + &)
  (<sp> ^previous t225
    ^spatial <spatial>

```

```

^sp apply-create-referent
^condition <cg> + &, <co2> + &, <cp2> + &, <cf> + &
^action <a1> + &, <a2> + &
; ; 7-10-95 - meta-attributes, like ^id for objects:
^leaf <co2> + &, <cp2> + &
)
(<spatial> ^<cg> <co2> ^<co2> <cp2> ^<cp2> <a1> ^<a1> <a2>)
(<cg> ^goal g ^operator* <co2> ^problem-space <cp2> ^state s)
(<co2> ^operator* create-referent ^for obj)
(<cp2> ^problem-space s-construct)

(<a1> ^obj referent ^referent <a2>)
(<a2> ^referent-of obj ^type s-model))

; ; 1058 Soar> p s-construct*create-referent
; ; 1059 (sp s-construct*create-referent
; ; 1060 (goal <g> ^operator <o> ^problem-space <p> ^state <s>)
; ; 1061 (<o> ^name create-referent ^for <obj>))
; ; 1062 (<p> ^name s-construct)
; ; 1063 -->
; ; 1064 (<obj> ^referent <r> + ^referent <r> &)
; ; 1065 (<r> ^referent-of <obj> + ^type s-model +))

;;; -----
;;; t245 - "so it's sticking ... let's see, it's uh, p o 25 {p__o25^M}
;;; ok the operator has got"

;; 1-29-96 - print out an operator, when (a) it provides the value for
;; a variable binding, and (b) it's fresh (multiple operators can on
;; display at once).

(sp po*display*print-operator
  (state <g> ^superstate nil
    ^operator.goal <op-related>
    ^wm.<< condition variable >> <op-related>
    ^wm.<< bound-by binding-context >> operator* ; (a)
    ^wm.operator-id <o25>
    ^wm.dp.newest <newest> ; (b)
    ^wm.fixated (^ << operator* operator-id >> ; 1-27-96
      ^region <newest>)
  )
  -->
  (<g> ^operator <o>)
  (<o> ^name display
    ^goal print-op
    ^val <op-related>
    ^terminate-and-reject t))

(sp a*display*t245
  (state <s> ^superstate nil
    ^current-display t236
    ^operator <o> +)
  (<o> ^name display ^val operator*)
  -->
  (<o> ^time t245))

```

```

(sp ao*display*print*t245
 :o-support
 (state <s> ^superstate nil
      ^current-display t245)
 -->
 (<s> ^on-display t245 + &))

;; 3-24-95 - meta-attributes describe the actual object attributes;
;; they provide information we would derive perceptually from, say,
;; the structure of an identifier. they can't be fixated on.

(sp d*print-id*t245
 (state <s> ^superstate nil
      ^on-display t245
      ^display <d>)
 -->
 (<d> ^t245 <o> + &)
 (<o> ^previous t236
      ^meta-attribute object-id + &, id + &, ; 3-24-95
      attribute + &, value + &
      ^object-id o25
      ^id o25 + &, u20 + &
      ^attribute for ^value u20
      ;; 10-3-95 - the augmentations as they actually appear:
      ^name* create-referent ^for u20))

;; 1068 Soar> p o25
;; 1069 (O25 ^name create-referent ^for U20)
;; 1070
;; 1071 Soar>

;;; -----
;;; t252 - "ok the operator has got, what is u 20 {p u20} the operator
;;; has this for argument"

;; 1-29-96 - print the value of (1) an attribute, if (2) we have the
;; value's id.
;; 8-15-95 - print an object, if the attribute it belongs
;; know more than one thing about that attribute (recalling
;; something about it means we care about it).
;; 9-21-95 - multi-attributes (eg, ^properties) don't count.
;; 1-13-96 - print operators with print-op.
;; 1-27-96 - the attribute must be fresh [prevents this from
;; overlapping with print-object-fresh].

(sp po*display*print-object
 (state <g> ^superstate nil
      ^operator.goal <for> ; 1
      ^wm.<for> <u20> { <> <u20> <other-k> } ; 8-15-95
      ^wm.object-id <u20> ; 2
      ^wm (- ^id <other-k>) ; 9-21-95
      - ^wm (^<for> <var> ^<var> operator*) ; 1-13-96
      ^wm.dp.newest <newest> ; 1-27-96
      ^wm.fixated (^object-id <u20> ^region <newest>)
 )

```

```

-->
 (<g> ^operator <o>)
 (<o> ^name display
      ^goal print-object
      ^val <for>
      ^trace <u20>
      ^terminate-and-reject t))

(sp a*display*t252
 (state <s> ^superstate nil
      ^current-display t245
      ^operator <o> +)
 (<o> ^name display ^val for)
 -->
 (<o> ^time t252))

(sp ao*display*print*t252
 :o-support
 (state <s> ^superstate nil
      ^current-display t252)
 -->
 (<s> ^on-display t252 + &))

(sp d*print-id*t252
 (state <s> ^superstate nil
      ^on-display t252
      ^display <d>)
 -->
 (<d> ^t252 <o> + &)
 (<o> ^previous t245
      ^meta-attribute meta-attribute + &, object-id + &, id + &,
      attribute + &, value + &
      ^object-id u20
      ^id u20 + &, w8 + &, w13 + &, e15 + &, u14 + &, u15 + &, u17 + &
      ^attribute left-edge + &, right-edge + &, bar-level + &,
      word-id + &, category + &, annotation + &,
      empty-node + &, spec + &, zero-head + &, head + &
      ^value u20 + &, w8 + &, w13 + &, e15 + &, u14 + &, u15 + &, u17 + &,
      max + &, n + &, specified + &
      ;; the actual object:
      ^left-edge W8 ^right-edge W13 ^bar-level max ^word-id W13 ^category n
      ^annotation specified ^empty-node E15 ^spec U14 ^zero-head U15
      ^head U17))

;; 1071 Soar> p u20
;; 1072 (U20 ^left-edge W8 ^right-edge W13 ^bar-level max ^word-id W13 ^category n
;; 1073 ^annotation specified ^empty-node E15 ^spec U14 ^zero-head U15
;; 1074 ^head U17)
;; 1075
;; 1076 Soar>

;;; -----
;;; t323 - "so maybe i'll leave it on, i could leave it on until,
;;; exhaustion in this space? what {pgs^M} space am i in?"

```

```
;; 10-3-95 - when the goal is to understand the current context or
;; the proposal context (of an operator being returned up the
;; stack), print the context stack.
```

```
(sp po*display*print-stack
 (state <g> ^superstate nil
  ^operator.goal { << current-context proposal-context >>
    <goal> }
  )
-->
```

```
(<g> ^operator <o>)
<o> ^name display
  ^goal print-stack
  ^val <goal>
  ^terminate-and-reject t))
```

```
(sp a*display*t323
 (state <s> ^superstate nil
  ^current-display t252
  ^operator <o> +)
<o> ^name display ^val current-context)
-->
<o> ^time t323))
```

```
(sp ao*display*print*t323
 :o-support
 (state <s> ^superstate nil
  ^current-display t323)
-->
<s> ^on-display t323 + &))
```

```
(sp d*print*t323
 (state <s> ^superstate nil
  ^on-display t323
  ^display <d>)
-->
<d> ^t323 <stack> + &)
<stack> ^context <c1> + &, <c2> + &, <c3> + &, <c4> + &
  ^previous t252)
<c1> ^operator* comprehend-input ^op-id o2 ^state s1
  ^impasse* nil ; 12-16-95 - consistent
  ^problem-space top-ps
  ^older none ^newer <c2>)
<c2> ^operator* none ^state s4 ^problem-space comprehension
  ^impasse* operator-no-change
  ^older <c1> ^newer <c3>)
<c3> ^operator* s-constructor16 ^op-id o24 ^state s15
  ^problem-space create-operator
  ^impasse* state-no-change
  ^older <c2> ^newer <c4>)
<c4> ^operator* create-referent ^op-id o25 ^state s15
  ^problem-space s-construct
  ^impasse* operator-no-change
  ^older <c3> ^newer none))
```

```
;; 1076 Soar> pgs
;; 1077 : ==>G: G1
;; 1078 : P: P1 top-ps
;; 1079 : S: S1
;; 1080 : O: O2 comprehend-input
;; 1081 : ==>G: G2 operator no-change
;; 1082 : P: P2 comprehension
;; 1083 : S: S4
;; 1084 : ==>G: G15 state no-change
;; 1085 : P: P68 create-operator
;; 1086 : S: S15
;; 1087 : O: O24 s-constructor16
;; 1088 : ==>G: G16 operator no-change
;; 1089 : P: P85 s-construct
;; 1090 : S: S15
;; 1091 : O: O25 create-referent(cop)
;; 1092
;; 1093 Soar>
```

```
;;; -----
;;; t364 - "and then take it off this and put it on the state but i'm
;;; not sure in any event i have to run some more so where was i that
;;; was the match set {run__1^Mm} ok what rrr i just hit control h
;;; again"
```

```
;; 8-25-95 - if the goal is to figure out where we were in the run,
;; and we see assertions, but no associated builds [or other
;; evidence of having run those assertions], run 1 elaboration to
;; see where we are.
;; 1-22-96 - for some reason it doesn't work to make this
;; one-elaboration, or anything else.
```

```
(sp po*display*run*where-was-i
 (state <g> ^superstate nil
  ^operator.goal where-was-i
  ^wm.match-set <something>
  ^wm (- ^builds)
  )
-->
<g> ^operator <o>)
<o> ^name display
  ^goal run:where-was-i ; 1-22-96
  ^val where-was-i
  ^terminate-and-reject t))
```

```
(sp a*display*t364
 (state <s> ^superstate nil
  ^current-display t323
  ^operator <o> +)
<o> ^name display ^val where-was-i
  )
-->
<o> ^time t364))
```

```

(sp ao*display*print*t364
 :o-support
 (state <s> ^superstate nil
 ^current-display t364)
 -->
 (<s> ^on-display t364 + &))

(sp d*print*t364
 (state <s> ^superstate nil
 ^on-display t364
 ^display <d>))
 -->
 (<d> ^t364 <builds> + &)
 (<builds> ^previous t323
 ^spatial <spatial>
 ^builds builds-364
 ^build chunk-128 + &, chunk-129 + &)
 (<spatial> ^chunk-128 chunk-129 + &)
 )

;; 1093 Soar> run 1
;; 1094
;; 1095 Build: chunk-128
;; 1096 Build: chunk-129

;;; -----
;;; t373 - "ok what, rrr i just hit control h again, fine let's see
;;; {^?} what the chunks are doing {p_chunk-128^M} ok this chunk is
;;; testing for s constructor 16"

;; 10-3-95 - print a specific chunk (not a generic "chunk", and not a
;; more specific but still abstract-chunk), when not on display.

(sp po*display*print-chunk
 (state <g> ^superstate nil
 ^operator.goal <chunk>
 ^wm.chunk <chunk>
 ^wm (- ^abstract-chunk <chunk>))
 -->
 (<g> ^operator <o>))
 (<o> ^name display
 ^goal print-chunk
 ^val <chunk>
 ^trace <chunk>
 ^terminate-and-reject t))

(sp a*display*t373
 (state <s> ^superstate nil
 ^current-display t364
 ^operator <o> +)
 (<o> ^name display ^val chunk-128)
 -->
 (<o> ^time t373))

;; 8-28-95 - remove everything but t323 (the goal stack, which mostly
;; remains).

(sp ao*display*print*t373
 :o-support
 (state <s> ^superstate nil
 ^current-display t373)
 -->
 (<s> ^on-display t373 + &, tinit - t225 - t236 - t245 - t252 - ))

(sp d*print*t373
 (state <s> ^superstate nil
 ^on-display t373
 ^display <d>))
 -->
 (<d> ^t373 <chunk> + &)
 (<chunk> ^previous t364
 ^spatial <spatial>
 ^chunk chunk-128
 ^condition <cg> + &, <cs> + &, <co> + &, <ca> + &, <cr> + &
 ^action <a1> + &, <a2> + &
 ;; 7-10-95 - meta-attributes, like ^id for objects:
 ^leaf <co> + &, <cr> + &
 )
 (<spatial> ^<cg> <co> ^<co> <cs> ^<cs> <ca> ^<ca> <cr> ^<cr> <a1> ^<a1> <a2>))
 (<cg> ^goal g ^operator <co> ^state <cs>))
 (<cs> ^assigners <ca>))
 (<co> ^operator* s-constructor16 ^type s-model-constructor)
 (<ca> ^path-to-referent path-to-referent
 ^referent <cr>))
 (<cr> ^referent nil) ; 12-16-95 - a negated test

 (<a1> ^ul referent ^referent <a2>))
 (<a2> ^referent-of ul ^type s-model)
 )

;; 1097 Soar> p chunk-128
;; 1098 (sp chunk-128
;; 1099 :chunk
;; 1100 (goal <g1> ^operator <o1> ^state <s1>))
;; 1101 (<o1> ^name s-constructor16 ^type s-model-constructor)
;; 1102 (<s1> ^assigners <a1>))
;; 1103 (<a1> ^n <n2>))
;; 1104 (<n2> ^max <n1>))
;; 1105 (<n1> ^head <ul>))
;; 1106 (<ul> ^referent <r*1>))
;; 1107 -->
;; 1108 (<ul> ^referent <r1> + ^referent <r1> &))
;; 1109 (<r1> ^referent-of <ul> + ^type s-model +))
;; 1110
;; 1111
;; 1112 Soar>

;;; -----
;;; t431 - so it must have just changed it on, on the superstate? is

```

```

;;; that, oh these are shared states, i see ... no i don't see, what
;;; the hell built that {^[v]}

;; 1-22-96 - scroll to an sp if we've seen it or its parts before
;; before, recently or not (sps change more slowly than states; see
;; t445).
;; 1-28-96 - if the imagined-but-seen feature is an sp name
;; (assertion or SP), don't scroll if we're staring at it. we still
;; scroll if the feature is a condition (like s-model-constructor)
;; or an action, in which case the feature could be common to two
;; SPs (the one we're staring at, and the one containing the
;; imagined-but-seen code fragment).

(sp po*display*scroll*to-sp
  (state <g> ^superstate nil
    ^operator.goal <apply-create-referent>
    ^wm. << fixated-recently imagined-but-seen >>
      <apply-create-referent>
    ^wm.<< assertion sp condition action >> <apply-create-referent>
    - ^wm.fixated (^ << sp assertion >> ; 1-28-96
      <apply-create-referent>
      - ^imagined-at)
  )
  -->
  (<g> ^operator <o>)
  (<o> ^name display
    ^goal scroll:to-sp
    ^val <apply-create-referent>
    ^trace <apply-create-referent>
    ^terminate-and-reject t))

(sp a*display*t431
  (state <s> ^superstate nil
    ^current-display t373
    ^operator <o> +)
  (<o> ^name display ^val apply-create-referent)
  -->
  (<o> ^time t431))

;; 8-28-95 - this actually goes back past the beginning. remove the
;; goal stack (t323), of which just the top shows. 236
;; (apply-create-referent) is what we care about.

(sp ao*display*print*t431
  :o-support
  (state <s> ^superstate nil
    ^current-display t431)
  -->
  (<s> ^on-display t323 - t364 - t373 -
    tinit + &, t225 + &, t236 + &, t245 + &, t252 + & ))

;;; -----
;;; t445 - "this said if you're in the s construct problem space, you
;;; slap ... that attribute on the {^V} object, so i'm in the
;;; s-construct problem space, i'm going to slap that thing on there,

```

```

;;; and lo and behold i get this chunk because they share the state"

;; 1-22-96 - scroll to states we've seen before.

(sp po*display*scroll*to-state
  (state <g> ^superstate nil
    ^operator.goal <shared-state>
    ^wm. << fixated-recently imagined-but-seen >> <shared-state>
    ^wm.state <shared-state>
    - ^wm.fixated (^state <shared-state> - ^imagined-at)
  )
  -->
  (<g> ^operator <o>)
  (<o> ^name display
    ^goal scroll:to-state
    ^val <shared-state>
    ^trace <shared-state>
    ^terminate-and-reject t))

(sp a*display*t445
  (state <s> ^superstate nil
    ^current-display t431
    ^operator <o> +)
  (<o> ^name display ^val shared-state)
  -->
  (<o> ^time t445))

(sp ao*display*t445
  :o-support
  (state <s> ^superstate nil
    ^current-display t445)
  -->
  (<s> ^on-display tinit - t225 - t236 - t245 - t252 -
    t323 + &, t364 + &, t373 + &))

;;; -----
;;; t503 - "i now have it sitting there {^[>]} and the second chunk i
;;; know is just going to be testing for {p__chunk-129^M} um, the
;;; bead, right yeah there's the conjunct symbol"

;; 9-11-95 - covered by po*display*print-chunk

(sp a*display*t503
  (state <s> ^superstate nil
    ^current-display t445
    ^operator <o> +)
  (<o> ^name display ^val chunk-129)
  -->
  (<o> ^time t503))

(sp ao*display*print*t503
  :o-support
  (state <s> ^superstate nil
    ^current-display t503)
  -->

```

```

(<s> ^on-display t503 + &))
(sp d*print*t503
 (state <s> ^superstate nil
  ^on-display t503
  ^display <d>)
-->
(<d> ^t503 <chunk> + &)
(<chunk> ^previous t373
 ^spatial <spatial>
 ^chunk chunk-129
 ^condition <cg> + &, <cs> + &, <co> + &, <ca> + &, <cr> + &
 ^action <ac> + &)
(<spatial> ^<cg> <cs> ^<cs> <co>
 ^<co> <ca> ^<ca> <cr> ^<cr> <ac>)
(<cg> ^goal g ^state <cs> ^operator <co>)
(<cs> ^state initial-state ^assigners <ca>)
(<co> ^operator* s-constructor16 ^conjunct-symbol conjunct
 ^type s-model-constructor)
(<ca> ^path-to-referent path-to-referent
 ^referent <cr>)
(<cr> ^referent nil) ; 8-27-95 - was "no-referent-on-u-model"
(<ac> ^conjunct-symbol replacement))

;; 1112 Soar> p chunk-129
;; 1113 (sp chunk-129
;; 1114 :chunk
;; 1115 (goal <gl> ^state <s1> ^operator <ol>)
;; 1116 (<s1> ^name initial-state ^assigners <al>)
;; 1117 (<ol> ^name s-constructor16 ^conjunct-symbol conjunct
;; 1118 ^type s-model-constructor)
;; 1119 (<al> ^n <n2>)
;; 1120 (<n2> ^max <nl>)
;; 1121 (<nl> ^head <ul>)
;; 1122 (<ul> ^referent <r*1>)
;; 1123 -->
;; 1124 (<ol> ^conjunct-symbol conjunct - ^conjunct-symbol <cl> +))

;;; -----
;;; t513 "um, the bead, right yeah there's the conjunct symbol, fine,
;;; ok {ms^M} terminate create referent, fine..."

;; 10-4-95 - when thinking about an sp, and we know its lhs is
;; satisfied, but nothing's asserted, print the match set,
;; "expecting" the sp to be asserted.

(sp po*display*match-set*asserted-sp
 (state <g> ^superstate nil
  ^wm.sp <terminate-create-referent>
  ^wm.<terminate-create-referent> lhs
  - ^wm.assertion
  )
-->
<g> ^operator <o>)
<o> ^name display

```

```

^goal match-set:asserted-sp
^val <terminate-create-referent>
^terminate-and-reject t))

(sp a*display*t513
 (state <s> ^superstate nil
  ^current-display t503
  ^operator <o> +)
 (<o> ^name display ^val terminate-create-referent)
-->
(<o> ^time t513))

(sp ao*display*print*t513
 :o-support
 (state <s> ^superstate nil
  ^current-display t513)
-->
(<s> ^on-display t513 + &))

(sp d*print*t513
 (state <s> ^superstate nil
  ^on-display t513
  ^display <d>)
-->
(<d> ^t513 <m> + &)
(<m> ^previous t503
 ^spatial <spatial>
 ^match-set ms-513
 ^assertions terminate-create-referent + &,
 head-noun-cop-animate + &,
 head-noun-cop + &
 ^retractions-name retractions-513
 ^retractions chunk-129 + &,
 chunk-128 + &,
 touch-conjunct-symbol + &,
 create-referent-noun + &)
(<spatial> ^terminate-create-referent head-noun-cop-animate
 ^head-noun-cop-animate head-noun-cop
 ^chunk-128 chunk-129
 ;; 9-12-95 - only one is in wm:
 ^head-noun-cop chunk-128 + &, chunk-129 + &
 ^chunk-129 touch-conjunct-symbol
 ^touch-conjunct-symbol create-referent-noun)
)

;; 1127 Soar> ms
;; 1128 Assertions:
;; 1129 s-construct*terminate*create-referent
;; 1130 s-construct*propose*add-property*head-noun*cop*animate
;; 1131 s-construct*propose*add-property*head-noun*cop
;; 1132 Retractions:
;; 1133 chunk-129
;; 1134 chunk-128
;; 1135 s-construct*create-referent*touch-conjunct-symbol
;; 1136 s-construct*propose-create-referent*noun

```

```

;; 1137

;;; -----
;;; t552 - "which means that in both spaces it will have, added the
;;; properties to {run__1^M} hm! why didn't it build a chunk"

;; 10-4-95 - when we recall that the goal (search for "operator or sp")
;; modifies some target, go ahead and run it (the goal). [the
;; process of deciding on "run 1" for t552 vs. "ms, run 1" for t564
;; isn't represented, but one can imagine.]
;; 9-12-95 - fire again when we discover that the apply-sp in
;; particular, rather than the operator in general, is actually what
;; modifies the target. [we somehow get from seeing the proposal to
;; the operator itself. this is where the programmer anticipates
;; some chunks that don't happen.]
;; 1-13-96 - make sure we didn't imagine this.

(sp po*display*run-to-builds
  (state <s> ^superstate nil
    ^operator.goal <cause>
    ^wm.<cause> target ; 9-12-95
  )
  -->
  (<s> ^operator <o>)
  (<o> ^name display
    ^goal run:to-builds
    ^val <cause>
    ^terminate-and-reject t))

;; 9-13-95 - ^val and ^current-display aren't enough context.

(sp a*display*t552
  (state <s> ^superstate nil
    ^current-display t513
    ^operator <o> +
    ^wm.add-property target) ; 9-13-95
  (<o> ^name display ^val add-property)
  -->
  (<o> ^time t552 + &))

(sp ao*display*print*t552
  :o-support
  (state <s> ^superstate nil
    ^current-display t552)
  -->
  (<s> ^on-display t552 + &))

(sp d*print*t552
  (state <s> ^superstate nil
    ^on-display t552
    ^display <d>)
  -->
  (<d> ^t552 <builds> + &)
  (<builds> ^previous t513))

```

```

;;; -----
;;; t564 - "oh, those are operators, it's proposing them, fine
;;; {ms^Mrun__1^M} fine"

;; 12-12-95 - if something just built (or didn't build) chunks,
;; see what's about to fire (do more if that gives no information).
;; t564: "ms" generates empty assertions; "run 1" generates a selection
;; t587: "ms" generates contentful assertions, which is good enough.

(sp po*display*match-set*after-builds
  (state <g> ^superstate nil
    ^operator.goal <add-property>
    ^wm.<add-property> builds)
  -->
  (<g> ^operator <o>)
  (<o> ^name display
    ^goal match-set:after-builds
    ^val <add-property>
    ^terminate-and-reject t))

;; 10-4-95 - somehow this was transferring to other operators and
;; marking them with t564. it's used a lot later.

(sp a*display*t564
  :o-support
  (state <s> ^superstate nil
    ^current-display t552
    ^operator <o> +)
  (<o> ^name display ^val add-property - ^time) ; 10-4-95
  -->
  (<o> ^time t564 + &))

(sp ao*display*print*t564
  :o-support
  (state <s> ^superstate nil
    ^current-display t564)
  -->
  (<s> ^on-display t323 - t364 - t373 -
    t564 + &))

;; 9-13-95 - ignore the "ms" output (564), which scrolls off with the
;; run 1.

(sp d*print*t564
  (state <s> ^superstate nil
    ^on-display t564
    ^display <d>)
  -->
  (<d> ^t564 <ms> + &, <o> + &))

(<ms> ^previous t552
  ^match-set ms-t564
  ^assertions nil
  ^retractions-name retractions-t564
  ^retractions nil)

```

```

(<o> ^previous t552
 ^decision-cycle 77
 ^operator-id o28
 ^selected add-property
 ^argument t))

;; 1140 Soar> ms
;; 1141 Assertions:
;; 1142 Retractions:
;; 1143
;; 1144 Soar> run 1
;; 1145 77: 0: 028 add-property(t)
;; 1146 Soar>

;;; -----
;;; t575 - "fine so we're going to do it one at a time with an
;;; operator, slap them on, ok so this is the thing that's putting
;;; animate true on {ms^M} touch the conjunct symbol"

;; covered by po*display*match-set*after-builds

(sp a*display*t575
 :o-support
 (state <s> ^superstate nil
 ^current-display t564
 ^operator <o> +)
 (<o> ^name display ^val add-property - ^time)
 -->
 (<o> ^time t575 + &))

(sp ao*display*print*t575
 :o-support
 (state <s> ^superstate nil
 ^current-display t575)
 -->
 (<s> ^on-display t575 + &))

(sp d*print*t575
 (state <s> ^superstate nil
 ^on-display t575
 ^display <d>)
 -->
 (<d> ^t575 <m> + &)
 (<m> ^previous t564
 ^spatial <spatial>
 ^match-set ms-575
 ^assertions touch-conjunct-symbol + &,
 apply-add-property + &
 ^retractions-name retractions-575
 ^retractions terminate-create-referent + &,
 apply-create-referent + &)
 (<spatial> ^touch-conjunct-symbol apply-add-property
 ^apply-add-property terminate-create-referent
 ^terminate-create-referent apply-create-referent)
)

```

```

;; 1146 Soar> ms
;; 1147 Assertions:
;; 1148 s-construct*add-property*touch-conjunct-symbol
;; 1149 s-construct*add-property
;; 1150 Retractions:
;; 1151 s-construct*terminate*create-referent
;; 1152 s-construct*create-referent
;; 1153
;; 1154 Soar>

;;; -----
;;; t582 - "touch the conjunct symbol, add the properties so i should
;;; get two more chunks, yes {run_1^M}, and then i should get"

;; 1-22-96 - covered by po*display*run-to-builds

(sp ao*display*t582
 :o-support
 (state <s> ^superstate nil
 ^operator.name display
 ^current-display t582)
 -->
 (<s> ^on-display t582 + &))

(sp a*display*t582
 :o-support
 (state <s> ^superstate nil
 ^current-display t575
 ^operator <o> +)
 (<o> ^name display ^val apply-add-property - ^time)
 -->
 (<o> ^time t582 + &))

(sp d*initial*t582
 (state <s> ^superstate nil
 ^on-display t582
 ^display <d>)
 -->
 (<d> ^t582 <builds> + &)
 (<builds> ^previous t575
 ^spatial <spatial>
 ^builds builds-582
 ^build chunk-130 + &, chunk-131 + &)
 (<spatial> ^chunk-130 chunk-131 + &))

;; 1154 Soar> run 1
;; 1155
;; 1156 Build: chunk-130
;; 1157 Build: chunk-131
;; 1158 Soar>

;;; -----
;;; t587 - "and then i should get reconsiders {ms ... ^M} yup"

;; 10-4-95 - covered by

```

```

(sp a*display*t587
  (state <s> ^superstate nil
    ^current-display t582
    ^operator <o> +)
  (<o> ^name display ^val apply-add-property - ^time)
  -->
  (<o> ^time t587 + &))

(sp ao*display*print*t587
  :o-support
  (state <s> ^superstate nil
    ^current-display t587)
  -->
  (<s> ^on-display t587 + &))

(sp d*print*t587
  (state <s> ^superstate nil
    ^on-display t587
    ^display <d>)
  -->
  (<d> ^t587 <m> + &)
  (<m> ^previous t582
    ^spatial <spatial>
    ^match-set ms-587
    ^assertions terminate-add-property ; was appended with "-587"
    ^retractions-name retractions-587
    ^retractions chunk-131 + &, chunk-130 + &,
      touch-conjunct-symbol + &,
      head-noun-cop-animate + &)
  (<spatial> ^terminate-add-property chunk-131
    ^chunk-131 chunk-130
    ^chunk-130 touch-conjunct-symbol
    ^touch-conjunct-symbol head-noun-cop-animate))

;; 1158 Soar> ms
;; 1159 Assertions:
;; 1160 s-construct*terminate*add-property
;; 1161 Retractions:
;; 1162 chunk-131
;; 1163 chunk-130
;; 1164 s-construct*add-property*touch-conjunct-symbol
;; 1165 s-construct*propose*add-property*head-noun*cop*animate
;; 1166
;; 1167 Soar>

;;; -----
;;; t593 - "and {run__1 1^M ms^M} ok {run} what i'd really like to be
;;; able to do i need a command that does an ms prints out the
;;; productions in the match set and then does a run 1 for me instead
;;; of having to retype this all the time ok {__1^M}"

;; 10-4-95 - when we're thinking about an sp that terminates an
;; operator, and chunks have been built, but no new operator has
;; been selected, run up to the selection of the next operator.

```

```

(sp po*display*run*to-op-after-builds
  (state <s> ^superstate nil
    ^operator.goal <terminate-add-property>
    ^wm.<terminate-add-property> reconsider
    ^wm.builds { <> nil }
    )
  -->
  (<s> ^operator <o>)
  (<o> ^name display
    ^goal run:to-op-after-builds
    ^val <terminate-add-property>
    ^terminate-and-reject t))

(sp a*display*t593
  :o-support
  (state <s> ^superstate nil
    ^current-display t587
    ^operator <o> +)
  (<o> ^name display - ^time
    ^val terminate-add-property)
  -->
  (<o> ^time t593 + &))

(sp ao*display*t593
  :o-support
  (state <s> ^superstate nil
    ^operator.name display
    ^current-display t593)
  -->
  (<s> ^on-display t593 + &,
    t503 - t513 - t552 -))

(sp d*run-ms-run*t593
  (state <s> ^superstate nil
    ^on-display t593
    ^display <d>)
  -->
  (<d> ^t593 <ms> + &, <o> + &)
  (<ms> ^previous t587
    ^match-set ms-t593
    ^assertions nil
    ^retractions-name retractions-t593
    ^retractions nil)
  (<o> ^previous t587
    ^decision-cycle 78
    ^operator-id o27
    ^selected add-property
    ^argument policeman))

;; 1167 Soar> run 1
;; 1168
;; 1169 Soar> ms
;; 1170 Assertions:
;; 1171 Retractions:
;; 1172

```

[^:593]

```

;; 1173 Soar> run 1 [^^:596]
;; 1174 78: 0: 027 add-property(policeman)
;; 1175 Soar> [^:598]

;;; -----
;;; t618 - "fine so it's going to add the next one {run__1^M} fine now
;;; what {ms^M} yeah yeah yeah yeah

;; 10-4-95 - covered by po*display*match-set*after-selection

(sp a*display*t618
 :o-support
 (state <s> ^superstate nil
 ^current-display t593
 ^operator <o> +)
 (<o> ^name display ^val add-property - ^time)
 -->
 (<o> ^time t618 + &))

(sp ao*display*t618
 :o-support
 (state <s> ^superstate nil
 ^operator.name display
 ^current-display t618)
 -->
 (<s> ^on-display t618 + &))

(sp d*initial*t618
 (state <s> ^superstate nil
 ^on-display t618
 ^display <d>)
 -->
 (<d> ^t618 <builds> + &, <match-set> + &)
 (<builds> ^builds builds-616
 ^previous t593
 ^spatial <spatial-1>
 ^build chunk-132 + &, chunk-133 + &)
 (<spatial-1> ^chunk-132 chunk-133)

(<match-set> ^match-set ms-616
 ^previous t593
 ^spatial <spatial-2>
 ^assertions terminate-add-property
 ^retractions-name retractions-616
 ^retractions chunk-132 + &,
 touch-conjunct-symbol + &,
 head-noun-cop + &)

(<spatial-2> ^terminate-add-property chunk-132
 ^chunk-132 touch-conjunct-symbol
 ^touch-conjunct-symbol head-noun-cop))

;; 1175 Soar> run 1
;; 1176
;; 1177 Build: chunk-132

```

```

;; 1178 Build: chunk-133
;; 1179 Soar>
;; 1179 Soar> ms
;; 1180 Assertions:
;; 1181 s-construct*terminate*add-property
;; 1182 Retractions:
;; 1183 chunk-132
;; 1184 s-construct*add-property*touch-conjunct-symbol
;; 1185 s-construct*propose*add-property*head-noun*cop
;; 1186
;; 1187 Soar> run 1

;;; -----
;;; t621 - {run} sooo this is going to reconsider that i assume {__}
;;; it's going to get exhaustion {1^M ms^M run__1^M} yeah

;; 12-12-95 - when an assertion reconsiders the current
;; (add-property) operator, and there are two sets of chunks (one
;; for each of two properties) this word is comprehended, so run up
;; to the exhausted operator. [could separate out a
;; "comprehension-done" goal; "^selected exhausted" inhibits a
;; refracted fixate]
;; 1-13-96 - two builds -> two args, plus ^operator.

(sp po*display*run*to-end-of-space
 (state <s> ^superstate nil
 ^operator.goal <terminate-add-property>
 ^wm.<terminate-add-property> reconsider
 ^wm.argument <policeman> { <> <policeman> <t> }
 )
 -->
 (<s> ^operator <o>)
 (<o> ^name display
 ^goal run:to-end-of-space
 ^val run-to-end-of-space
 ^terminate-and-reject t))

(sp a*display*t621
 :o-support
 (state <s> ^superstate nil
 ^current-display t618
 ^operator <o> +)
 (<o> ^name display ^val run-to-end-of-space - ^time)
 -->
 (<o> ^time t621 + &))

(sp ao*display*t621
 :o-support
 (state <s> ^superstate nil
 ^operator.name display
 ^current-display t621)
 -->
 (<s> ^on-display t621 + &))

```

```

(sp d*run-ms-run*t621
  (state <s> ^superstate nil
    ^on-display t621
    ^display <d>)
  -->
  (<d> ^t621 <ms> + &, <o> + &)
  (<ms> ^previous t618
    ^match-set ms-t621
    ^assertions nil
    ^retractions-name retractions-t621
    ^retractions nil)
  (<o> ^decision-cycle 79
    ^previous t618 ; 9-17-95 - point over 618
    ^operator-id o26
    ^selected exhausted))

;; 1187 Soar> run 1
;; 1188
;; 1189 Soar> ms
;; 1190 Assertions:
;; 1191 Retractions:
;; 1192
;; 1193 Soar> run 1
;; 1194 79: O: 026 exhausted
;; 1195 Soar> pgs

;;; -----
;;; t639 - "ok {pgs^M} where am i"

;; 10-4-95 - covered by po*display*print-stack

(sp a*display*t639
  (state <s> ^superstate nil
    ^current-display t621
    ^operator <o> +)
  (<o> ^name display ^val current-context)
  -->
  (<o> ^time t639))

(sp ao*display*print*t639
  :o-support
  (state <s> ^superstate nil
    ^current-display t639)
  -->
  (<s> ^on-display t639 + &,
    t564 - t564 - t575 - t582 - t587 - t593 - t618 -))

(sp d*pgs*t639
  (state <s> ^superstate nil
    ^on-display t639
    ^display <d>)
  -->
  (<d> ^t639 <stack> + &)
  (<stack> ^context <c1> + &, <c2> + &, <c3> + &, <c4> + &
    ^previous t621))

```

```

(<c1> ^operator* comprehend-input ^op-id o2 ^state s1
  ^impasse* nil
  ^problem-space top-ps
  ^older none ^newer <c2>)
(<c2> ^operator* none ^state s4
  ^impasse* operator-no-change
  ^problem-space comprehension
  ^older <c1> ^newer <c3>)
(<c3> ^operator* s-constructor16 ^op-id o24 ^state s15
  ^impasse* state-no-change
  ^problem-space create-operator
  ^older <c2> ^newer <c4>)
(<c4> ^operator* exhausted ^op-id o26 ^state s15
  ^impasse* operator-no-change
  ^problem-space s-construct
  ^older <c3> ^newer none))

;; 1195 Soar> pgs
;; 1196 : ==>G: G1
;; 1197 : P: P1 top-ps
;; 1198 : S: S1
;; 1199 : O: O2 comprehend-input
;; 1200 : ==>G: G2 operator no-change
;; 1201 : P: P2 comprehension
;; 1202 : S: S4
;; 1203 : ==>G: G15 state no-change
;; 1204 : P: P68 create-operator
;; 1205 : S: S15
;; 1206 : O: O24 s-constructor16
;; 1207 : ==>G: G16 operator no-change
;; 1208 : P: P85 s-construct
;; 1209 : S: S15
;; 1210 : O: 026 exhausted
;; 1211
;; 1212 Soar>

;;; -----
;;; t646 "s15 now has an utterance model object {[v ^[v] u something, u20"

;; 10-4-95 - if we want to comprehend an object, and we know we've seen
;; a feature of that object before, but it's not on the display now,
;; scroll.

(sp po*display*scroll*to-object
  (state <g> ^superstate nil
    ^operator.goal <u-model>
    ^wm.object <u-model>
    ^wm. << fixated-recently imagined-but-seen >>
      <referent>
    ^wm.<u-model> <referent>
    - ^wm.fixated (^attribute <referent> - ^imagined-at)
  )
  -->
  (<g> ^operator <o>)
  (<o> ^name display

```

```

^goal scroll:to-object
^val <u-model>
^trace <u-model>
^terminate-and-reject t))

(sp a*display*t646
 (state <s> ^superstate nil
  ^current-display t639
  ^operator <o> +)
 (<o> ^name display ^val u-model - ^time)
 -->
 (<o> ^time t646))

(sp ao*display*print*t646
 :o-support
 (state <s> ^superstate nil
  ^current-display t646)
 -->
 (<s> ^on-display t245 + &, t252 + &, t323 + &, t364 + &, t373 + &, t503 + &
  t621 - t639 -))

;;; -----
;;; t649 - "u something, u20 let's look at u20 {[^>p__u20^M]"

;; 9-20-95 - when trying to comprehend an object, if we have its id
;; but haven't printed the object, go to the bottom and print.
;; 1-13-96 - if the object is on display, it's stale. otherwise, same
;; as po*display*print-object.

(sp po*display*print-object*fresh
 (state <g> ^superstate nil
  ^operator.goal <for>
  ^wm.<for> <u20> { <> <u20> <other-k> }
  ^wm.object-id <u20>
  ^wm (- ^id <other-k>)
  - ^wm.dp.newest ; 1-13-96
 )
 -->
 (<g> ^operator <o>)
 (<o> ^name display
  ^goal print-object-fresh
  ^val <for>
  ^trace <u20>
  ^terminate-and-reject t))

(sp a*display*t649
 (state <s> ^superstate nil
  ^current-display t646
  ^operator <o> +)
 (<o> ^name display ^val u-model - ^time)
 -->
 (<o> ^time t649))

(sp ao*display*print*t649
 :o-support

```

```

(state <s> ^superstate nil
  ^current-display t649)
 -->
 (<s> ^on-display t245 - t252 - t323 - t364 - t373 - t503 -
  t621 + &, t639 + &, t649 + &))

(sp d*print*t649
 (state <s> ^superstate nil
  ^on-display t649
  ^display <d>)
 -->
 (<d> ^t649 <o> + &)
 (<o> ^previous t639
  ^meta-attribute meta-attribute + &, object-id + &, id + &,
  attribute + &, value + &
  ^object-id u20
  ^id u20 + &, w8 + &, w13 + &, e15 + &, u14 + &, u15 + &, u17 + &, r9 + &
  ^attribute left-edge + &, right-edge + &, bar-level + &,
  word-id + &, category + &, annotation + &,
  empty-node + &, spec + &, zero-head + &, head + &,
  referent + &
  ^value u20 + &, w8 + &, w13 + &, e15 + &, u14 + &, u15 + &, u17 + &,
  max + &, n + &, specified + &, r9 + &
  ;; the actual object:
  ^left-edge W8 ^right-edge W13 ^bar-level max ^word-id W13 ^category n
  ^annotation specified ^empty-node E15 ^spec U14 ^zero-head U15
  ^head U17 ^referent R9))

;; 1212 Soar> p u20
;; 1213 (U20 ^referent R9 ^left-edge W8 ^right-edge W13 ^bar-level max ^word-id W13
;; 1214 ^category n ^annotation specified ^empty-node E15 ^spec U14
;; 1215 ^zero-head U15 ^head U17)
;; 1216
;; 1217 Soar>

;;; -----
;;; t654 - "right, which has referent r9 {p__r9} it's sitting there,
;;; it has two properties, everything looks good"

;; 10-4-95 - covered by po*display*print-object

(sp a*display*t654
 (state <s> ^superstate nil
  ^current-display t649
  ^operator <o> +)
 (<o> ^name display ^val referent - ^time)
 -->
 (<o> ^time t654))

(sp ao*display*print*t654
 :o-support
 (state <s> ^superstate nil
  ^current-display t654)
 -->
 (<s> ^on-display t654 + &))

```

```

(sp d*print*t654
  (state <s> ^superstate nil
    ^on-display t654
    ^display <d>))
-->
(<d> ^t654 <o> + &)
(<o> ^previous t649
  ^meta-attribute meta-attribute + &, object-id + &, id + &,
  attribute + &, value + &
  ^object-id r9
  ^id u20 + &, p87 + &, p88 + &
  ^attribute type + &, properties + &, referent-of + &
  ^value P87 + &, P88 + &, U20 + &, s-model + &
  ;; the object:
  ^properties P87 + &, P88 + &
  ^referent-of U20 ^type s-model))

;; 1217 Soar> p r9
;; 1218 (R9 ^properties P87 ^properties P88 ^referent-of U20 ^type s-model)
;; 1219
;; 1220 Soar>

;;; -----
;;; t685 - "here's the big question, do i want to let it return this
;;; thing ... how is it going to return this thing {ms} implement
;;; exhausted, what does that do"

(sp a*display*t685
  (state <s> ^superstate nil
    ^current-display t654
    ^operator <o> +)
  (<o> ^name display ^val exhausted - ^time)
  -->
  (<o> ^time t685))

(sp ao*display*print*t685
  :o-support
  (state <s> ^superstate nil
    ^current-display t685)
  -->
  (<s> ^on-display t685 + &))

(sp d*print*t685
  (state <s> ^superstate nil
    ^on-display t685
    ^display <d>))
-->
(<d> ^t685 <m> + &)
(<m> ^match-set ms-685
  ^previous t654
  ^spatial <spatial>
  ^assertions implement-exhausted
  ^retractions terminate-add-property + &,
  apply-add-property + &)
(<spatial> ^implement-exhausted terminate-add-property

```

```

  ^terminate-add-property apply-add-property))

;; 1220 Soar> ms
;; 1221 Assertions:
;; 1222 s-construct*implement*exhausted
;; 1223 Retractions:
;; 1224 s-construct*terminate*add-property
;; 1225 s-construct*add-property
;; 1226

;;; -----
;;; t691 - "implement exhaust ed what does that do
;;; {p__s-construct*implement*exhausted^M}"

;; 10-4-95 - covered by po*display*print-sp*when-an-apply-sp

(sp a*display*t691
  (state <s> ^superstate nil
    ^current-display t685
    ^operator <o> +)
  (<o> ^name display ^val implement-exhausted - ^time)
  -->
  (<o> ^time t691))

(sp ao*display*print*t691
  :o-support
  (state <s> ^superstate nil
    ^current-display t691)
  -->
  (<s> ^on-display t691 + &))

(sp d*print*t691
  (state <s> ^superstate nil
    ^on-display t691
    ^display <d>))
-->
(<d> ^t691 <sp> + &)
(<sp> ^previous t685
  ^spatial <spatial>
  ^sp implement-exhausted
  ^condition <cg> + &, <co> + &, <cp> + &, <cs> + &,
  <csp> + &, <cso> + &
  ^action <a1> + &, <a2> + &
  ^leaf <co> + &, <cp> + &, <csp> + & ; 9-21-95 - ?
  )
(<spatial> ^<cg> <co> + &, ^<co> <cp> + &, ^<cp> <cs> + &, ^<cs> <csp> + &,
  ^<csp> <cso> + &, ^<cso> <a1> + &, ^<a1> <a2> + &)
(<cg> ^goal g ^operator* <co> ^problem-space <cp> ^state state ^object <cs>)
(<co> ^operator* exhausted)
(<cp> ^problem-space s-construct)
(<cs> ^problem-space <csp> ^operator* <cso> ^state superstate)
(<csp> ^problem-space create-operator)
(<cso> ^operator* name*)

(<a1> ^exhausted reconsider)

```

```

(a2> ^superstate construction-done))

;; 1227 Soar> p s-construct*implement*exhausted
;; 1228 (sp s-construct*implement*exhausted
;; 1229 (goal <g> ^operator <o> ^problem-space <p> ^state <s> ^object <sg>)
;; 1230 (<o> ^name exhausted)
;; 1231 (<p> ^name s-construct)
;; 1232 (<sg> ^problem-space <p*1> ^operator <o*1> ^state <ss>)
;; 1233 (<p*1> ^name create-operator)
;; 1234 (<o*1> ^name <name>)
;; 1235 -->
;; 1236 (<g> ^operator <o> @)
;; 1237 (<ss> ^annotation construction-done + ^annotation construction-done &))
;; 1238
;; 1239

;;; -----
;;; t720 - "so it's going to slap construction done on s15
;;; {run_1^Mp_s15} ... s-construction done, u-constructor applied
;;; u-model success"

;; 9-21-95 - when there's a modification pending to the superstate,
;; make it and print the result.

(sp po*display*run*action-and-print
 (state <s> ^superstate nil
  ^operator.goal <superstate>
  ^wm.action <superstate>
  ^wm.<superstate> <s15>
  ^wm.id <s15>
 )
-->
(<s> ^operator <o>)
<o> ^name display
 ^goal run:action-and-print
 ^val <superstate>
 ^terminate-and-reject t))

(sp a*display*t720
 (state <s> ^superstate nil
  ^current-display t691
  ^operator <o> +)
 (<o> ^name display ^val superstate - ^time)
-->
<o> ^time t720))

(sp ao*display*print*t720
 :o-support
 (state <s> ^superstate nil
  ^current-display t720)
-->
<s> ^on-display t720 + &,
 t618 - t621 - t639 - ))

(sp d*print*t720

```

```

(state <s> ^superstate nil
 ^on-display t720
 ^display <d>)
-->
<d> ^t720 <build> + &, <object> + &)
<build> ^previous t691
 ^builds builds-720
 ^build chunk-134)
<object> ^previous t691
 ^meta-attribute meta-attribute + &, object-id + &, id + &,
 attribute + &, value + &
 ^object-id s15
 ^id s15 + &, o3 + &, a3 + &, f4 + &, a64 + &, r7 + &,
 a65 + &, r8 + &
 ^attribute annotation + &, name + &, ordering-info + &,
 adjacency-info + &, for-formatting + &,
 assigners + &, receivers + &,
 assigners2 + &, receivers2 + &
 ^value construction-done + &, u-constructor-applied + &,
 u-model-success + &, o3 + &, a3 + &, f4 + &,
 a64 + &, r7 + &, a65 + &, r8 + &
;; the actual object:
 ^annotation construction-done + &, u-constructor-applied + &,
 u-model-success + & ^name initial-state ^ordering-info O3
 ^adjacency-info A3 ^for-formatting F4 ^assigners A64 ^receivers R7
 ^assigners2 A65 ^receivers2 R8))

;; 1240 Soar> run 1
;; 1241
;; 1242 Build: chunk-134
;; 1243 Soar> p s15
;; 1244 (S15 ^annotation construction-done ^annotation u-constructor-applied
;; 1245 ^annotation u-model-success ^name initial-state ^ordering-info O3
;; 1246 ^adjacency-info A3 ^for-formatting F4 ^assigners A64 ^receivers R7
;; 1247 ^assigners2 A65 ^receivers2 R8)
;; 1248
;; 1249 Soar>

;;; -----
;;; t730 - "s-construction done u-constructor applied u-model success,
;;; this {ms^M} is going to let me propose the return operator"

;; 10-4-95 - covered by po*display*match-set*asserted-sp

(sp a*display*t730
 (state <s> ^superstate nil
  ^current-display t720
  ^operator <o> +)
 (<o> ^name display ^val propose-return-operator - ^time)
-->
<o> ^time t730))

(sp ao*display*print*t730
 :o-support
 (state <s> ^superstate nil

```

```

        ^current-display t730)
-->
(<s> ^on-display t730 + &))
(sp d*print*t730
 (state <s> ^superstate nil
  ^on-display t730
  ^display <d>))
-->
(<d> ^t730 <m> + &)
(<m> ^previous t720
 ^spatial <spatial>
 ^match-set ms-730
 ^assertions propose-return-operator + &,
  terminate-s-model-constructor + &
 ^retractions nil)
(<spatial> ^propose-return-operator terminate-s-model-constructor))

;; 1249   Soar> ms
;; 1250   Assertions:
;; 1251     create-operator*propose-return-operator
;; 1252     create-operator*terminate-s-model-constructor
;; 1253   Retractions:
;; 1254
;; 1255   Soar>

;;; -----
;;; t738 - "this is going to let me propose the return operator, where
;;; {pgs} create, in the create operator space"

;; 10-4-95 - covered by po*display*print-stack

(sp a*display*t738
 (state <s> ^superstate nil
  ^current-display t730
  ^operator <o> +)
 (<o> ^name display ^val proposal-context - ^time)
-->
 (<o> ^time t738))

(sp ao*display*print*t738
 :o-support
 (state <s> ^superstate nil
  ^current-display t738)
-->
 (<s> ^on-display t738 + &))

(sp d*print*t738
 (state <s> ^superstate nil
  ^on-display t738
  ^display <d>))
-->
 (<d> ^t738 <stack> + &)
 (<stack> ^context <c1> + &, <c2> + &, <c3> + &, <c4> + &
 ^previous t730)

```

```

 (<c1> ^operator* comprehend-input ^op-id o2 ^state s1
  ^impasse* nil
  ^problem-space top-ps
  ^older none ^newer <c2>))
 (<c2> ^operator* none ^state s4
  ^impasse* operator-no-change
  ^problem-space comprehension
  ^older <c1> ^newer <c3>))
 (<c3> ^operator* s-constructor16 ^op-id o24 ^state s15
  ^impasse* state-no-change
  ^problem-space create-operator
  ^older <c2> ^newer <c4>))
 (<c4> ^operator* exhausted ^op-id o26 ^state s15
  ^impasse* operator-no-change
  ^problem-space s-construct
  ^older <c3> ^newer none))

;; 1255   Soar> pgs
;; 1256     : ==>G: G1
;; 1257     :   P: P1 top-ps
;; 1258     :   S: S1
;; 1259     :   O: O2 comprehend-input
;; 1260     :   ==>G: G2 operator no-change
;; 1261     :   P: P2 comprehension
;; 1262     :   S: S4
;; 1263     :   ==>G: G15 state no-change
;; 1264     :   P: P68 create-operator
;; 1265     :   S: S15
;; 1266     :   O: O24 s-constructor16
;; 1267     :   ==>G: G16 operator no-change
;; 1268     :   P: P85 s-construct
;; 1269     :   S: S15
;; 1270     :   O: O26 exhausted
;; 1271
;; 1272   Soar>

;;; -----
;;; t759 "i think we're getting close to the right place, terminate
;;; s-model constructor, see what that's doing {p
;;; terminate-s-model-constructor}"

;; 10-4-95 - if the goal is to comprehend an sp that's about to fire,
;; and we're being careful, print the sp (if it's not already).

(sp po*display*print-sp*when-paying-attention
 (state <g> ^superstate nil
  ^operator.goal <terminate-smc>
  ^wm.assertion <terminate-smc>
  ^wm.high-level-goal pay-attention
  )
-->
 (<g> ^operator <o>))
 (<o> ^name display
 ^goal print-sp:paying-attention
 ^val <terminate-smc>

```

```

    ^trace <terminate-smc>
    ^terminate-and-reject t))

(sp a*display*t754
  (state <s> ^superstate nil
    ^current-display t738
    ^operator <o> +)
  (<o> ^name display ^val terminate-s-model-constructor - ^time)
  -->
  (<o> ^time t754))

(sp ao*display*print*t754
  :o-support
  (state <s> ^superstate nil
    ^current-display t754)
  -->
  (<s> ^on-display t754 + &
    ^on-display t649 - t654 - t685 - t691 - t720 - t730 -))

(sp d*print*t754
  (state <s> ^superstate nil
    ^on-display t754
    ^display <d>)
  -->
  (<d> ^t754 <sp> + &, <i> + &)
  (<sp> ^previous t738
    ^spatial <spatial>
    ^sp terminate-s-model-constructor
    ^condition <cg> + &, <cs> + &, <co> + &, <cp> + &
    ^action <al> + &
    ^leaf <cs> + &, <co> + &, <cp> + &,
    )
  (<spatial> ^<cg> <cs> ^<cs> <co> ^<co> <cp> ^<cp> <al>)
  (<cg> ^goal g ^state <cs> ^operator* <co> ^problem-space <cp>)
  (<cs> ^annotation construction-done)
  (<co> ^type s-model-constructor)
  (<cp> ^problem-space create-operator)
  (<al> ^s-model-constructor reconsider)
  )

;; 1272 Soar> p create-operator*terminate-s-model-constructor
;; 1273 (sp create-operator*terminate-s-model-constructor
;; 1274 (goal <g> ^state <s*1> ^operator <o> ^problem-space <p>))
;; 1275 (<s*1> ^annotation construction-done)
;; 1276 (<o> ^type s-model-constructor)
;; 1277 (<p> ^name create-operator)
;; 1278 -->
;; 1279 (<g> ^operator <o> @))
;; 1280
;; 1281
;; 1282 Soar>

;;; -----
;;; t770 - "ok fine that looked for the construction done {^[z ^[z
;;; ^[z] to put out the reconsider, now what is this doing"

```

```

;; 9-25-95 - covered by po*display*scroll*sp

(sp a*display*t770
  (state <s> ^superstate nil
    ^current-display t754
    ^operator <o> +)
  (<o> ^name display ^val propose-return-operator - ^time)
  -->
  (<o> ^time t770))

(sp ao*display*print*t770
  :o-support
  (state <s> ^superstate nil
    ^current-display t770)
  -->
  (<s> ^on-display t730 + &))

;;; -----
;;; t774 "now what is this doing {(mouse copy)
;;; p__create-operator*propose-return-operator^M (mouse paste)}
;;; that also looks for the construction done it says"

;; 1-21-96 - if the goal object went away and came back, and it's
;; important, print it. [perhaps she planned to print this before
;; t770, but perhaps the plan is mediated by the display.]

(sp po*display*print*high-level-goal
  (state <g> ^superstate nil
    ^operator.goal <propose-return-operator>
    ^wm.fixated-recently <propose-return-operator>
    ^wm.fixated.<assertion> <propose-return-operator>
    ^wm.high-level-goal <return-new-pointer>
    ^wm.<return-new-pointer> <propose-return-operator>
    )
  -->
  (<g> ^operator <o>)
  (<o> ^name display
    ^goal print:high-level-goal
    ^val <propose-return-operator>
    ^trace <propose-return-operator>
    ^terminate-and-reject t))

(sp a*display*t774
  (state <s> ^superstate nil
    ^current-display t770
    ^operator <o> +)
  (<o> ^name display ^val propose-return-operator - ^time)
  -->
  (<o> ^time t774))

(sp ao*display*print*t774
  :o-support
  (state <s> ^superstate nil
    ^current-display t774)
  -->

```

```

(<s> ^on-display t774 + &))

(sp d*print*t774
 (state <s> ^superstate nil
  ^on-display t774
  ^display <d>)
-->
(<d> ^t774 <sp> + &)
(<sp> ^previous t754
 ^spatial <spatial>
 ^sp propose-return-operator
 ^condition <cg> + &, <cs> + &, <cp> + &, <co> + &
 ^action <a1> + &, <a2> + &
 ^leaf <cs> + &, <cp> + &, <co> + &
 )
(<spatial> ^<cg> <cs> ^<cs> <cp> ^<cp> <co> ^<co> <a1> ^<a1> <a2>)
(<cg> ^goal g ^state <cs> ^problem-space <cp>
 ^operator* <co> + &, op + &) ; also "op" because it's bound
(<cs> ^annotation construction-done)
(<cp> ^problem-space create-operator)
(<co> ^type s-model-constructor + &, u-model-constructor + &)

(<a1> ^return-operator <a2>)
(<a2> ^new-operator op))

;; 1282 Soar> p create-operator*propose-return-operator      [^^:122]
;; 1283 (sp create-operator*propose-return-operator
;; 1284 (goal <g> ^state <s*1> ^problem-space <p> ^operator <op>)
;; 1285 (<s*1> ^annotation construction-done)
;; 1286 (<p> ^name create-operator)
;; 1287 (<op> ^type { << u-model-constructor s-model-constructor >> <t*1> })
;; 1288 -->
;; 1289 (<g> ^operator <o> + ^operator <o> >)
;; 1290 (<o> ^name return-operator + ^new-operator <op> +))
;; 1291

;;; -----
;;; t811 - "oh i see so it's going to put s-constructor 16 out there
;;; so let's do that {run__1^Mms^Mrun__1^M}"

;; 12-23-95 - when thinking about a variable, and we know the operator
;; that it binds to, and that operator isn't selected, run til it is.

(sp po*display*run*to-expected-op
 (state <s> ^superstate nil
  ^operator.goal <op>
  ^wm.variable <op>
  ^wm.<op> operator*
  ^wm.operator* <s-constructor16>
 )
-->
(<s> ^operator <o>)
<o> ^name display
 ^goal run:to-expected-op
 ^val <op>

```

```

^terminate-and-reject t))

(sp a*display*t811
 (state <s> ^superstate nil
  ^current-display t774
  ^operator <o> +)
 (<o> ^name display ^val op - ^time)
-->
 (<o> ^time t811))

(sp ao*display*print*t811
 :o-support
 (state <s> ^superstate nil
  ^current-display t811)
-->
 (<s> ^on-display t811 + &))

(sp d*print*t811
 (state <s> ^superstate nil
  ^on-display t811
  ^display <d>)
-->
 (<d> ^t811 <o> + &)
 (<o> ^previous t774
  ^decision-cycle 80
  ^operator-id o29
  ^selected return-operator))

;; 1293 Soar> run 1
;; 1294
;; 1295 Soar> ms
;; 1296 Assertions:
;; 1297 Retractions:
;; 1298
;; 1299 Soar> run 1
;; 1300 80: 0: 029 return-operator
;; 1301 Soar>

;;; -----
;;; t817 - "there's the return operator, so far so good {o^?p__o29}"

;; 12-20-95 - covered by po*display*ms-after-selection

(sp a*display*t817
 (state <s> ^superstate nil
  ^current-display t811
  ^operator <o> +)
 (<o> ^name display ^val op - ^time)
-->
 (<o> ^time t817))

(sp ao*display*print*t817
 :o-support
 (state <s> ^superstate nil
  ^current-display t817)

```

```

-->
(<s> ^on-display t817 + &))

(sp d*print*t817
 (state <s> ^superstate nil
  ^on-display t817
  ^display <d>))

-->
(<d> ^t817 <o> + &)
(<o> ^previous t811
 ^meta-attribute object-id + &, id + &, attribute + &, value + &
 ^object-id o29 ; 5-14-95 - meta-attribute - use it?
 ^id o29 + &, o24 + & ; 5-14-95 - a meta-attribute everywhere?
 ^attribute new-operator ^value o24
 ^name* return-operator ^new-operator o24))

;; 1301 Soar> p o29
;; 1302 (O29 ^name return-operator ^new-operator O24)
;; 1303
;; 1304 Soar>

;;; -----
;;; t845 - "eeuuu, rats, i think all those chunks i built {^[v] test
;;; for operator six {^[v] uh s-whatever-it-is, i think they test for
;;; the s-constructor {^[v] let's see shall we, yeah s-constructor16,
;;; there it is"

;; 1-26-96 - covered by po*display*scroll*to-sp

(sp a*display*t845
 (state <s> ^superstate nil
  ^current-display t817
  ^operator <o> +)
 (<o> ^name display ^val s-constructor16 - ^time)
 -->
 (<o> ^time t845))

(sp ao*display*print*t845
 :o-support
 (state <s> ^superstate nil
  ^current-display t845)
 -->
 (<s> ^on-display t730 - t738 - t754 - t774 - t811 - t817 -
 t323 + &, t364 + &, t373 + &, t503 + &))

```

C.3.3. fixate.soar

```

po*fixate*condition..... 157
po*fixate*action..... 157
po*fixate*binding-attribute*target..... 157

```

```

po*fixate*binding-context..... 157
po*fixate*nil-object..... 158
po*fixate*postpone..... 158
po*fixate*operators..... 158
po*fixate*current-context..... 158
po*fixate*no-referent..... 158
po*fixate*where-was-i*chunks-built..... 159
po*fixate*where-was-i*assertions..... 159
po*fixate*builds..... 159
po*fixate*chunk..... 159
po*fixate*no-problem-space..... 159
po*fixate*operator-targets..... 159
p*fixate*superstate..... 159
po*fixate*shared-state..... 160
po*fixate*assertions..... 160
po*fixate*current-context*shared-state..... 160
po*fixate*chunk*second..... 160
po*fixate*action..... 160
po*fixate*chunks-retracting..... 161
po*fixate*actions-refract..... 161
po*fixate*no-chunk..... 161
po*fixate*selected-operator..... 161
po*fixate*argument..... 161
po*fixate*previous-argument..... 161
po*fixate*sp-causes-builds..... 162
po*fixate*referent..... 162
po*fixate*state-id..... 162
po*fixate*id-of-imagined-object..... 162
p*fixate*u-something..... 162
po*fixate*augmentation..... 162
po*fixate*two-valued-attribute..... 163
po*fixate*superstate-id..... 163
po*fixate*annotations..... 163
po*fixate*proposal-context..... 163
po*fixate*assertion..... 163
p*fixate*nil-assertion-worst..... 164
p*fixate*assertion*pay-attention..... 164
po*fixate*binding-attribute*param..... 164
po*fixate*bind-op*space..... 164
po*fixate*selected-id..... 164
p*fixate*operator-id..... 164
po*fixate*bind-op*id..... 165
po*fixate*s-model-constructor..... 165

```

```

;; -*- Mode: Sde -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; File : fixate.sll.soar
;; Author : Erik Altmann
;; Created On : Wed Jun 29 13:30:13 1994
;; Last Modified By : Erik Altmann <altmann@ALECTRO.SOAR.CS.CMU.EDU>
;; Last Modified : On 01 Aug 1995, 14:27:11
;; Update Count : 1907
;;

```

```

;;
;; PURPOSE - Proposals for the fixate operator.
;;
;; Codes:
;; "tt" - goal test. "1" = ^superstate.operator.goal
;;           (most goal-dependent)
;;           "2" = ^goal
;;           "3" = no test for the goal
;;           (least goal-dependent)
;; "pref" - preference. "n" = none, "y" = best,
;;           "i" = interleave-best
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; -----
;;; look at conditions and actions

;; 10-13-95 - when we're thinking about a lhs, look at displayed
;; conditions.

(sp po*fixate*condition ; ftt: 2; fpref: n
  (state <s> ^superstate.operator
    ^goal lhs
    ^wm.dp.<reg> (^condition (<c> ^<att> <val>)
      - ^condition <val>) ; not the id values
      - ^wm (^condition <att> ^<att> <val>)
      )
  -->
  (<s> ^operator <o>)
  (<o> ^name fixate ^region <reg> ^terminate-and-reject t
    ^spatial <c>
    ^condition <att> + &, <val> + &
    ^<att> <val>
    ^trace :condition + &, <att> + &, <val> + &))

;; 10-13-95 - when we're thinking rhs, look at displayed actions.

(sp po*fixate*action ; ftt: 2; fpref: n
  (state <s> ^superstate.operator
    ^goal rhs
    ^wm.dp.<reg> (^action (<a> ^<att> <val>)
      - ^action <val>) ; not the id values
      - ^wm (^action <att> ^<att> <val>)
      )
  -->
  (<s> ^operator <o>)
  (<o> ^name fixate ^region <reg> ^terminate-and-reject t
    ^spatial <a>
    ^action <att> + &, <val> + &
    ^<att> <val>
    ^trace :action + &, <att> + &, <val> + &))

;; 10-16-95 - if there's a variable on the rhs, see what attribute
;; binds it.
;;

```

```

;; if the display reads:
;;
;;   (sp (^for <obj>) --> (<obj> ^referent <r>))
;;
;; encode "^bound-by for ^variable obj" (and other stuff). cf
;; :bind-param, which here would focus on "<r>" instead of "<obj>".
;;
;; 9-21-95 - [context objects, like operator*, are different. see
;; :bind-obj.]

(sp po*fixate*binding-attribute*target ; ftt: 3; fpref: n
  (state <s> ^superstate.operator
    ^wm (- ^bound-by <for>)
    ^wm.dp.<reg>.condition. { <> operator* <for> } <obj> ; 9-21-95
    ^wm.dp.<reg>.action.<obj> <r>))
  -->
  (<s> ^operator <o>)
  (<o> ^name fixate ^region <reg> ^terminate-and-reject t
    ^bound-by <for>
    ^condition <for>
    ^variable <obj>
    ^trace bound-by + &, <for> + &, :bind-target + &))

;; 7-30-95 - if a rhs variable is bound-by some attribute on the lhs,
;; see what object the attribute augments (in the test).
;;
;; if the display reads:
;;
;;   (goal <g> ^operator <o>)
;;   (<o> ^name create-referent ^for <obj>)
;;
;;   -->
;;   (<obj> ^referent <r> + ^referent <r> &)
;;   (<r> ^referent-of <obj> + ^type s-model +))
;;
;; this encodes "^operator* create-referent" as the context that
;; binds "^for <obj>". this means we could look elsewhere for the
;; create-referent operator, identify its parts, and thereby find
;; which part binds to <obj>.
;;
;; 1-14-96 - the goal is always a binding context, and gives no
;; information.

(sp po*fixate*binding-context ; ftt: 2; fpref: i
  (state <s> ^superstate.operator
    ^goal lhs
    ^wm.bound-by <for>
    ^wm (- ^binding-context <operator*>)
    ^wm.dp.<reg>
    (^condition
      (^ { <> goal <operator*> } <create-referent> ; 1-13-96
        ^ { <> <operator*> <for> })
      - ^condition <create-referent>)) ; not id values
  -->
  (<s> ^operator <o>)
  (<o> ^name fixate ^region <reg> ^terminate-and-reject t

```

```

^interleave-best t
^binding-context <operator*>
^<operator*> <create-referent>
^<for> <create-referent>
^trace <operator*> + &, <for> + &, <create-referent> + &,
      :bind-obj + &))

;;; -----
;;; t306 - "because this is just the bare node, it doesn't have any of
;;; the properties on it

;; 10-16-95 - if we're thinking about returning structure, and a rhs
;;   action creates something but we don't know what, imagine an empty
;;   object [enables imagine*postpone.]

(sp po*imagine*nil-object          ; itt: 1; ipref: b f?:n
  (state <s> ^superstate.operator.goal return-new-pointer
    ^wm.action <obj>
    ^wm.<obj> <referent>)
  -->
  (<s> ^operator <o> + >)
  (<o> ^name imagine ^terminate-and-reject t
    ^object <referent>
    ^nil <referent>          ; 9-20-95 - ^<ref> nil = "no <ref>"
    ^trace nil + & <referent> + &))

;; 8-21-95 - imagine the option of waiting to return new structure
;; [formed perhaps through reflection on the problems that result if
;; you return a pointer at the wrong time]

(sp po*imagine*postpone          ; itt: 2; ipref: b f?:n
  (state <s> ^goal return-new-pointer
    ^wm.object <referent>
    ^wm.nil <referent>)
  -->
  (<s> ^operator <o> + >)
  (<o> ^name imagine ^terminate-and-reject t
    ^high-level-goal postpone
    ^trace high-level-goal + &, postpone + &))

;;; -----
;;; t315 - "i could leave it on, until, exhaustion in this space?"

;; 8-16-95 - given that we're thinking about postponing until some
;; point, generate different potential points. these are all the
;; operators that occur in the protocol [operators give us probes].

(sp po*imagine*operators          ; itt: 1; ipref: n (x4) f?:y
  (state <s> ^superstate.operator.goal postpone)
  -->
  (<s> ^operator <o1> <o2> <o3> <o4>)
  (<o1> ^name imagine ^terminate-and-reject t
    ^operator* create-referent
    ^trace operator* + &, create-referent + &)
  (<o2> ^name imagine ^terminate-and-reject t

```

```

^operator* return-operator
^trace operator* + &, return-operator + &)
(<o3> ^name imagine ^terminate-and-reject t
  ^operator* add-property
  ^trace operator* + &, add-property + &)
(<o4> ^name imagine ^terminate-and-reject t
  ^operator* exhausted
  ^trace operator* + &, exhausted + &))

;;; -----
;;; t322 - "what space am i in, i'm in s-construct"

;; 10-16-95 - when the current context is displayed, see what it is
;; [cf :proposal-context].

(sp po*fixate*current-context    ; ftt: 3; fpref: n
  (state <s> ^superstate.operator
    ^wm (- ^current-context)
    ^wm.dp.<reg>.context
    (<current> ^newer none ^problem-space <space>))
  -->
  (<s> ^operator <o>)
  (<o> ^name fixate ^region <reg> ^terminate-and-reject t
    ^current-context <space>
    ^trace :current-context + &, <space> + &))

;;; -----
;;; t306 - "this is just the bare node, it doesn't have any of the
;;; properties on it"

;; 2-18-96 - "^properties" is an attribute of a u-model subobject,
;; not the u-model itself ("u-model.referent.properties"). fixate
;; on absence of the intermediate subobject ("referent").

;; 2-27-96 - <u20> can be other than a u-model (eg, o29). only
;; recognize u20 as a u-model in f:u-model, which as a
;; pseudo-chunk, by our rules, can't have a variable to the
;; u-model's current id. would have to imagine that u20 is a
;; u-model, when activated by the things that now activate
;; f:u-model.

(sp po*fixate*no-referent        ; ftt: 3; fpref: n (2-18-96)
  (state <s> ^superstate.operator
    ^wm.object u-model
    ^wm.attribute-of-object <u20>
    ^wm (- ^referent nil)
    ^wm.dp.<reg> (^object-id <u20> - ^referent))
  -->
  (<s> ^operator <o>)
  (<o> ^name fixate ^region <reg> ^terminate-and-reject t
    ^attribute referent
    ^referent nil
    ^trace :no-referent))

;;; -----

```

```

;;; t362 - "where was i, that was the match set"

;; 10-17-95 - locate ourselves in the run/print loop by looking to
;; see what happened last. two things to look for are chunk builds
;; and asserted sps (former follows latter).

(sp po*fixate*where-was-i*chunks-built ; ftt: 2; fpref: b
 (state <s> ^goal where-was-i
  ^wm (- ^builds )
  ^wm.dp.<reg>.builds <bn>)
-->
(<s> ^operator <o> + >)
(<o> ^name fixate ^region <reg> ^terminate-and-reject t
 ^builds <bn>
 ^trace <bn> + &, where-was-i:chunks + &))

(sp po*fixate*where-was-i*assertions ; ftt: 2; fpref: b
 (state <s> ^goal where-was-i
  ^wm (- ^match-set )
  ^wm.dp.<reg>.match-set <ms>)
-->
(<s> ^operator <o> + >)
(<o> ^name fixate ^region <reg> ^terminate-and-reject t
 ^match-set <ms>
 ^trace <ms> + &, where-was-i:assertions + &))

;;; -----
;;; t366 - "ok ... let's see what what the chunks are doing"

;; 10-17-95 - look at chunk builds. [overlaps with
;; where-was-i:chunks, but that's proposed best.]

(sp po*fixate*builds ; ftt: 3; fpref: n
 (state <s> ^superstate.operator
  - ^wm.builds <builds-364>
  ^wm.dp.<reg>.builds <builds-364>)
-->
(<s> ^operator <o>)
(<o> ^name fixate ^region <reg> ^terminate-and-reject t
 ^builds <builds-364>
 ^trace builds + &, <builds-364> + &, g:builds + &))

;; 3-26-95 - look at a chunk within the build set. need to care
;; about assertions before we care about chunks they might have
;; built [converging evidence].

(sp po*fixate*chunk ; ftt: 3; fpref: n
 (state <s> ^superstate.operator
  ^wm.match-set <ms>
  ^wm.builds <builds>
  - ^wm.chunk <chunk>
  ^wm.dp.<reg> (^builds <builds>
  ^build <chunk>))
-->
(<s> ^operator <o>))

```

```

(<o> ^name fixate ^region <reg> ^spatial <chunk> ^terminate-and-reject t
 ^chunk <chunk> + &
 ^trace :chunk + &, <chunk> + &))

;;; -----
;;; t403 - "oh it's not testing for a problem space that's why ok"

;; 9-9-95 - if we're thinking about the lhs, note if a problem-space
;; test is missing. [current-context:goal-stack ::
;; problem-space:conditions]

(sp po*fixate*no-problem-space ; ftt: 2; fpref: n
 (state <s> ^goal lhs
  ^wm (- ^condition problem-space)
  ^wm.dp.<reg> (^chunk <chunk-128>
  - ^condition.problem-space))
-->
(<s> ^operator <o>))
(<o> ^name fixate ^region <reg> ^terminate-and-reject t
 ^problem-space nil
 ^condition problem-space
 ^trace :no-space))

;;; -----
;;; t416 - "so it [apply-create-referent] must have just changed it
;;; on, on the superstate?"

;; 10-17-95 - when there's a chunk but we don't know what context was
;; modified, imagine possible contexts that could have been modified.

(sp po*imagine*operator-targets ; itt: 2; ipref: i (x2) f?:n
 (state <s> ^goal operator*
  ^wm.chunk <chunk>
  ^wm.problem-space nil)
-->
(<s> ^operator <o1> <o2>)
(<o1> ^name imagine ^terminate-and-reject t
 ^target superstate + &
 ^interleave-best t
 ^trace target + &, superstate + &)
(<o2> ^name imagine ^terminate-and-reject t
 ^target top-context + &
 ^interleave-best t
 ^trace target + &, top-context + &))

;;; -----
;;; t422 - "oh these are shared states, i see"

;; 8-31-95 - prefers :superstate-id at this point. [a reasonable
;; selection rule, but very specific.]
;; 12-21-95 - ["superstate" -> "superstate"] fails.]

(sp p*fixate*superstate
 (state <s> ^operator <o> +
  ^superstate.operator.goal superstate))

```

```

(<o> ^name fixate ^superstate)
-->
(<s> ^operator <o> >))

;; 9-1-95 - when we know the superstate id, look to see if it's also
;; the id of the current state.

(sp po*fixate*shared-state ; ftt: 2; fpref: b
 (state <s> ^goal superstate
  ^wm.superstate <s15>
  ^wm (- ^state shared-state)
  ^wm.dp.<reg>.context (^newer none ^state <s15>)
  ^superstate.operator)
-->
(<s> ^operator <o> + >)
(<o> ^name fixate ^region <reg> ^terminate-and-reject t
 ^state shared-state
 ^trace state + &, :shared-state + &))

;;; -----
;;; t427 - "no i don't see, what the hell built that {^[v]}"

;; 10-17-95 - when thinking about the superstate, imagine what sps
;; might modify it were they to be asserted. [this is the same set
;; as f:apply-sps.]

(sp po*imagine*assertions ; itt: 1; ipref: i (x4) f?:y
 (state <s> ^superstate.operator.goal superstate)
-->
(<s> ^operator <o1> <o2> <o3> <o4>)
(<o1> ^name imagine ^terminate-and-reject t
 ^assertion apply-create-referent
 ^sp apply-create-referent
 ^interleave-best t
 ^trace apply-create-referent + &)
(<o2> ^name imagine ^terminate-and-reject t
 ^assertion implement-exhausted
 ^sp implement-exhausted
 ^interleave-best t
 ^trace implement-exhausted + &)
(<o3> ^name imagine ^terminate-and-reject t
 ^assertion apply-add-property
 ^sp apply-add-property
 ^interleave-best t
 ^trace apply-add-property + &)
(<o4> ^name imagine ^terminate-and-reject t
 ^assertion apply-return-operator
 ^sp apply-return-operator
 ^interleave-best t
 ^trace apply-return-operator + &))

;;; -----
;;; t435 - "this said if you're in the s construct problem space, you
;;; slap ... that attribute on the {^V} object"

```

```

;; 10-17-95 - if there's a shared state, and an asserted sp, and
;; that sp is on the display, check what space it tests. that
;; context contains the shared state.

;; 9-18-95 - [is this the inference we scrolled back back for?]

(sp po*fixate*current-context*shared-state ; ftt: 2; fpref: n
 (state <s> ^superstate.operator
  ^goal <apply-create-referent>
  ^wm.state shared-state
  ^wm (- ^<s-construct> shared-state)
  ^wm.dp.<reg> (^sp <apply-create-referent>
    ^condition.problem-space <s-construct>
    - ^condition <s-construct>) ; not the id values
  )
-->
(<s> ^operator <o> >)
(<o> ^name fixate ^region <reg> ^terminate-and-reject t
 ^current-context <s-construct>
 ^<s-construct> shared-state ; [9-18-95 - inhibits s-construct goal]
 ^trace :current-context-ss + &, <s-construct> + &))

;;; -----
;;; t497 - "i now have it sitting there {^[>]} and the second chunk i
;;; know is just going to be testing for {p__chunk-} um {129} the bead
;;; right yeah there's the conjunct symbol, fine"

;; 1-13-96 - the chunk we now have in WM (3) is associated with the
;; current goal (1, 2), so look at a second chunk if there is one
;; [a representation of the goal (shared-state) being done.]

(sp po*fixate*chunk*second ; ftt: 1; fpref: b
 (state <s> ^superstate.operator.goal <shared-state> ; 1
  ^wm.<shared-state> <chunk1> ; 2
  ^wm (^chunk <chunk1> - ^chunk <chunk2>) ; 3
  ^wm.dp.<reg> (^builds <builds>
    ^build <chunk2>))
-->
(<s> ^operator <o> + >)
(<o> ^name fixate ^region <reg> ^terminate-and-reject t
 ^chunk <chunk2>
 ^trace :second + &, <chunk2> + &))

;;; -----
;;; t492 - "alright rick which means i now have, i now have it sitting
;;; there .. and the second chunk i know is just going to be testing
;;; for, um, the bead right yeah there's the conjunct symbol fine"

;; 12-23-95 - when an action creates an object with substructure,
;; imagine that the action creates the substructure as well. [this
;; reflects the work it takes to follow links to infer what an
;; action really does.]

(sp po*imagine*action ; itt: 2; ipref: b f?:y
 (state <s> ^goal rhs

```

```

^wm.action <ul>
^wm.<ul> <referent>
^operator.om <om>

-->
(<s> ^operator <o> + >)
(<o> ^name imagine ^terminate-and-reject t
 ^action <referent>
 ^trace action + &, <referent> + &))

;;; -----
;;; t529 - "why are the chunks retracting"

;; 9-11-95 - when chunks we know of are retracting, take note.

(sp po*fixate*chunks-retracting ; ftt: 3; fpref: n
 (state <s> ^superstate.operator
 ; 10-17-95 - don't move this!
 ^wm.dp.<reg>.retractions <chunk-128>
 ^wm.chunk <chunk-128>
 ^wm (- ^retraction <chunk-128>)
 )
-->
(<s> ^operator <o>)
(<o> ^name fixate ^region <reg> ^spatial <chunk-128> ^terminate-and-reject t
 ^retraction <chunk-128>
 ^trace retraction + &, <chunk-128> + &))

;; 9-11-95 - the referent action negates the no-referent condition.
;; imagine that's what causes the retraction.

(sp po*imagine*actions-refract ; itt: 3; ipref: b f?:n
 (state <s> ^superstate.operator
 ^wm.retraction <chunk-128>
 ^wm.action <obj>
 ^wm.<obj> <referent>
 ^wm.condition <referent>
 ^wm.<referent> nil)
-->
(<s> ^operator <o> + >)
(<o> ^name imagine ^terminate-and-reject t
 ^<referent> retraction
 ^trace <referent> + &, retraction + &))

;;; -----
;;; t554 - "hm! why didn't it build a chunk"

;; 12-18-95 - when we just issued a command, and something is
;; modifying the superstate, note the absence of chunk builds in the
;; newest region. [represents a failed expectation.]

(sp po*fixate*no-chunk ; ftt: 3; fpref: b
 (state <s> ^superstate.operator
 ^wm.target superstate
 ^wm.<add-property> target
 ^wm (- ^chunk nil)

```

```

^wm.dp.newest <reg>
^wm.dp.<reg> (- ^builds)
)

-->
(<s> ^operator <o> + >)
(<o> ^name fixate ^region <reg> ^terminate-and-reject t
 ^abstract-chunk nil
 ^builds nil
 ^chunk nil
 ^trace :no-chunk + &))

;;; -----
;;; t571 - "ok so this is the thing that's putting animate true on"

;; 10-17-95 - look at a selected operator. (:selected-id looks at the id.)
;; 2-2-96 - [register that it's an operator that is selected.
;; bind-op*id and bind-op*space register ^operator*, and other
;; things. selection isn't their concern.]

(sp po*fixate*selected-operator ; ftt: 3; fpref: n
 (state <s> ^wm (- ^selected <add-property>)
 ^superstate.operator
 ^wm.dp.<reg>.selected <add-property>)
-->
(<s> ^operator <o>)
(<o> ^name fixate ^region <reg> ^terminate-and-reject t
 ^selected <add-property>
 ^operator* <add-property> ; 2-2-96
 ^trace <add-property> + &, :selected + &))

;; 10-17-95 - if an operator's selected, look at its argument.

(sp po*fixate*argument ; ftt: 2; fpref: b
 (state <s> ^superstate.operator
 ^goal <add-property> ; 10-24-95 - ftt 2
 ^wm.selected <add-property>
 ^wm (- ^argument)
 ^wm.dp.<reg> (^selected <add-property> ^argument <t>))
-->
(<s> ^operator <o> + >)
(<o> ^name fixate ^region <reg> ^terminate-and-reject t
 ^<add-property> <t>
 ^argument <t>
 ^trace <add-property> + &, :argument + &, <t> + &))

;;; -----
;;; t614 - "fine so it's going to add the next one"

;; 1-13-96 - look at a property that's not the one in wm [we know
;; about "two properties"; cf po*fixate*two-valued-attribute]

(sp po*fixate*previous-argument ; ftt: 3; fpref: b (1-13-96)
 (state <s> ^superstate.operator
 ^wm.argument <policeman>
 ^wm (- ^argument <t>)

```

```

    ^wm.dp.<reg>.argument <t>)
-->
(<s> ^operator <o> + >)
(<o> ^name fixate ^region <reg> ^terminate-and-reject t
  ^argument <t>
  ^trace <t> + &, :previous-arg + &))

;;; -----
;;; t580 - "and then i should get reconsiders, yup"

;; 10-17-95 - if we're thinking about an sp, and finally got a real
;; chunk build, suppose that the sp caused the chunk build.

(sp po*imagine*sp-causes-builds      ; itt: 1; ipref: b f?:n
  (state <s> ^superstate.operator.goal <sp>
    ^wm.<sp> target      ; sp modifies target
    ^wm.builds <> nil)
  -->
  (<s> ^operator <o> + >)
  (<o> ^name imagine ^terminate-and-reject t
    ^<sp> builds + &
    ^trace <sp> + &, builds + &))

;;; -----
;;; t648 - "{^[v] u-something {^[v] u20, let's look at u-20
;;; {^[>p_u20^M}"

;; 2-18-96 - the referent is a central program object. have we seen
;; the referent -- or seen :no-referent? if so, then we've seen the
;; encompassing u-model, so we can go find its id.

(sp po*imagine*referent              ; itt: 1; ipref: b f?:y
  (state <s> ^superstate.operator.goal u-model
    ^wm.object u-model)
  -->
  (<s> ^operator <o> + >)
  (<o> ^name imagine ^terminate-and-reject t
    ^attribute referent + &
    ^u-model referent + &
    ^trace :referent))

;; 9-18-95 - if we care about the current context, look at the state,
;; since we might want to print something with the id.

(sp po*fixate*state-id                ; ftt: 3; fpref: n
  (state <s> ^superstate.operator
    ^wm.current-context <s-construct>
    ^wm (- ^state)
    ^wm.dp.<reg>.context (^newer none ^state <s15>))
  -->
  (<s> ^operator <o>)
  (<o> ^name fixate ^region <reg> ^terminate-and-reject t
    ^state <s15>
    ^trace state + &, <s15> + &, :state-id + &))

```

```

;; 9-20-95 - look at the object whose attribute we imagined seeing.

;; 2-18-96 - when there's an imagined-but-seen attribute, fixate on
;; object identifiers. [we'd prefer to fixate on an object that's
;; related to the imagined attribute -- specifically, the object
;; that contains the attribute. however, the attribute in this case
;; ("referent") didn't exist yet; the model saw it's absence
;; (:no-referent). the programmer is probably guided by the lexical
;; relationship between "u-model" and "u20"; see
;; p*fixate*u-something.]

(sp po*fixate*id-of-imagined-object    ; ftt: 2; fpref: b
  (state <s> ^goal <u-model>
    ^wm.<u-model> <referent>
    ^wm.imagined-but-seen <referent>
    ^wm.attribute <referent>
    - ^wm.dp.newest      ; 5-9-96 - we've scrolled
    ^wm (- ^object-id <u20>)
    ^wm.dp.<reg>.object-id <u20>)
  -->
  (<s> ^operator <o> + >)
  (<o> ^name fixate ^region <reg> ^terminate-and-reject t
    ^object-id <u20> + &
    ^<u-model> <u20> + &
    ^trace <u20> + &, :id-of-imagined-att + &))

;; 2-18-96 - the model doesn't represent the fine-grain lexical
;; structure that that would let it infer "u" from "u-model".
;; programmer seems to make this kind of inference only once, so
;; represent it as a special case. (note that she could also have
;; encoded the "u" prefix when it first occurred, again because it
;; was somewhat meaningful.

(sp p*fixate*u-something
  (state <s> ^superstate.operator.goal u-model
    ^operator <o1> + <o2> +)
  (<o1> ^name fixate ^object-id u20)
  (<o2> ^name fixate ^object-id <> u20)
  -->
  (<s> ^operator <o1> > <o2>))

;;; -----
;;; t652 - "right which has referent r9"

;; 10-18-95 - look at the augmentations of an object. eg,
;; (u20 ^referent r9)

(sp po*fixate*augmentation            ; ftt: 3; fpref: n
  (state <s> ^superstate.operator
    ^wm.dp.<reg>
    (^object-id <u20> ^id <r9> ^<referent> <r9>
    - ^meta-attribute <referent>)
    ^wm (- ^id <r9>)      ; 8-14-95 - refract
    )
  -->
  (<s> ^operator <o> + >))

```

```

(<s> ^operator <o>)
(<o> ^name fixate ^region <reg> ^terminate-and-reject t
 ^<referent> <r9>
 ^attribute <referent>           ; 3-24-95 - see att-of-id probe
 ^id <r9>
 ^object-id <r9>                 ; 10-18-95 - infer it's an object
 ^trace <referent> + &, <r9> + &, :augmentation + &
 ;; 10-18-95 - accumulate info about the augmented object
 ^attribute-of-object <u20> + &
 ^<u20> <r9> + &)           ; 8-29-95 - for probes in next goal
)

;;; -----
;;; t655 - "it's sitting there, it has two properties"

; 9-21-95 - when we know of one, know to look for the second, and
; encode that as "two".

(sp po*fixate*two-valued-attribute      ; ftt: 3; fpref: n
 (state <s> ^superstate.operator
  ^wm.attribute <att>
  ^wm.<att> <vall>
  ^wm (- ^two <att>)
  ^wm.dp.<reg>.<att> <vall> { <> <vall> <val2> })
-->
(<s> ^operator <o>)
(<o> ^name fixate ^region <reg> ^terminate-and-reject t
 ^two <att> + &           ; cf "nil properties"!
 ^trace :two + &, <att> + &))

;;; -----
;;; t708 - "course its the same as the state"

; 10-18-95 - if the superstate is a target, look at its id (in case
; we need to print it). [target is abstract, action is a specific
; sp action; the two overlap on superstate]

(sp po*fixate*superstate-id            ; ftt: 3; fpref: n
 (state <s> ^superstate.operator
  ^wm.<< target action >> superstate
  ^wm (- ^id <s15>)
  ^wm.dp.<reg>.context (^newer none ^older.state <s15>))
-->
(<s> ^operator <o>)
(<o> ^name fixate ^region <reg> ^terminate-and-reject t
 ^superstate <s15>
 ^id <s15>                 ; 9-21-95
 ^trace superstate + &, <s15> + &, :superstate-id + &))

;;; -----
;;; t726 - "s-construction done, u-constructor applied, u-model
;;; success, this is going to let me propose the return operator"

; 10-18-95 - if we're thinking about something marked construction
; done, look at various flags that would be present when

```

```

; construction is done. [i have no idea where "^superstate
; construction-done" came from.]

(sp po*fixate*annotations              ; ftt: 2; fpref: b
 (state <s> ^goal <superstate>
  ^wm.<superstate> construction-done
  ^wm (- ^annotation construction-done)
  ^wm.dp.<reg>
  (^annotation construction-done ; 9-23-95 - convenient reduction
   ;; u-constructor-applied u-model-success
  ))
-->
(<s> ^operator <o> + >)
(<o> ^name fixate ^region <reg> ^terminate-and-reject t
 ^annotation construction-done + &
 ^trace :annotations + &))

;;; -----
;;; t730 - "this is going to let me propose the return operator --
;;; where -- create, in the create-operator space"

; 2-21-95 - infer that the space above the newest context is the
; proposal context.
; 10-16-95 - like *current-context, but tests the goal because we
; don't always care about the proposal.

(sp po*fixate*proposal-context          ; ftt: 2; fpref: b
 (state <s> ^superstate.operator
  ^goal proposal-context
  ^wm (- ^proposal-context <space>)
  ^wm.dp.<reg>.context
  (<current> ^newer none ^older <super>)
  (<super> ^problem-space <space>))
-->
(<s> ^operator <o> + >)
(<o> ^name fixate ^region <reg> ^terminate-and-reject t
 ^proposal-context <space>
 ^trace :proposal-context + &, <space> + &))

;;; -----
;;; t753 - "terminate s-model constructor, see what that's doing"

; 10-18-95 - look at an asserted sp.
; 9-25-95 - [set goals all over the place if we only test for absence
; in fixation memory.]

(sp po*fixate*assertion                ; ftt: 3; fpref: n
 (state <s> ^wm.dp.<reg>.assertions <sp>
  ^wm (- ^assertion <sp>) ; 9-25-95
  ^superstate.operator)
-->
(<s> ^operator <o>)
(<o> ^name fixate ^region <reg> ^spatial <sp> ^terminate-and-reject t
 ^assertion <sp>
 ^trace <sp> + &, :assertion + &))

```

```

;; 12-18-95 - if we know that something issues a reconsider, don't
;; bother looking at a nil assertions set, because we care more
;; about the upcoming operator selection. [makes ^assertions nil
;; work; represents part of the expectation for the upcoming
;; operator.]

(sp p*fixate*nil-assertion-worst
 (state <s> ^operator <o> +
   ^wm.<terminate-sp> reconsider)
 (<o> ^name fixate ^assertion nil)
 -->
 (<s> ^operator <o> <))

;; 2-27-96 - if we're paying attention, prefer to look at assertions.

(sp p*fixate*assertion*pay-attention
 (state <s> ^operator <o> +
   ^wm.high-level-goal pay-attention)
 (<o> ^name fixate ^assertion <a>)
 -->
 (<o> ^interleave-best t))

;;; -----
;;; t802 - "operator op"

;; 10-16-95 - if there's a variable on the rhs, see what attribute
;; binds it. eg, if the display reads:
;; (sp (goal ^operator* <op>) --> (<o> ^new-operator <op>))
;; this encodes "^bound-by operator* ^variable op" (etc).
;;
;; here the bound variable is a parameter being added to an object
;; on the rhs. in :bind-target, the bound variable is the object to
;; be modified.

(sp po*fixate*binding-attribute*param ; ftt: 3; fpref: n
 (state <s> ^superstate.operator
   ^wm (- ^bound-by <for>) ; 8-7-95
   ^wm.dp.<reg>.condition.<operator*> <op>
   ^wm.dp.<reg>.action.<new-operator> <op>)
 -->
 (<s> ^operator <o>)
 (<o> ^name fixate ^region <reg> ^terminate-and-reject t
   ^bound-by <operator*>
   ^condition <operator*>
   ^attribute <new-operator>
   ^<new-operator> <op>
   ^<op> <operator*>
   ^variable <op>
   ^trace condition + &, <operator*> + &, <op> + &, :bind-param + &
 ))

;;; -----
;;; t806 - "oh i see so it's going to put s-constructor16"

;; 12-23-95 - find out what the variable binds, using the

```

```

;; problem-space condition to index the context stack. [like
;; :bind-op-id, but takes a different perceptual path.]

(sp po*fixate*bind-op*space ; ftt: 2; fpref: b (12-23-95)
 (state <s> ^superstate.operator
   ^goal <op> ; what op are we binding?
   ^wm.variable <op>
   ^wm.problem-space <create-operator>
   ^wm (- ^operator* <s-constructor16>)
   ^wm.dp.<reg>.context (^problem-space <create-operator>
     ^operator* <s-constructor16>))
 -->
 (<s> ^operator <o> + >)
 (<o> ^name fixate ^region <reg> ^terminate-and-reject t
   ^<create-operator> <s-constructor16> + &
   ^operator* <s-constructor16> + &
   ^trace <create-operator> + &, <s-constructor16> + &,
     :bind-op-space + &))

;;; -----
;;; t814 - "there's the return operator, so far so good"

;; 3-23-95 - look at the id of a selected operator (:selected looks at
;; the name). see also the search control below.

(sp po*fixate*selected-id ; ftt: 3; fpref: n; nr
 (state <s> ^superstate.operator
   ^wm (- ^operator-id <o29>)
   ^wm.dp.<reg> (^selected <return-op> ^operator-id <o29>))
 -->
 (<s> ^operator <o>)
 (<o> ^name fixate ^region <reg> ^terminate-and-reject t
   ^operator-id <o29>
   ^id <o29>
   ^<o29> <return-op>
   ^trace <o29> + &, :selected-id + &))

;; 9-17-95 - when the goal is to comprehend the operator, look at the
;; id, because we may need it to print the operator.
;; 9-21-95 - fires usefully early on.

(sp p*fixate*operator-id
 (state <s> ^operator <o> +
   ^superstate.operator.goal operator*)
 (<o> ^name fixate ^operator-id)
 -->
 (<s> ^operator <o> >))

;;; -----
;;; t826 - "which is in fact s-constructor sixteen"

;; 10-18-95 - if we're returning an operator, and know its id, look
;; for that id in the context stack and note the operator's name.
;;

```

C.3.4. mechanism.soar

```

;; if the display reads:
;;
;;      :      ==>G: G15 state no-change
;;      :      P: P68 create-operator
;;      :      S: S15
;;      :      O: O24 s-constructor16
;;      :      ==>G: G16 operator no-change
;;
;;      [...]
;;
;;      Soar> run 1
;;      80:          O: O29 return-operator
;;
;;      Soar> p o29
;;      (O29 ^name return-operator ^new-operator O24)
;;
;; this will note that o24 is s-constructor16 and that that
;; s-constructor16 is an operator.

(sp po*fixate*bind-op*id          ; ftt: 3; fpref: b
  (state <s> ^superstate.operator
    ^wm.operator-id <o29>
    ^wm.<o29> <o24>
    ^wm (- ^<o24> <s-constructor16>)
    ^wm.dp.creg).context (^op-id <o24> ^operator* <s-constructor16>))
-->
(<s> ^operator <o> + >)
(<o> ^name fixate ^region <reg> ^terminate-and-reject t
  ^<o24> <s-constructor16> + &
  ^operator* <s-constructor16> + &
  ^trace <o24> + &, operator* + &, <s-constructor16> + &,
  :bind-op-id + &))

;;; -----
;;; t843 - "i think all those chunks i built test for operator"

;; 3-1-96 - if we're thinking about an s-model-constructor
;; condition, and there's an instance of this type of operator in
;; WM, imagine that instance being tested elsewhere. Experience
;; suggests this is possible, and also that SPs and chunks usually
;; test the operator they apply.

(sp po*imagine*s-model-constructor ; itt: 3; ipref: b f?:y
  (state <s> ^wm.condition s-model-constructor
    ^wm.operator* <s-constructor16>)
-->
(<s> ^operator <o> + >)
(<o> ^name imagine ^terminate-and-reject t
  ^condition <s-constructor16>
  ^trace condition + &, <s-constructor16> + &))

```

po*attend.....	166
ao*attend.....	166
ao*attend*previous-not-newest.....	166
ao*attend*mark-locally.....	166
a*wm*newest-from-not-newest.....	166
p*attend*old-regions*reject.....	167
po*attend*old-regions.....	167
ao*attend*old-regions.....	167
ao*display*emulator*first.....	167
ao*display*emulator.....	167
a*state*fixate-meta-attributes.....	167
ao*fixate.....	167
ao*fixate*unpack-fixation-object.....	167
ao*fixate*mark-imagined-object.....	168
ao*comprehend*unpack-fixation-object.....	168
a*fixate*dont-interleave-best.....	168
a*fixate*newest.....	168
p*fixate*interleave-best.....	168
p*fixate*newest*interleave-best.....	168
p*fixate*top-down.....	168
p*fixate*bottom-up.....	168
p*fixate*invariant-feature*dont-interleave-best.....	168
p*fixate*fixated-recently-in-view*best.....	169
ao*comprehend*remember-display-command.....	169
p*comprehend*best*when-for-last-displayed-region.....	169
p*display*dunk-comprehend.....	169
p*display*reject-duplicates.....	169
a*goal-select*proposed-during.....	169
ao*goal-select*select-now.....	169
ao*goal-select*comprehend*create-token.....	169
ao*goal-select*new-goal*fixate/imagine.....	170
ao*goal-select*new-goal*probe.....	170
ao*imagine.....	170
ao*imagine*imagined-but-seen.....	170
ao*fixated-recently.....	170
p*imagine*refract.....	170
p*imagine*worst-after-new-output.....	171
p*generic*indifferent.....	171
a*substate*create-time.....	171
a*substate*initialize-goal-set.....	171
ao*probe*goal.....	171
p*generic*reject.....	171
p*generic*terminate-and-reject.....	171
a*topstate*create-display-wm-dp-time-dummy.....	171
a*topstate*clean-up-old-comprehends-and-displays.....	171
ao*substate*count-first.....	171
ao*substate*count-second.....	171
po*probe*where-was-i.....	172
po*probe*with-previous-goal.....	172
a*state*important-objects.....	172
po*probe*with-important-object.....	172

```

po*probe*with-high-level-goal..... 172
po*probe*with-attribute..... 173
po*probe*with-part..... 173
p*probe*repeated-goal*worst..... 173
p*probe*new-important-object*best..... 173
p*probe*new-attribute*best..... 173
p*probe*non-important-after-imagine*worst..... 173
p*probe*fixated-recently*best..... 173
p*probe*fixated-recently*worst..... 173
p*probe*where-was-i*best..... 173
a*state*new-important-object*best..... 173
p*probe*retrieved-by-probe*best..... 174
p*probe*new-high-level-goal*best..... 174
p*probe*rhs-when-sp..... 174
p*probe*lhs-best..... 174
p*probe*rhs-better-when-apply-sp..... 174
ao*comprehend*create-dp-on-om..... 174
ao*comprehend*cleanup-naked-region-pointers..... 174
ao*comprehend*applied*first..... 174
ao*comprehend*applied*second..... 174
ao*comprehend*applied..... 174
a*state*applied-newer..... 175
a*wm*unpack-applied-om..... 175
a*dp*unpack-applied-om..... 175
a*subgoal*wm-pointer..... 175
a*subgoal*hold-back-wm-pointer-until-attend..... 175
ao*probe*unpack-probe-om-to-superop-om..... 175
ao*probe*unpack-probe-om-to-superop-om*fixated..... 175

```

```

;; -*- Mode: Sde -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; File           : mechanism.soar
;; Author        : Erik Altmann
;; Created On     : Sat Jun 4 18:06:50 1994
;; Last Modified By: Erik Altmann <altmann@electro.soar.cs.cmu.edu>
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; -----
;;; attend (8) - proposal (2; "po*"), subgoal selection (1; "p*"), and
;;; application (5) includes new/old region calculation.

;; 8-12-95 - channel selection: notices a new display region, then
;; puts a pointer on the display model and marks previous regions no
;; longer newest. chunks attend automatically to this region while
;; it's on display, rebuilding the display model for every goal.

(sp po*attend
 (state <s> ^superstate.display.<external-new>
 (<ptr> ^previous <external-old>)
 ^superstate.wm.dp (- ^<external-new>) ; 9-18-95
 ^time <internal-new>)
 -->
 (<s> ^operator <o> + >) ; 9-18-95

```

```

(<o> ^name attend ^type new-region
 ^terminate-and-reject t
 ^external-new <external-new>
 ^<external-new> <internal-new> ; refract proposal and chunk
 ^<internal-new> <ptr> ; access route on dp
 ^trace <external-new> + &, <internal-new> + &))

;; 7-30-95 - like fixate, but don't test goal (want the chunks to
;; fire to accumulate not-newest). each om.dp is copied with
;; i-support to wm.dp

(sp ao*attend
 (state <s> ^operator (^name attend
 ^external-new <external-new>
 ^<external-new> <internal-new>
 ^<internal-new> <ptr>)
 ^superstate.operator.om.dp
 (<dp> - ^<external-new>))
 -->
 (<dp> ^<external-new> <internal-new> + &
 ^<internal-new> <ptr> + &
 ^internal <internal-new> + &)) ; 8-25-95 - for ^newest

;; 11-5-95 - use the current internal time to look up the current
;; external region, use that to find the previous region, look that up
;; to find the internal name of the previous region, and mark that as
;; not-newest.

(sp ao*attend*previous-not-newest
 (state <s> ^time <internal-new>
 ^wm.dp.<external-new> <internal-new>
 ^superstate.display.<external-new>.previous <external-old>
 ^wm.dp.<external-old> <internal-old>
 ^superstate.operator.om.dp <dp>)
 -->
 (<dp> ^not-newest <internal-old> + &))

(sp ao*attend*mark-locally
 (state <s> ^operator.name attend
 ^operator.external-new)
 -->
 (<s> ^attended external-new)) ; 9-23-95 - po*attend*old-regions uses this

;; 8-25-95 - give direct access to the newest.

(sp a*wm*newest-from-not-newest
 (state ^superstate nil
 ^wm.dp.internal <time>
 ^wm.dp (- ^not-newest <time>)
 ^wm.dp <dp>)
 -->
 (<dp> ^newest <time> + &)) ; the two overlap

;; 9-14-95 - propose an instance of attend if there was a display
;; operator just before the current goal, but only if there's no new

```

```

;; (unnamed) external region, which means that there was an intention
;; that only manipulated the display. mark the "newest" region
;; "not-newest", building that chunk that will mark that region from
;; then on.
;; 12-19-95 - [the om.display condition was preventing chunk transfer]

(sp p*attend*old-regions*reject ; 12-19-95
  ;; 9-18-95 - just displayed something ...
  (state <s> ^superstate.operator.om (- ^display)
    ^operator <o> +)
  (<o> ^name attend ^type old-regions)
  -->
  (<s> ^operator <o> -))

(sp po*attend*old-regions
  (state <s> - ^attended ; 11-7-95 - don't select both
    ^superstate <ss>
    ^superstate.wm.dp.newest <newest>)
  ;; 9-18-95 - ... but attend chunks have recognized every region:
  - { (<ss> ^display.<external-new>)
    (<ss> ^wm.dp (- ^<external-new>)) }
  -->
  (<s> ^operator <o> + >)
  (<o> ^name attend ^terminate-and-reject t
    ^type old-regions
    ^newest <newest>
    ^trace not-newest + &, <newest> + &))

(sp ao*attend*old-regions
  (state <s> ^operator.name attend
    ^operator.type old-regions
    ^operator.newest <n>
    ^superstate.operator.om.dp <dp>)
  -->
  (<dp> ^not-newest <n> + &))

;;; -----
;;; display emulator mechanism (2) - knowledge is in display.soar

;; apply the display operator: ^time becomes ^current-display.

(sp ao*display*emulator*first
  (state <t> ^superstate nil
    ^operator.name display
    ^operator.time <dinit>
    - ^current-display)
  -->
  (<t> ^current-display <dinit>))

(sp ao*display*emulator
  (state <t> ^superstate nil
    ^operator.name display
    ^operator.time <new>
    ^current-display { <> <new> <old> })
  -->

```

```

  (<t> ^current-display <old> - <new>))

;;; -----
;;; fixate operator (13): application (5), subgoal selection (8)

;; 3-4-95 - build a fixation object in wm.
;; 3-16-95 - fixation chunks were more general than the unpacking
;; chunks. fixation objects were staying active even when the goal
;; was something else. (specifically, the generic sp method would
;; fire as the fixation object for implement-exhausted got itself
;; unpacked.) [8-10-95 - need to be able to argue that comprehend
;; apply chunks could be goal-independent; see comprehend:255.]

(sp a*state*fixate-meta-attributes
  (state <s> ^superstate.operator)
  -->
  (<s> ^fixate-meta-attribute
    name + &,
    goal + &, ; 12-6-94
    fgoal + &, ; 8-14-95
    region + &, ; 7-26-95
    newest + &, ; 8-27-95
    interleave-best + &, ; 8-27-95
    dont-interleave-best + &, ; 8-27-95
    ; local + &, ; 8-29-95
    terminate + &,
    terminate-and-reject + &,
    trace + &,
    imagined-at + &,
    spatial + & ; 9-16-95
  ))

(sp ao*fixate
  (state <s> ^operator.name fixate
    ^operator.<a> <v>
    - ^fixate-meta-attribute <a>
    ^superstate.operator.om <om>
    ^operator.region <r> ; 9-10-95
    ^superstate.operator.goal <goal> ; 3-16-95 - see above
  )
  -->
  (<om> ^fixated <f> + &)
  (<f> ^<a> <v> + & ^region <r> + &)) ; for ao*imagine

;; 3-4-95 - then unpack it.
;; 9-4-95 - hit max-chunks without this; <om> changes regularly, and
;; then the resulting chunks match in both contexts and fire again.

(sp ao*fixate*unpack-fixation-object
  (state <s> ^wm.fixated (<f> ^ { <> imagined-at <> region <> time <a> } <v>))
    ^wm (- ^<a> <v>) ; 9-4-95
    ^time <time>
    ^superstate.operator.om <om>)
  -->
  (<om> ^<a> <v> + &))

```

```

(<f> ^time <time> + &))
;; 9-10-95 - prevent "hallucination"; if this was an image, build a
;; chunk that marks it as such in the future, so that a previous image
;; doesn't look like something seen.

(sp ao*fixate*mark-imagined-object
 (state <s> ^wm.imagined <i> ^ { <> imagined-at <> region <> time <a> } <v> )
 ^wm (- ^<a> <v> )
 ^time <time> )
-->
<i> ^imagined-at <time> + &))

;; 3-22-95 - when a new comprehend arrives, refresh anything we've
;; fixated on that might have slipped out of WM.

(sp ao*comprehend*unpack-fixation-object
 (state <s> ^superstate nil
 ^operator.name comprehend
 ^wm.fixated (<f> ^ { <> imagined-at <> region <> time <a> } <v> )
 ^wm (- ^<a> <v> )
 ^operator.om <om> )
-->
<om> ^<a> <v> + &))

(sp a*fixate*dont-interleave-best
 (state <s> ^operator <o> +
 <o> ^dont-interleave-best t)
-->
<o> ^interleave-best t -))

;; 8-27-95 - mark fixates that are for elements of the newest region.
;; 9-23-95 - not-newest chunks fire in parallel; wait until there's
;; just one newest, so we don't have to reject ones that turn out not
;; to be.

(sp a*fixate*newest
 (state <s> ^operator <o> +
 ^wm.dp.newest <newest>
 ^wm.dp (- ^newest <> <newest>))
 <o> ^name << fixate imagine >> )
-->
<o> ^newest <newest> + &)) ; 8-27-95 - o-supported

;; 12-30-95 - when a fixate operator is marked "interleave-best",
;; and the previous operator wasn't, generate a best preference.
;; probe search control makes an interleaved probe on that feature
;; best. [this lets non-newest fixates creep in after newest
;; fixates, which seems ok.] [with "- ^name fixate" for <n>, a
;; string of old fixates could block out a new one.]

(sp p*fixate*interleave-best
 (state <s> ^operator <n> <o> +
 <n> - ^interleave-best - ^name attend)

```

```

<o> ^name << fixate imagine >> ^interleave-best t)
-->
<s> ^operator <o> >))

;; 7-29-95 - prefer the region that is the newest.
;; 9-1-95 - inhibit this method when others may be better. we we're
;; scrolling back instead of generating new stuff, we probably
;; scrolled there for a reason other than to look at the region that
;; happens to be newest. [9-13-95]

(sp p*fixate*newest*interleave-best
 :o-support
 (state <s> ^operator <o> +
 ^wm.dp.newest) ; 9-14-95 - there is a newest
 <o> ^name << fixate imagine >>
 ^region <time> ^newest <time> )
-->
<o> ^interleave-best t)) ; 8-27-95 - i-supported

;; 8-27-95 - prefer fixating top-down (when there's spatial information)
;; 9-25-95 - except when the supergoal says otherwise.

(sp p*fixate*top-down
 (state <s> - ^goal rhs
 ^operator <oabove> + <obelow> +
 ^wm.dp.<reg>.spatial.<above> <below> )
 <oabove> ^name fixate ^interleave-best t ^region <reg> ^spatial <above> )
 <obelow> ^name fixate ^interleave-best t ^region <reg> ^spatial <below> )
-->
<s> ^operator <oabove> > <obelow>))

(sp p*fixate*bottom-up
 (state <s> ^goal rhs
 ^operator <oabove> + <obelow> +
 ^wm.dp.<reg>.spatial.<above> <below> )
 <oabove> ^name fixate ^interleave-best t ^region <reg> ^spatial <above> )
 <obelow> ^name fixate ^interleave-best t ^region <reg> ^spatial <below> )
-->
<s> ^operator <oabove> < <obelow>))

;; 8-30-95 - ignore the state and problem-space conditions, unless
;; targets are involved.

(sp p*fixate*invariant-feature*dont-interleave-best
 (state <s> ^operator <o> +
 ^wm (- ^target))
 <o> ^name fixate
 ;; 1-13-96 - HERE: problem-space gives information, in
 ;; particular for indexing the context stack; change this in
 ;; prose:
 ^condition { <cond> << goal ;; problem-space
 state >> }
 - ^<cond> nil)
-->
<o> ^dont-interleave-best t))

```

```

;; 12-21-95 - if something we fixated-recently is back on display,
;; meaning we likely scrolled it back into view, prefer fixating on
;; it.

(sp p*fixate*fixated-recently-in-view*best
  (state <s> ^operator <o> +
    ^wm.fixated-recently <val>
    ^first-op <l>
    - ^fixate-meta-attribute <att>))
  (<o> ^name fixate ^<att> <val>)
  -->
  (<s> ^operator <o> >))

;;; -----
;;; goal selection (9): comprehend vs. display (4) and comprehend vs
;;; comprehend (5)

;; 1-21-96 - when a display operator is proposed, select it to
;; replace a comprehend operator, but only if we're not selecting
;; the same display goal twice.

;; 3-19-95 - prefer to comprehend the most recent-displayed region.
;; retract the preference as soon as we've selected this goal.
;; 3-22-95 - make the goal acceptable, so that if a goal is the
;; undoing of its conditions and they fall out of WM, we select it
;; anyway. thus the display substitutes for wm. [required for the
;; return-operator goal]
;; 5-5-95 - why don't we just prefer ^applied-newer when a display
;; command is selected? because when the display command is no longer
;; selected the preference would retract. so we need a history of
;; display commands, which is what we get by storing the display
;; command on the goal for which it was proposed.

;; 8-12-95 - remember what display command we issued after a comprehend
;; operator, so we can re-select the comprehend after the display.

(sp ao*comprehend*remember-display-command
  (state <s> ^operator.name display
    ^operator <display>
    ^applied-newer (<goal> ^om <om>)) ; previous comprehend
  -->
  (<om> ^display <display> + &))

(sp p*comprehend*best*when-for-last-displayed-region
  (state <s> ^applied-newer
    (<best> ^goal <t>)) ; retract when this changes
  (<best> ^name comprehend
    ^om.display.val <t> ; just displayed this goal
    ^goal <t>))
  -->
  (<s> ^operator <best> + >))

(sp p*display*dunk-comprehend
  (state <s> ^operator <comp> <display> +
    - ^reject <display>

```

```

    - ^applied-newer.om.display.goal <goal>)
  (<comp> ^name comprehend)
  (<display> ^name display ^goal <goal> ; - ^too-soon ; 1-28-96 - HERE
  )
  -->
  (<s> ^operator <comp> @ - <display> >))

;; 1-21-96 - some display proposals match multiple times; after one is
;; selected, reject the others.

(sp p*display*reject-duplicates
  (state <s> ^operator <o> +
    ^applied-newer.om.display.goal <goal>)
  (<o> ^name display ^goal <goal>)
  -->
  (<s> ^operator <o> -))

;; 11-17-95 - marked a proposed comprehend operator with the operator
;; it was proposed during (an indicator of recency).

(sp a*goal-select*proposed-during
  (state <s> ^superstate nil
    ^operator (<o> ^name comprehend)
    ^operator { <> <o> <p> } +)
  (<p> ^name comprehend - ^proposed-during) ; want only the first
  -->
  (<p> ^proposed-during <o>))

;; 3-20-95 - select a "select-now" goal immediately. depends only on
;; what is happening at the top level [and doesn't chunk].
;; 11-16-95 - don't apply this method when about to, or having just,
;; invested in manipulating the display and maybe guessing features.

(sp ao*goal-select*select-now ; 11-20-95 - "spm14"
  (state <s> ^operator <old> { <> <old> <new> } +
    - ^applied.goal <new-goal> ; select only a different goal
    ^wm (- ^<< fixated-recently imagined-but-seen >> ; 11-16-95
      <old-goal>))
  (<old> ^goal <old-goal>)
  (<new> ^name comprehend
    ^goal <new-goal>
    ^proposed-during <old>
    ^select-now ok)
  -->
  (<s> ^operator <old> @ - <new> >))

;; 3-27-95 - need to cripple the chunks learned from the
;; goal-selection methods below.

(sp ao*goal-select*comprehend*create-token
  (state <s> ^superstate nil
    ^operator <o> +)
  (<o> ^name comprehend - ^token)
  -->
  (<o> ^token (make-constant-symbol token)))

```

```
;; 11-15-95 - select a goal if we fixate on or imagine the goal object.
;; 9-18-95 - wait for "interleave-best" to get its chance.
```

```
(sp ao*goal-select*new-goal*fixate/imagine ; 11-19-95 - "spm11"
  (state <s> ^operator (^name << fixate imagine >>
    ^<att> <new-goal>)
    - ^fixate-meta-attribute <att>)
  (<new> ^proposed-during <old>)
  ;; begin common
  (state <s> ^superstate.operator <old> { <> <old> <new> } +
    ^superstate <ss>
    - ^superstate.applied.goal <new-goal>
    ^second-op <2>) ; 9-18-95
  (<new> ^name comprehend
    ^goal <new-goal>
    ^token <token>) ; cripple the chunk
  -->
  (<ss> ^operator <old> @ - <new> >))
  ;; end common
```

```
;; 11-16-95 - imagined-but-seen means we've invested effort in
;; searching LTM and found something, so make it harder to select a
;; new goal and abandon this effort. converging evidence can still
;; come from seeing something. this is appropriate place, given
;; that we're probably interested in the visible context of some
;; scrolled-to feature. this context should provide the impetus for
;; the next action. (note that fixated-recently, in contrast, is
;; automatic and does not reflect cognitive investment)
```

```
(sp ao*goal-select*new-goal*probe
  (state <s> ^operator (^probe t
    ^goal <new-goal>)
    ^wm (- ^imagined-but-seen) ; 11-16-95
  (<new> ^proposed-during <old>)
  ;; begin common
  (state <s> ^superstate.operator <old> { <> <old> <new> } +
    ^superstate <ss>
    - ^superstate.applied.goal <new-goal>
    ^second-op <2>)
  (<new> ^name comprehend
    ^goal <new-goal>
    ^token <token>)
  -->
  (<ss> ^operator <old> @ - <new> >))
  ;; end common
```

```
;;; -----
;;; imagine operator (5): application (3; includes recency
;;; calculation) and subgoal-selection (2)
```

```
(sp ao*imagine
  (state <s> ^operator.name imagine
    ^operator.<a> <v>
    - ^fixate-meta-attribute <a>
    ^superstate.operator.om <om>
```

```
^wm (- ^<a> <v>)
^superstate.operator.token <token>
)
```

```
-->
(<om> ^fixated <i> + & ^imagined <i> + &
  (<i> ^<a> <v> + &))
```

```
;; 9-10-95 - imagined-but-seen is a feature fixated on earlier that
;; we've now imagined and recognized. (a chunk delivers a time when
;; the feature wasn't imagined.)
```

```
(sp ao*imagine*imagined-but-seen
  (state <s> ^time <now>
    ^wm (- ^imagined-but-seen <feature>)
    ^wm.imagined
    (^ { <> imagined-at <> time <> region <att> } <feature>
    ^time <now>)
    ^wm.fixated
    (^ { <> imagined-at <> time <> region <att> } <feature>
    ^time { <> <now> <then> }
    - ^imagined-at <then>)
    ^superstate.operator.om <om>)
  -->
  (<om> ^imagined-but-seen <feature> + &))
```

```
;; 9-10-95 - fixated-recently is a feature fixated on recently that is
;; now gone from the display but still in wm. [is this the visual
;; scratchpad?]
;; 9-11-95 - the chunks buy us one extra goal of persistence, because
;; they fire when op3 is selected, as the stuff from op1 is leaving.
;; similarly, the chunks from the original fixation during op0
;; transferred when op1 was selected, just before the display that
;; dunks op1 removes the region (which the chunks test). [i think]
```

```
(sp ao*fixated-recently
  (state <s> ^wm.fixated
    (^ { <> imagined-at <> region <> time <att> } <feature>
    ^region <now-gone>)
    ^wm.dp (- ^internal <now-gone>)
    ^superstate.operator.om <om>
  )
  -->
  (<om> ^fixated-recently <feature> + &))
```

```
;; 10-23-95 - make an imagine worst if it imagines something already
;; in WM. [this replaces all negation refractions in imagines.]
```

```
(sp p*imagine*refract
  (state <s> ^operator <o> +
    ^wm.<att> <val>
    - ^fixate-meta-attribute <att>)
  (<o> ^name imagine ^<att> <val>)
  -->
  (<s> ^operator <o> -))
```

```

;; 1-28-96 - avoid imagining things when there's new output to look
;; at.

(sp p*imagine*worst-after-new-output
 (state <s> ^operator <o> +
          ^superstate.operator.om.display
          ^wm.dp.newest)
 (<o> ^name imagine)
 -->
 (<s> ^operator <o> <))

;;; -----
;;; shared (aka miscellaneous) (10)
;;; subgoal selection: default indifferent (1)
;;; timestamp comprehension goals from below (1)
;;; maintain relevant-objects set ("^goal") (2)
;;; terminate-and-reject mechanism (2)
;;; other per-goal initializations (4)
;;; halting (not loaded)

;; all operators are indifferent.

(sp p*generic*indifferent
 (state <s> ^operator <o> +)
 -->
 (<s> ^operator <o> =))

(sp a*substate*create-time
 (state <s> ^impasse no-change
          ^superstate.operator.name comprehend)
 -->
 (<s> ^time (make-constant-symbol) ; every state has it
          ^dummy d))

;; 8-8-94 - want the parent-goal elaborations to bind from the local
;; context, so we can add to it w/o chunking [still need to understand
;; this]. [8-12-95 - still used]

(sp a*substate*initialize-goal-set
 (state <s> ^superstate.operator.goal <t>)
 -->
 (<s> ^goal <t> + &))

;; 9-21-95 - note in the subgoal when one non-attend op has been
;; selected. used by goal- and subgoal-selection. ["wait for lots of
;; preferences, because newest-is-best does."]

;; consider the probe a goal we've thought about.

(sp ao*probe*goal ; adds to "relevant-object" set
 (state <s> ^superstate.operator
          ^operator.goal <t>
          ^operator.probe t)
 -->
 (<s> ^goal <t> + &))

```

```

(sp p*generic*reject
 (state <s> ^reject <o>)
 -->
 (<s> ^operator <o> -))

(sp p*generic*terminate-and-reject
 (state <s> ^operator.terminate-and-reject
          ^operator <o>)
 -->
 (<s> ^operator <o> @
          ^reject <o> + &))

;; 8-12-95 - dummy: o-support for application chunks, which augment
;; the operator and not the state. display gets i-supported
;; augmentations from the simulator. wm gets i-supported
;; augmentations from ^applied.om. dp gets them from the om of the
;; current operator.

(sp a*topstate*create-display-wm-dp-time-dummy
 (state <s> ^superstate nil)
 -->
 (<s> ^display <d>
          ^time (make-constant-symbol) ; every state has it
          ^wm <wm>
          ^dummy d)
 (<wm> ^dp <dp>))

;; 9-18-95 - keep wm garbage from swelling production memories.

(sp a*topstate*clean-up-old-comprehends-and-displays
 :o-support
 (state <s> ^superstate nil
          ^reject <o>
          - ^operator <o> +)
 -->
 (<s> ^reject <o> -))

(sp ao*substate*count-first
 (state <s> ^superstate.operator
          - ^first-op
          ^operator (<o> ^name << imagine fixate comprehend >>))
 -->
 (<s> ^first-op <o>))

(sp ao*substate*count-second
 (state <s> ^superstate.operator
          ^first-op <o>
          - ^second-op
          ^operator ({ <> <o> <p> } ^name << imagine fixate comprehend >>))
 -->
 (<s> ^second-op <p>))

;; 10-7-95 - no state-nochanges occur in correct behavior.

(sp halt*no-change-stack

```

```

; (state ^impasse no-change
;   ^attribute state)
; -->
; (write |
;State no change -- halting.|)
; (halt))

;;; -----
;;; probing (20): proposals (7), subgoal selection (13)

;; 9-13-95 - when we've done something disorienting, probe with this
;; goal: if we also set this goal at intervals, stuff will accumulate
;; on this cue (in particular, builds-364, so we have a reason to
;; comprehend chunk-128 after returning from apply-create-referent).
;; 9-14-95 - "disorientation" = a recent display command, but no
;; newest region (we would fixate on that).

(sp po*probe*where-was-i
 (state <g> ^superstate.operator.om.display
   ^wm.dp (- ^newest))
 -->
 (<g> ^operator <o>))
 (<o> ^name comprehend ^goal where-was-i ^probe t ^om <om>
   ^trace |(where-was-i)| ^terminate-and-reject t))

;; 9-25-94 - probe with any goal that comes our way, as long as we
;; haven't before.

(sp po*probe*with-previous-goal
 (state <g> ^superstate.applied-older.goal <t>
   ^wm)
   ; 9-18-95 - wait for attend
 -->
 (<g> ^operator <o>))
 (<o> ^name comprehend ^goal <t> ^probe t ^om <om>
   ^trace |(previous-goal)| ^terminate-and-reject t))

;; 3-26-95 - added condition and action, to deal with comprehending
;; chunk-128. added the best preference so that when conditions and
;; actions are fixated on, we probe with them right away. had
;; thought before about adding this best preference.
;; 5-20-95 - adding this generates a probe for "t" but a fixate comes
;; along first to dislodge o24 (cf 5-5-95).
;; 8-10-95 - added sp, so that an apply-create-referent probe would
;; tickle comprehend:261.
;; 8-24-95 - s-construct, round about 330.
;; 9-1-95 - think about the target of some modification (eg, the
;; superstate, which then becomes salient)
;; 9-3-95 - when we do have it in wm, it's relevant (eg, s-construct,
;; about which we know that it builds chunks).
;; 9-4-95 - object's, like u-models, are things we recognize as
;; significant.
;; 9-6-95 - as represented in a context-stack; as opposed to
;; problem space, which is a condition.
;; 9-10-95 - probe with things we just saw, to import stuff we learned
;; then; lets us go back and forth and learn things. [removed,

```

```

;; because fixated-recently is hard to get rid of, and ^state is an
;; important-object.]
;; 12-2-95 - works with or without these; problem-space changes the trace.

(sp a*state*important-objects ; 6-4-96 - last count: 15 important objects
 (state <s> ^superstate.operator)
 -->
 (<s> ^important-object ; 11-9-95 - don't rearrange these!
   operator* + &,
   state + &,
   superstate + &,
   fixated-recently + &, ; 9-10-95
   imagined-but-seen + &, ; 12-2-95
   impasse* + &,
   ; ; problem-space + &, ; 9-3-95; 12-2-95
   current-context + &, ; 9-6-95 - -> prob. space if it works
   target + &, ; 9-1-95
   object + &, ; 9-4-95
   assertion + &,
   retraction + &,
   condition + &, ; 3-26-95
   action + &, ; 3-26-95
   chunk + &, ; 5-20-95
   sp + &, ; 8-10-95
   ; ; imagined-but-seen + & ; 9-19-95
   ; ; high-level-goal ; 8-25-95 - now separate
   ; ; current-context ; 8-24-95
   ; ; augments-superobject ; 4-17-95
 ))

(sp po*probe*with-important-object
 (state <g> ^important-object <object>
   ^wm.<object> <probe>
   ^superstate.operator)
 -->
 (<g> ^operator <o> +)
 (<o> ^name comprehend ^goal <probe> ^probe t ^om <om>
   ^important-object <object>
   ^trace :imp-obj + &, <object> + &
   ^terminate-and-reject t))

;; 8-29-95 - the more cues for the high-level goal, the more likely we
;; are to probe with it (hence select it).

(sp po*probe*with-high-level-goal
 (state <g> ^wm.high-level-goal <goal>
   ^wm.<goal> <cue>
   ^superstate.operator)
 -->
 (<g> ^operator <o> +)
 (<o> ^name comprehend ^goal <goal> ^probe t ^om <om>
   ^high-level-goal t
   ^trace |(high-level-goal)| ^terminate-and-reject t))

;; 3-3-95 - if a value is an id, probe with the attribute that points

```

```

;; to it. [p*fixate*augmentation*worst takes care of limiting how
;; many attributes make it into WM.]

(sp po*probe*with-attribute
  (state <g> ^superstate.operator
    ^wm.attribute <att>
    ^wm.id <id>
    ^wm.<att> <id>)
  -->
  (<g> ^operator <o>)
  (<o> ^name comprehend ^goal <att> ^probe t ^om <om>
    ^trace :att-of-id
    ^terminate-and-reject t))

;; 2-8-95 - to get lhs and rhs below propose-return-operator (779?)

(sp po*probe*with-part
  (state <g> ^wm.part <p>
    ^superstate.operator)
  -->
  (<g> ^operator <o>)
  (<o> ^name comprehend ^goal <p> ^probe t ^om <om>
    ^trace :part ^terminate-and-reject t))

(sp p*probe*repeated-goal*worst
  (state <s> ^superstate.operator
    ^operator <o> +
    ^goal <goal>)
  (<o> ^probe t ^goal <goal>)
  -->
  (<s> ^operator <o> < >))

;; 12-11-95 - probe with an important object after fixating on it.

(sp p*probe*new-important-object*best
  (state <s> ^operator <n> <o> +)
  (<n> ^name << fixate imagine >> ^<att> <val>)
  (<o> ^probe t ^important-object <att> ^goal <val>)
  -->
  (<s> ^operator <o> >))

;; 12-11-95 - think about a new attribute retrieved from ltm or display.

(sp p*probe*new-attribute*best
  (state <s> ^operator <n> <o> +)
  (<n> ^name << fixate imagine >> ^<att> <val>)
  (<o> ^probe t ^goal <att>)
  -->
  (<s> ^operator <o> >))

;; 9-21-95 - in general we want to think about something after
;; imagining it.

(sp p*probe*non-important-after-imagine*worst
  (state <s> ^operator <n> <o> +)

```

```

  (<n> ^name imagine)
  (<o> ^probe t - ^important-object)
  -->
  (<s> ^operator <o> < >))

;; 9-10-95 - if something's still hanging around in WM, probe with it,
;; to see if relevant things from when we fixated on it are relevant
;; now. [we sometimes alternate rapidly between displays, carrying
;; diffs in our heads.]

(sp p*probe*fixated-recently*best
  (state <s> ^operator <o> +
    ^wm.fixated-recently <feature>
    ^wm.dp (- ^newest))
  (<o> ^probe t ^goal <feature>)
  -->
  (<s> ^operator <o> >))

;; 9-22-95 - generally don't care about what's fixated-recently,
;; though do the calculation (elsewhere) in case we do.

(sp p*probe*fixated-recently*worst
  (state <s> ^operator <o> +
    ^wm.fixated-recently <feature>
    ^wm.dp.newest)
  (<o> ^probe t ^goal <feature>)
  -->
  (<s> ^operator <o> < >))

(sp p*probe*where-was-i*best
  (state <s> ^operator <o> +
    ^wm.dp (- ^newest))
  (<o> ^goal where-was-i ^probe t)
  -->
  (<s> ^operator <o> >))

;; 9-4-95 - probe with an important object right after we recall it,
;; so we can have an inference chain, and also so that we can get to
;; use more generic goals and avoid the tyranny of immediate ones.
;; 9-23-95 - see below (cpu time: 99.710 seconds)

;(sp p*probe*retrieved-by-probe*best
;  (state <s> ^operator <n> <o> +
;    ^important-object <important-obj>)
;  (<n> ^probe t ^goal <old> ^om.<important-obj> <new>)
;  (<o> ^probe t ^goal { <> <old> <new> })
;  -->
;  (<s> ^operator <o> >))

;; 9-23-95 - same as above, but w/o quadratic (cpu time: 91.260 seconds)

(sp a*state*new-important-object*best
  (state <s> ^operator <n>
    ^important-object <important-obj>)

```

```

(<n> ^probe t ^goal <old>
 ^om.<important-obj> { <> <old> <new>}}
-->
(<n> ^good-next-goal <new> + &))

(sp p*probe*retrieved-by-probe*best
 (state <s> ^operator.good-next-goal <new>
 ^operator <o> +)
 (<o> ^probe t ^goal <new>)
-->
(<s> ^operator <o> >))

;; 9-18-95 - same as above, for hlg.

(sp p*probe*new-high-level-goal*best
 (state <s> ^superstate.operator
 ^operator <n> <o> +)
 (<n> ^probe t ^goal <old> ^om.high-level-goal <new>)
 (<o> ^probe t ^high-level-goal t ^goal { <> <old> <new> })
-->
(<s> ^operator <o> >))

;; 10-11-95 - search control that governs lhs and rhs probes.
;; [removing any breaks the trace.]

;; 3-21-95 - when thinking about an sp, prefer probing to see what we
;; know about its parts.

(sp p*probe*rhs-when-sp
 (state <s> ^operator <o> +
 ^superstate.operator.goal <sp>
 ^wm.sp <sp>
 ^second-op <2>)
 (<o> ^probe t ^goal rhs)
-->
(<s> ^operator <o> >))

;; 10-11-95 - in general, think about the lhs first (top-down).

(sp p*probe*lhs-best
 (state <s> ^operator <o> +)
 (<o> ^probe t ^goal lhs)
-->
(<s> ^operator <o> >))

;; 3-6-95 - but when it's an apply sp, first think about its actions.

(sp p*probe*rhs-better-when-apply-sp
 (state <s> ^operator <o1> + <o2> +
 ^goal <implement-exhausted>
 ^wm.apply-sp <implement-exhausted>)
 (<o1> ^goal lhs)
 (<o2> ^goal rhs)
-->
(<s> ^operator <o1> < <o2>))

```

```

;;; -----
;;; working memory (11) (includes ^applied goals)

;; 8-12-95 - put a display-model stub on the operator memory. attend
;; chunks rebuild this dp for every goal.

(sp ao*comprehend*create-dp-on-om
 (state <s> ^superstate nil
 ^operator <o>)
 (<o> ^name comprehend ^om (<om> - ^dp))
-->
(<om> ^dp <dp>))

;; 9-10-95 - remove region pointers whose regions have disappeared,
;; say when this goal brackets a scroll command. this makes the model
;; forget immediately about the disappeared region, remembering only
;; what it's ^fixated-recently. [ao*fixated-recently relies on this;
;; there's no usable representation of "^gone-but-recent-region".]

(sp ao*comprehend*cleanup-naked-region-pointers
 (state <s> ^superstate nil
 ^operator.name comprehend
 ^applied.om.dp
 (<dp> ^internal <internal>
 ^<internal> (<naked-id> - ^<att>)
 ^<external> <internal>))
-->
(<dp> ^internal <internal> -
 ^<internal> <naked-id> -
 ^<external> <internal> -))

;; working memory consists of current and previous operator memories.

(sp ao*comprehend*applied*first
 (state <s> ^operator.name comprehend
 ^operator <o2> ; add to applied
 - ^applied
 )
-->
(<s> ^applied <o2> + &))

(sp ao*comprehend*applied*second
 (state <s> ^operator.name comprehend
 ^operator <o2> ; add to applied
 ^applied <o1>
 - ^applied <o2> ; refract
 - ^applied-older
 )
-->
(<s> ^applied <o2> + & ^applied-older <o1>))

(sp ao*comprehend*applied
 (state <s> ^operator.name comprehend
 ^operator <o2> ; add to applied
 ^applied-newer { <> <o2> <o1> } ; add only if different

```

```

    ^applied-older <o0>
  - ^applied <o2>          ; refract
    )
-->
(<s> ^applied <o2> + & ^applied-older <o1>
 ^applied <o0> - ^applied-older <o0> -))

;; 3-22-95 - if the applied-older gets reselected, the other one is
;; older (ao*comprehend*applied won't fire). [this is still broken;
;; it should test the goals, and not the operator ids.]
;; 10-2-95 - broken, but in principle could need it.

;; 3-22-95 - ^applied-newer is the current comprehend.

(sp a*state*applied-newer
 (state <s> ^applied <newer>
 - ^applied-older <newer>)
-->
(<s> ^applied-newer <newer>))

;; 2-25-95 - wm consists of the operator memories of the operators
;; on ^applied (current and previous). unpack the display model
;; separately.

(sp a*wm*unpack-applied-om
 (state <s> ^superstate nil ; don't write wm in subgoal
 ^applied.om.{ <> dp <att> } <val>
 ^wm <wm>)
-->
(<wm> ^<att> <val> + &))

(sp a*dp*unpack-applied-om
 (state <s> ^superstate nil ; don't write wm in subgoal
 ^applied.om.dp.<att> <val>
 ^wm.dp <dp>)
-->
(<dp> ^<att> <val> + &))

;; 2-24-95 - copy wm to the subgoal. wm is read only in the subgoal.
;; 7-26-95 - want attend chunks to be independent of the goal. [was:
;; "make sure chunks test goal"]

(sp a*subgoal*wm-pointer
 (state <s> ^superstate.dummy          ; o-support for chunks
 ^superstate.wm <wm>)
-->
(<s> ^wm <wm>))

;; 9-18-95 - don't copy the pointer down until any new regions have
;; been attended to. [get's rid of expensive search control for
;; attend, because nothing else is proposed before attend applies.]

(sp a*subgoal*hold-back-wm-pointer-until-attend
 (state <s> ^superstate.display.<external-new>
 ^superstate.wm.dp (- ^<external-new>) ; 9-18-95

```

```

    ^superstate.wm <wm>)
-->
(<s> ^wm <wm> -))

;; 2-24-95 - unpack probe results to om. this connects probe results
;; to the current goal. they show up in wm.
;; 3-4-95 - unpack only augmentations that aren't already there.
;; the effect is that now probing with the previous goal can't
;; re-add things that are already there. this saves in some cases
;; lots of chunks, but also means that probing with the previous
;; goal is a different kind rehearsal (need to think about this).
;; the chunks represent knowledge indexed by pairs of goals; big
;; payoffs would greet goals selected in the right pairwise order.
;; 8-12-95 - the first sp is unpacks top-level wm structure. the
;; second unpacks the fixated subobject. there's no dp on probes,
;; because we can't recall things to the display model (we can only
;; imagine things to the fixated subobject).
;; 9-19-95 - removing this may have destructively increased generality.

(sp ao*probe*unpack-probe-om-to-superop-om
 (state <s> ^operator.name comprehend
 ^operator.probe t
 ^operator.om.{ <> fixated <> dp <att> } <val>
 ^wm (- ^<att> <val>)          ; 3-4-95 - see above
 ; 8-12-95 - commented out:
 ^superstate.operator.goal <goal> ; chunking; need this? 9-19-95
 ^superstate.operator.om <om>)
-->
(<om> ^<att> <val> + &))

(sp ao*probe*unpack-probe-om-to-superop-om*fixated
 (state <s> ^operator.name comprehend
 ^operator.probe t
 ^operator.om.fixated.<att> <val>
 ^wm (- ^<att> <val>)          ; 3-4-95 - see above
 ; 8-12-95 - commented out:
 ^superstate.operator.goal <goal> ; chunking; need this?
 ^superstate.operator.om <om>)
-->
(<om> ^<att> <val> + &))

```

C.4. Index of rules

a*display*t225.....	138	ao*display*print*t770.....	154
a*display*t236.....	139	ao*display*print*t774.....	154
a*display*t245.....	139	ao*display*print*t811.....	155
a*display*t252.....	140	ao*display*print*t817.....	155
a*display*t323.....	141	ao*display*print*t845.....	156
a*display*t364.....	141	ao*display*t225.....	138
a*display*t373.....	142	ao*display*t445.....	143
a*display*t431.....	143	ao*display*t582.....	146
a*display*t445.....	143	ao*display*t593.....	147
a*display*t503.....	143	ao*display*t618.....	148
a*display*t513.....	144	ao*display*t621.....	148
a*display*t552.....	145	ao*display*tinit.....	138
a*display*t564.....	145	ao*fixate*mark-imagined-object.....	168
a*display*t575.....	146	ao*fixate*unpack-fixation-object.....	167
a*display*t582.....	146	ao*fixate.....	167
a*display*t587.....	147	ao*fixated-recently.....	170
a*display*t593.....	147	ao*goal-select*comprehend*create-token.....	169
a*display*t618.....	148	ao*goal-select*new-goal*fixate/imagine.....	170
a*display*t621.....	148	ao*goal-select*new-goal*probe.....	170
a*display*t639.....	149	ao*goal-select*select-now.....	169
a*display*t646.....	150	ao*imagine*imagined-but-seen.....	170
a*display*t649.....	150	ao*imagine.....	170
a*display*t654.....	150	ao*probe*goal.....	171
a*display*t685.....	151	ao*probe*unpack-probe-om-to-superop-om*fixed.....	175
a*display*t691.....	151	ao*probe*unpack-probe-om-to-superop-om.....	175
a*display*t720.....	152	ao*substate*count-first.....	171
a*display*t730.....	152	ao*substate*count-second.....	171
a*display*t738.....	153	d*initial*t582.....	146
a*display*t754.....	154	d*initial*t618.....	148
a*display*t770.....	154	d*pgs*t639.....	149
a*display*t774.....	154	d*print*t236.....	139
a*display*t811.....	155	d*print*t323.....	141
a*display*t817.....	155	d*print*t364.....	142
a*display*t845.....	156	d*print*t373.....	142
a*display*tinit.....	138	d*print*t503.....	144
a*dp*unpack-applied-om.....	175	d*print*t513.....	144
a*fixate*dont-interleave-best.....	168	d*print*t552.....	145
a*fixate*newest.....	168	d*print*t564.....	145
a*goal-select*proposed-during.....	169	d*print*t575.....	146
a*state*applied-newer.....	175	d*print*t587.....	147
a*state*fixate-meta-attributes.....	167	d*print*t649.....	150
a*state*important-objects.....	172	d*print*t654.....	151
a*state*new-important-object*best.....	173	d*print*t685.....	151
a*subgoal*hold-back-wm-pointer-until-attend.....	175	d*print*t691.....	151
a*subgoal*wm-pointer.....	175	d*print*t720.....	152
a*substate*create-time.....	171	d*print*t730.....	153
a*substate*initialize-goal-set.....	171	d*print*t738.....	153
a*topstate*clean-up-old-comprehends-and-displays.....	171	d*print*t754.....	154
a*topstate*create-display-wm-dp-time-dummy.....	171	d*print*t774.....	155
a*wm*newest-from-not-newest.....	166	d*print*t811.....	155
a*wm*unpack-applied-om.....	175	d*print*t817.....	156
ao*attend*mark-locally.....	166	d*print*tinit.....	138
ao*attend*old-regions.....	167	d*print-id*t245.....	140
ao*attend*previous-not-newest.....	166	d*print-id*t252.....	140
ao*attend.....	166	d*run-ms-run*t593.....	147
ao*comprehend*applied*first.....	174	d*run-ms-run*t621.....	149
ao*comprehend*applied*second.....	174	d*tinit*t225.....	138
ao*comprehend*applied.....	174	f:abstract-sp.....	131
ao*comprehend*cleanup-naked-region-pointers.....	174	f:add-property-target.....	132
ao*comprehend*create-dp-on-om.....	174	f:apply-add-property-target.....	132
ao*comprehend*remember-display-command.....	169	f:apply-sps.....	128
ao*comprehend*unpack-fixation-object.....	168	f:exhausted-builds-proposal.....	133
ao*display*emulator*first.....	167	f:high-level-goal-cues.....	129
ao*display*emulator.....	167	f:nil-chunk-is-abstract.....	132
ao*display*po*init.....	137	f:operator-condition*lhs-means-rhs.....	131
ao*display*print*t236.....	139	f:operator-condition*part-of-lhs.....	131
ao*display*print*t245.....	140	f:operator.....	133
ao*display*print*t252.....	140	f:pay-attention-when-building-proposal.....	134
ao*display*print*t323.....	141	f:postpone-return-new-pointer.....	129
ao*display*print*t364.....	142	f:proposal-build-action.....	134
ao*display*print*t373.....	142	f:propose-return-operator.....	134
ao*display*print*t431.....	143	f:recall-for-part-of-u-model.....	128
ao*display*print*t503.....	143	f:recall-u-model.....	132
ao*display*print*t513.....	144	f:recognize-u-model-object.....	128
ao*display*print*t552.....	145	f:return-operator-target.....	134
ao*display*print*t564.....	145	f:s-construct-builds-chunk.....	135
ao*display*print*t575.....	146	f:select-exhausted.....	129
ao*display*print*t587.....	147	f:shared-states-build-chunks.....	130
ao*display*print*t639.....	149	f:sp-parts.....	134
ao*display*print*t646.....	150	f:sp.....	130
ao*display*print*t649.....	150	f:sps-propose-add-property.....	132
ao*display*print*t654.....	150	f:superstate-is-shared-too.....	130
ao*display*print*t685.....	151	f:terminate-create-referent.....	131
ao*display*print*t691.....	151	f:terminating-sps.....	132
ao*display*print*t720.....	152	f:u-model.....	133
ao*display*print*t730.....	152	p*attend*old-regions*reject.....	167
ao*display*print*t738.....	153	p*comprehend*best-when-for-last-displayed-region.....	169
ao*display*print*t754.....	154	p*display*dunk-comprehend.....	169

p*display*reject-duplicates.....	169	po*display*print-operator.....	139
p*fixate*assertion*pay-attention.....	164	po*display*print-sp*applies-operator.....	139
p*fixate*bottom-up.....	168	po*display*print-sp*when-paying-attention.....	153
p*fixate*fixated-recently-in-view*best.....	169	po*display*print-stack.....	141
p*fixate*interleave-best.....	168	po*display*run*action-and-print.....	152
p*fixate*invariant-feature*dont-interleave-best.....	168	po*display*run*to-end-of-space.....	148
p*fixate*newest*interleave-best.....	168	po*display*run*to-expected-op.....	155
p*fixate*nil-assertion-worst.....	164	po*display*run*to-op-after-builds.....	147
p*fixate*operator-id.....	164	po*display*run*where-was-i.....	141
p*fixate*superstate.....	159	po*display*run-to-builds.....	145
p*fixate*top-down.....	168	po*display*scroll*to-object.....	149
p*fixate*u-something.....	162	po*display*scroll*to-sp.....	143
p*generic*indifferent.....	171	po*display*scroll*to-state.....	143
p*generic*reject.....	171	po*fixate*action.....	157
p*generic*terminate-and-reject.....	171	po*fixate*annotations.....	163
p*imagine*refract.....	170	po*fixate*argument.....	161
p*imagine*worst-after-new-output.....	171	po*fixate*assertion.....	163
p*probe*fixated-recently*best.....	173	po*fixate*augmentation.....	162
p*probe*fixated-recently*worst.....	173	po*fixate*bind-op*id.....	165
p*probe*lhs-best.....	174	po*fixate*bind-op*space.....	164
p*probe*new-attribute*best.....	173	po*fixate*binding-attribute*param.....	164
p*probe*new-high-level-goal*best.....	174	po*fixate*binding-attribute*target.....	157
p*probe*new-important-object*best.....	173	po*fixate*binding-context.....	157
p*probe*non-important-after-imagine*worst.....	173	po*fixate*builds.....	159
p*probe*repeated-goal*worst.....	173	po*fixate*chunk*second.....	160
p*probe*retrieved-by-probe*best.....	174	po*fixate*chunk.....	159
p*probe*rhs-better-when-apply-sp.....	174	po*fixate*chunks-retracting.....	161
p*probe*rhs-when-sp.....	174	po*fixate*condition.....	157
p*probe*where-was-i*best.....	173	po*fixate*current-context*shared-state.....	160
po*attend*old-regions.....	167	po*fixate*current-context.....	158
po*attend.....	166	po*fixate*id-of-imagined-object.....	162
po*comprehend*actual-context.....	129	po*fixate*no-chunk.....	161
po*comprehend*assertion*imagined.....	133	po*fixate*no-problem-space.....	159
po*comprehend*assertion*pay-attention*fixated-recently.....	135	po*fixate*no-referent.....	158
po*comprehend*assertion*pay-attention.....	134	po*fixate*previous-argument.....	161
po*comprehend*assertion*real.....	133	po*fixate*proposal-context.....	163
po*comprehend*building-agent.....	132	po*fixate*selected-id.....	164
po*comprehend*builds.....	131	po*fixate*selected-operator.....	161
po*comprehend*chunk.....	130	po*fixate*shared-state.....	160
po*comprehend*current-context-when-exhausted.....	129	po*fixate*state-id.....	162
po*comprehend*high-level-goal.....	133	po*fixate*superstate-id.....	163
po*comprehend*imagined-chunk-cause.....	130	po*fixate*two-valued-attribute.....	163
po*comprehend*init.....	128	po*fixate*where-was-i*assertions.....	159
po*comprehend*new-operator.....	135	po*fixate*where-was-i*chunks-built.....	159
po*comprehend*objects-attribute.....	135	po*imagine*action.....	160
po*comprehend*operator-condition.....	131	po*imagine*actions-refract.....	161
po*comprehend*proposal-context.....	134	po*imagine*assertions.....	160
po*comprehend*recalled-condition.....	135	po*imagine*nil-object.....	158
po*comprehend*selected-operator.....	128	po*imagine*operator-targets.....	159
po*comprehend*sp-parts.....	131	po*imagine*operators.....	158
po*comprehend*superstate-target.....	130	po*imagine*postpone.....	158
po*comprehend*superstate.....	134	po*imagine*referent.....	162
po*comprehend*u-model.....	133	po*imagine*s-model-constructor.....	165
po*comprehend*where-was-i.....	130	po*imagine*sp-causes-builds.....	162
po*display*match-set*after-builds.....	145	po*probe*where-was-i.....	172
po*display*match-set*after-selection.....	138	po*probe*with-attribute.....	173
po*display*match-set*asserted-sp.....	144	po*probe*with-high-level-goal.....	172
po*display*print*high-level-goal.....	154	po*probe*with-important-object.....	172
po*display*print-chunk.....	142	po*probe*with-part.....	173
po*display*print-object*fresh.....	150	po*probe*with-previous-goal.....	172
po*display*print-object.....	140		

Appendix D

Detailed model traces for scrolling events

This appendix maps the 5 abstract model traces from Chapter 4 to Soar operator traces and actual code. The mapping includes all elements of the abstract traces, including operators, pre-loaded rules, and learned rules. Learned rules have their names underlined.

The correspondence between model goals and subgoals and Soar operators is described in Appendix B. A more detailed trace of the entire life of the model, showing all rule firings and encodings as well as operators, appears in Appendix F. To index the operator traces here to the rule-firing trace in Appendix F, use the decision cycle (DC) that appears at the left of each operator.

D.1. Scrolling event 1 - Figure 21, p. 45

D.1.1. Encoding episode

Decision cycle	<u>Soar operator trace</u>	<u>Abstract trace from figure</u>
12	O: 026 (comprehend create-referent :selected-op).....	comprehend selected operator
13	O: 028 (display match-set:after-selection (t225)).....	print match-set
14	O: 026 (comprehend create-referent :selected-op).....	comprehend selected operator
15	==>S: S5 (operator no-change)	
16	O: 032 (attend t225 constant9)	
	(sp po*fixate*assertion.....	1. if display shows an asserted SP, propose fixating on it
	(state <s> ^wm.dp.<reg>.assertions <sp> ^wm (- ^assertion <sp>) ^superstate.operator)	
	-->	
	(<s> ^operator <o>)	
	(<o> ^name fixate ^region <reg> ^spatial <sp> ^terminate-and-reject t ^assertion <sp> ^trace <sp> + &, :assertion + &))	
17	O: 040 (fixate create-referent :argument cop).....	fixate cop argument
18	O: 036 (fixate apply-create-referent :assertion).....	fixate create-referent asserted
	(sp chunk-27.....	<i>encode: feature memory</i> <i>(for illustration; not in Figure 21)</i>
	:chunk (state <s1> ^dummy d ^wm <wl> ^operator <o1>) <wl> -^assertion apply-create-referent ^dp <dl> <o1> ^goal create-referent ^om <o2> <dl> ^constant9 <r1> <r1> ^assertions apply-create-referent)	
	-->	
	(<o2> ^fixated <fl> + ^fixated <fl> &) <fl> ^assertion apply-create-referent + ^assertion apply-create-referent & ^region constant9 + ^region constant9 &))	
	(sp chunk-28.....	<i>encode:</i> <i>2. if WM says created-referent asserted,</i> <i>recall seeing it earlier</i> <i>(episodic memory)</i>
	:chunk (state <s1> ^dummy d ^wm <wl> ^operator <o1>) <wl> -^assertion apply-create-referent ^fixated <fl>) <fl> ^assertion apply-create-referent <o1> ^name comprehend ^om <o2>)	
	-->	
	(<o2> ^assertion apply-create-referent + ^assertion apply-create-referent &) <fl> ^time constant9 + ^time constant9 &))	

D.1.2. Recall episode

```

134 O: 0806 (comprehend chunk-128 :chunk).....comprehend chunk-128
135 O: 0810 (display print-chunk chunk-128 (t373)).....print chunk-128
136 O: 0806 (comprehend chunk-128 :chunk).....comprehend chunk-128
137 ==>S: S25 (operator no-change)
138 O: 0814 (attend t373 constant55)
139 O: 0818 (comprehend lhs :part)
140 O: 0834 (fixate :no-space)
141 O: 0820 (comprehend rhs :part)
142 O: 0811 (comprehend rhs :sp-parts).....comprehend right-hand sides
143 ==>S: S26 (operator no-change)
144 O: 0857 (comprehend lhs :part)
145 O: 0874 (fixate :action referent-of u1)
146 O: 0882 (comprehend u1 :imp-obj action)
147 O: 0873 (fixate :action type s-model)

```

```

148 O: 0887 (comprehend s-model :imp-obj action)
149 O: 0872 (fixate :action ul referent)..... fixate ul referent
150 O: 0891 (comprehend referent :imp-obj action)..... probe: comprehend referent
151 O: 0879 (fixate :condition referent nil)
152 O: 0898 (comprehend nil :imp-obj condition)
153 O: 0878 (fixate :condition path-to-referent)
154 O: 0904 (comprehend path-to-referent :imp-obj condition)
155 O: 0877 (fixate :condition operator* s-constructor16)
156 O: 0912 (comprehend operator* :op-cond).....comprehend operator condition
157 ==>S: S27 (operator no-change)
158 O: 0926 (comprehend lhs :part)
159 O: 0959 (fixate :no-space)..... fixate no problem space
160 O: 0966 (comprehend problem-space :imp-obj condition)
161 O: 0958 (fixate :condition type s-model-constructor)
162 O: 0953 (comprehend type :imp-obj action)
163 O: 0969 (imagine target top-context)
164 O: 0976 (comprehend top-context :imp-obj target)
165 O: 0968 (imagine target superstate)..... imagine superstate target
166 O: 0978 (comprehend superstate).....comprehend superstate
167 ==>S: S28 (operator no-change)

(sp po*imagine*assertions..... 3. if comprehending the superstate,
  (state <s> ^superstate.operator.goal superstate)      propose imagining asserted SPs that
  -->                                                    change the superstate
  (<s> ^operator <o1> <o2> <o3> <o4>)
  (<o1> ^name imagine ^terminate-and-reject t
    ^assertion apply-create-referent
    ^sp apply-create-referent
    ^interleave-best t
    ^trace apply-create-referent + &)
  (<o2> ^name imagine ^terminate-and-reject t
    ^assertion implement-exhausted
    ^sp implement-exhausted
    ^interleave-best t
    ^trace implement-exhausted + &)
  (<o3> ^name imagine ^terminate-and-reject t
    ^assertion apply-add-property
    ^sp apply-add-property
    ^interleave-best t
    ^trace apply-add-property + &)
  (<o4> ^name imagine ^terminate-and-reject t
    ^assertion apply-return-operator
    ^sp apply-return-operator
    ^interleave-best t
    ^trace apply-return-operator + &))

(sp po*fixate*shared-state..... 4. if comprehending the superstate,
  (state <s> ^goal superstate                          and superstate = current state,
    ^wm.superstate <s15>                               propose fixating on shared state
    ^wm (- ^state shared-state)
    ^wm.dp.<reg>.context (^newer none ^state <s15>)
    ^superstate.operator)
  -->
  (<s> ^operator <o> + >)
  (<o> ^name fixate ^region <reg> ^terminate-and-reject t
    ^state shared-state
    ^trace state + &, :shared-state + &))

168 O: 0992 (comprehend lhs :part)
169 O: 0987 (imagine apply-return-operator)
170 O: 0989 (fixate superstate s15 :superstate-id)
171 O: 0986 (imagine apply-add-property)
172 O: 01020 (fixate state :shared-state)..... fixate on shared state
173 O: 0985 (imagine implement-exhausted)
174 O: 01031 (comprehend implement-exhausted :imp-obj assertion)
175 O: 0984 (imagine apply-create-referent)..... imagine create-referent asserted

(sp chunk-28 ..... 2. if WM says created-referent asserted,
  :chunk                                               recall seeing it earlier
  (state <s1> ^dummy d ^wm <w1> ^operator <o1>)
  (<w1> -^assertion apply-create-referent ^fixated <f1>)
  (<f1> ^assertion apply-create-referent)
  (<o1> ^name comprehend ^om <o2>)
  -->

```

```

(<o2> ^assertion apply-create-referent + ^assertion apply-create-referent &)
(<fl> ^time constant9 + ^time constant9 &))

(sp po*comprehend*assertion*imagined.....5. if WM contains an asserted SP,
  (state <s> ^superstate nil
    ^wm.assertion <sp>
    ^wm.imagined-but-seen <sp>)
    and we recall having seen it,
    propose comprehending that asserted SP
  -->
  (<s> ^operator <o>)
  (<o> ^name comprehend
    ^goal <sp>
    ^trace :imagined-assertion
    ^om <om>))

176 O: 01037 (comprehend apply-create-referent.....comprehend create-referent asserted
  :imagined-assertion)

(sp po*display*scroll*to-sp.....6. if comprehending an SP that
  (state <g> ^superstate nil
    ^operator.goal <apply-create-referent>
    ^wm.<< fixated-recently imagined-but-seen >>
    <apply-create-referent>
    ^wm.<< assertion sp condition action >> <apply-create-referent>
    - ^wm.fixated (^ << sp assertion >>
      <apply-create-referent>
      - ^imagined-at)
    )
    we've seen but is now hidden,
    propose scrolling to it
  -->
  (<g> ^operator <o>)
  (<o> ^name display
    ^goal scroll:to-sp
    ^val <apply-create-referent>
    ^trace <apply-create-referent>
    ^terminate-and-reject t))

177 O: 01042 (display scroll:to-sp .....scroll to create-referent asserted
  apply-create-referent (t431))
178 O: 01037 (comprehend apply-create-referent.....comprehend create-referent asserted
  :imagined-assertion)
179 ==>S: S32 (operator no-change)

(sp po*fixate*current-context*shared-state ..... 7. if display shows an SP body, and
  (state <s> ^superstate.operator
    ^goal <apply-create-referent>
    ^wm.state shared-state
    ^wm (- ^<s-construct> shared-state)
    ^wm.dp.<reg> (^sp <apply-create-referent>
      ^condition.problem-space <s-construct>
      - ^condition <s-construct>) ; not the id values
    )
    WM contains a shared state,
    propose fixating the SP's problem space
  -->
  (<s> ^operator <o>)
  (<o> ^name fixate ^region <reg> ^terminate-and-reject t
    ^current-context <s-construct>
    ^<s-construct> shared-state
    ^trace :current-context-ss + &, <s-construct> + &))

180 O: 01047 (attend not-newest constant21)
181 O: 01069 (comprehend s15 :imp-obj superstate)
182 O: 01067 (comprehend shared-state :imp-obj state)
183 O: 01066 (fixate :current-context-ss s-construct).... fixate on s-construct

```

D.2. Scrolling event 2 - Figure 22, p. 53

D.2.1. Encoding episode

```

166 O: 0978 (comprehend superstate).....comprehend superstate
167 ==>S: S28 (operator no-change)
168 O: 0992 (comprehend lhs :part)
169 O: 0987 (imagine apply-return-operator)
170 O: 0989 (fixate superstate s15 :superstate-id)
171 O: 0986 (imagine apply-add-property)
172 O: 01020 (fixate state :shared-state)..... fixate on shared state
173 O: 0985 (imagine implement-exhausted)
174 O: 01031 (comprehend implement-exhausted :imp-obj assertion)
175 O: 0984 (imagine apply-create-referent)
176 O: 01037 (comprehend apply-create-referent.....comprehend create-referent asserted
           :imagined-assertion)
177 O: 01042 (display scroll:to-sp .....scroll to create-referent asserted
           apply-create-referent (t431))

```

D.2.2. Recall episode

```

178 O: 01037 (comprehend apply-create-referent.....comprehend create-referent asserted
           :imagined-assertion)
179 ==>S: S32 (operator no-change)

```

```

(sp chunk-34..... 1. if comprehending create-referent asserted,
  :chunk          and display shows referent-of <obj>,
  (state <s2> ^dummy d ^wm <w1> ^operator <o1>)      recall referent-of <obj> to WM
  (<w1> ^bound-by <for> ^dp <d1>)
  (<o1> ^goal apply-create-referent ^om <o2>)
  (<d1> ^constant13 <s1>)
  (<s1> ^condition <c1> ^action <a1>)
  (<c1> ^for obj)
  (<a1> ^referent-of obj)
-->
  (<o2> ^fixated <fl> + ^fixated <fl> &)
  (<fl> ^referent-of obj + ^referent-of obj & ^region constant13 +
    ^region constant13 &))

(sp po*fixate*current-context*shared-state ..... 2. if display shows an SP body, and
  (state <s> ^superstate.operator                    WM contains a shared state,
    ^goal <apply-create-referent>                    propose fixating the SP's problem space
    ^wm.state shared-state
    ^wm (- ^<s-construct> shared-state)
    ^wm.dp.<reg> (^sp <apply-create-referent>
      ^condition.problem-space <s-construct>
      - ^condition <s-construct>) ; not the id values
  )
-->
  (<s> ^operator <o>)
  (<o> ^name fixate ^region <reg> ^terminate-and-reject t
    ^current-context <s-construct>
    ^<s-construct> shared-state
    ^trace :current-context-ss + &, <s-construct> + &))
(sp a*state*important-objects..... creates important-objects table
  (state <s> ^superstate.operator)
-->
  (<s> ^important-object
    operator* + &,
    state + &,..... rule 3 (below) binds ^state shared-state
    superstate + &,
    fixated-recently + &,
    imagined-but-seen + &,
    impasse* + &,
    current-context + &,
    target + &,

```

```

        object + &,
        assertion + &,
        retraction + &,
        condition + &,
        action + &,
        chunk + &,
        sp + &,
    ))

(sp po*probe*with-important-object..... 3. if WM contains a state,
  (state <g> ^important-object <object>
    ^wm.<object> <probe>
    ^superstate.operator)
  -->
  (<g> ^operator <o> +)
  (<o> ^name comprehend ^goal <probe> ^probe t ^om <om>
    ^important-object <object>
    ^trace :imp-obj + &, <object> + &
    ^terminate-and-reject t))

180 O: 01047 (attend not-newest constant21)
181 O: 01069 (comprehend s15 :imp-obj superstate)
182 O: 01067 (comprehend shared-state :imp-obj state)... probe: comprehend shared state

(sp f:shared-states-build-chunks..... 4. if comprehending shared state,
  (state <s> ^operator.goal shared-state
    ^operator.om <om>)
  -->
  (<om> ^shared-state apply-chunks + &
    ^state shared-state + &))

(sp chunk-536..... encode:
  :chunk 5. if comprehending create-referent asserted
  (state <s1> ^dummy d ^wm <w1> ^operator <o1>) and WM contains shared state
  (<w1> ^shared-state apply-chunks recall that shared states cause chunks
    ^state shared-state) (re-encoded fact)
  (<o1> ^goal apply-create-referent ^om <o2>)
  -->
  (<o2> ^shared-state apply-chunks + ^shared-state apply-chunks &))

(sp po*comprehend*imagined-chunk-cause..... 6. if something causes chunks, and
  (state <s> ^superstate nil we recently saw that thing,
    ^wm.<< fixated-recently propose comprehending that thing
      imagined-but-seen >> <shared-state>
    ^wm.<shared-state> apply-chunks)
  -->
  (<s> ^operator <o>)
  (<o> ^name comprehend
    ^goal <shared-state>
    ^trace :imag-chunk-cause
    ^om <om>))

183 O: 01066 (fixate :current-context-ss s-construct)... fixate on s-construct
184 O: 01097 (comprehend shared-state :imag-chunk-cause)...comprehend shared state

(sp po*display*scroll*to-state..... 7. if comprehending a state, and
  (state <g> ^superstate nil we recently saw that state,
    ^operator.goal <shared-state> propose scrolling to it
    ^wm. << fixated-recently imagined-but-seen >> <shared-state>
    ^wm.state <shared-state>
    - ^wm.fixated (^state <shared-state> - ^imagined-at)
    )
  -->
  (<g> ^operator <o>)
  (<o> ^name display
    ^goal scroll:to-state
    ^val <shared-state>
    ^trace <shared-state>
    ^terminate-and-reject t))

185 O: 01103 (display scroll:to-state shared-state (t445))..scroll to state
186 O: 01097 (comprehend shared-state :imag-chunk-cause)...comprehend shared state

```

D.3. Scrolling event 3 - Figure 23, p. 59

D.3.1. Encoding episode

```

53 O: 0244 (comprehend for :objects-att).....comprehend for object
54 O: 0247 (display print-object for (t252)).....print u20
55 O: 0244 (comprehend for :objects-att).....comprehend for object
56 ==>S: S12 (operator no-change)
57   O: 0250 (attend t252 constant21)
58   O: 0266 (comprehend lhs :part)
59   O: 0263 (fixate head u17 :augmentation)
60   O: 0278 (comprehend head :att-of-id)

(sp f:recognize-u-model-object..... expertise recognizes object as a u-model
  :o-support ; tests nothing off state (by recognizing one of its attributes)
  (state <s> ^operator.goal { = <goal>
    << head left-edge right-edge bar-level
      word-id category annotation empty-node
      spec zero-head >> }
    ^operator.om <k>)
  -->
  (<k> ^object u-model + &
    ^u-model <goal> + &))

(sp po*fixate*no-referent..... 1. if WM contains a u-model, but
  (state <s> ^superstate.operator the display shows no referent,
    ^wm.object u-model propose fixating its absence
    ^wm.attribute-of-object <u20>
    ^wm (- ^referent nil)
    ^wm.dp.<reg> (^object-id <u20> - ^referent))
  -->
  (<s> ^operator <o>)
  (<o> ^name fixate ^region <reg> ^terminate-and-reject t
    ^attribute referent
    ^referent nil
    ^trace :no-referent))

61 O: 0262 (fixate zero-head u15 :augmentation)
62 O: 0284 (comprehend zero-head :att-of-id)
63 O: 0261 (fixate spec u14 :augmentation)
64 O: 0286 (comprehend spec :att-of-id)
65 O: 0260 (fixate empty-node e15 :augmentation)
66 O: 0288 (comprehend empty-node :att-of-id)
67 O: 0259 (fixate right-edge w13 :augmentation)
68 O: 0290 (comprehend right-edge :att-of-id)
69 O: 0257 (fixate left-edge w8 :augmentation)
70 O: 0292 (comprehend left-edge :att-of-id)
71 O: 0277 (fixate operator* for create-referent :bind-obj)
72 O: 0264 (comprehend operator* |(previous-goal)|)
73 O: 0281 (fixate :no-referent)..... fixate on no referent

(sp chunk-208..... encode:
  :chunk if WM contains no referent,
  (state <s1> ^dummy d ^wm <w1> ^operator <o1>) recall seeing no referent
  (<w1> -^referent nil ^fixated <f1>)
  (<f1> ^referent nil) (episodic memory; not shown in Figure 23)
  (<o1> ^name comprehend ^om <o2>)
  -->
  (<o2> ^referent nil + ^referent nil &)
  (<f1> ^time constant21 + ^time constant21 &))

(sp chunk-209..... encode:
  :chunk 2. if WM contains a referent,
  (state <s1> ^dummy d ^wm <w1> ^operator <o1>) recall looking for it
  (<w1> -^attribute referent ^fixated <f1>) (episodic memory)
  (<f1> ^attribute referent)
  (<o1> ^name comprehend ^om <o2>)
  -->
  (<o2> ^attribute referent + ^attribute referent &)
  (<f1> ^time constant21 + ^time constant21 &))

```

D.3.2. Recall episode, scrolling event 3

```

391 O: O2512 (comprehend u-model).....comprehend u-model
392 ==>S: S67 (operator no-change)

      (sp po*imagine*referent..... 3. if comprehending u-model,
        (state <s> ^superstate.operator.goal u-model          propose imagining referent on u-model
          ^wm.object u-model)
      -->
      (<s> ^operator <o> + >)
      (<o> ^name imagine ^terminate-and-reject t
        ^attribute referent + &
        ^u-model referent + &
        ^trace :referent))

393 O: O2530 (imagine :referent)..... imagine referent on u-model

      (sp chunk-209..... 2. if WM contains a referent,
        :chunk                                               recall looking for it
        (state <sl> ^dummy d ^wm <wl> ^operator <ol>)
        (<wl> -^attribute referent ^fixated <fl>)
        (<fl> ^attribute referent)
        (<ol> ^name comprehend ^om <o2>)
      -->
        (<o2> ^attribute referent + ^attribute referent &)
        (<fl> ^time constant21 + ^time constant21 &))

      (sp po*display*scroll*to-object.....4. if comprehending an object with
        (state <g> ^superstate nil                               an attribute we recall looking for,
          ^operator.goal <u-model>                             propose scrolling to the place we looked
          ^wm.object <u-model>
          ^wm. << fixated-recently imagined-but-seen >>
            <referent>
          ^wm.<u-model> <referent>
          - ^wm.fixated (^attribute <referent> - ^imagined-at)
          )
      -->
      (<g> ^operator <o>)
      (<o> ^name display
        ^goal scroll:to-object
        ^val <u-model>
        ^trace <u-model>
        ^terminate-and-reject t))

394 O: O2546 (display scroll:to-object u-model (t646)).....scroll to u-model
395 O: O2512 (comprehend u-model).....comprehend u-model
396 ==>S: S71 (operator no-change)

      (sp po*fixate*id-of-imagined-object..... 5. if comprehending an object whose
        (state <s> ^goal <u-model>                               structure we've seen and whose
          ^wm.<u-model> <referent>                               identifier is on display,
          ^wm.imagined-but-seen <referent>                     propose fixating on the identifier
          ^wm.attribute <referent>
          - ^wm.dp.newest ; means we've scrolled
          ^wm (- ^object-id <u20>)
          ^wm.dp.<reg>.object-id <u20>)
      -->
      (<s> ^operator <o> + >)
      (<o> ^name fixate ^region <reg> ^terminate-and-reject t
        ^object-id <u20> + &
        ^<u-model> <u20> + &
        ^trace <u20> + &, :id-of-imagined-att + &))

397 O: O2584 (fixate u20 :id-of-imagined-att)..... fixate on u20

      (sp po*display*print-object*fresh.....4. if comprehending an object, and
        (state <g> ^superstate nil                               WM contains its identifier, but
          ^operator.goal <for>                                   the version on display is stale,
          ^wm.<for> <u20> { <> <u20> <other-k> }               propose printing it fresh
          ^wm.object-id <u20>
          ^wm (- ^id <other-k>)
          - ^wm.dp.newest ; means screen is stale
        )
      )

```

```

)
-->
(<g> ^operator <o>)
(<o> ^name display
 ^goal print-object-fresh
 ^val <for>
 ^trace <u20>
 ^terminate-and-reject t))

```

programmer goes to prompt

```

398 O: 02588 (display print-object-fresh u-model (t649))....print u20 fresh
399 O: 02512 (comprehend u-model).....comprehend u-model
400 ==>S: S73 (operator no-change)
401   O: 02592 (attend t649 constant157)
402   O: 02597 (comprehend lhs :part)
403   O: 02613 (fixate referent r9 :augmentation)..... fixate on referent r9
404 O: 02631 (comprehend referent :objects-att)
405 O: 02633 (display print-object referent (t654))....print referent object
406 O: 02631 (comprehend referent :objects-att).....comprehend referent
407 ==>S: S74 (operator no-change)
408   O: 02636 (attend t654 constant159)
409   O: 02656 (fixate properties p88 :augmentation)..... fixate properties p88,
410   O: 02653 (fixate referent-of u20 :augmentation)
411   O: 02651 (fixate head u17 :augmentation)
412   O: 02654 (fixate properties p87 :augmentation)..... p87

```

D.4. Scrolling event 4 - Figure 24, p. 62

D.4.1. Encoding episode

```

486 O: 03140 (comprehend return-new-pointer.....comprehend return-new-pointer
              :high-lev-goal)

      (sp f:pay-attention-when-building-proposal.....1. if comprehending how to return a new pointer,
        (state <s> ^operator.goal return-new-pointer          and we're about to transfer to the superspace,
          ^wm.action proposal-build                          pay attention
          ^operator.om <om>))
      -->
      (<om> ^high-level-goal pay-attention + &
        ^pay-attention proposal-build + &))

487 ==>S: S89 (operator no-change)

      (sp p*fixate*assertion*pay-attention..... 2. if paying attention,
        (state <s> ^operator <o> +                          prefer fixating asserted SPs
          ^wm.high-level-goal pay-attention)
        (<o> ^name fixate ^assertion <a>))
      -->
      (<o> ^interleave-best t))

488 O: 03152 (comprehend lhs :part)
489 O: 03181 (fixate propose-return-operator..... fixate propose-return-operator
              :assertion)

      (sp chunk-1175..... 3. if WM contains propose-return-operator,
        :chunk                                               recall seeing it earlier
        (state <sl> ^dummy d ^wm <wl> ^operator <ol>))
        (<wl> ^assertion propose-return-operator          (encoded during DC 462, p. 303)
          ^fixated <fl>))
        (<fl> ^assertion propose-return-operator)
        (<ol> ^name comprehend ^om <o2>))
      -->
      (<o2> ^assertion propose-return-operator +
        ^assertion propose-return-operator &)
      (<fl> ^time constant180 + ^time constant180 &))

490 O: 03196 (comprehend propose-return-operator :imp-obj assertion)
491 O: 03182 (fixate terminate-s-model-creator..... fixate terminate-s-model-creator
              :assertion)

      (sp po*comprehend*assertion*pay-attention.....4. if paying attention, and WM contains
        (state <s> ^superstate nil                          an asserted SP we haven't seen before,
          ^wm.high-level-goal pay-attention                propose comprehending it
          ^wm.assertion <sp>
          ; ; actually saw <sp>, and for the first time:
          ^wm.fixated (^assertion <sp> ^time <now> - ^time <> <now>))
        -->
        (<s> ^operator <o>))
        (<o> ^name comprehend
          ^goal <sp>
          ^trace :assertion-pay-attn
          ^om <om>))

492 O: 03202 (comprehend terminate-s-model-creator.....comprehend terminate-s-model-creator
              :assertion-pay-attn)

      (sp po*display*print-sp*when-paying-attention.....5. if comprehending an asserted SP,
        (state <g> ^superstate nil                          and we're paying attention,
          ^operator.goal <terminate-smc>                    print the SP
          ^wm.assertion <terminate-smc>
          ^wm.high-level-goal pay-attention
          )
        -->
        (<g> ^operator <o>))
        (<o> ^name display
          ^goal print-sp:paying-attention

```

```

^val <terminate-smc>
^trace <terminate-smc>
^terminate-and-reject t))
493 O: 03206 (display print-sp:paying-attention.....print terminate-s-model-constructor
              terminate-s-model-constructor (t754))

```

D.4.2. Recall episode, scrolling event 4

```

494 O: 03202 (comprehend terminate-s-model-constructor.....comprehend terminate-s-model-constructor
              :assertion-pay-attn)
495 ==>S: S92 (operator no-change)
496 O: 03208 (attend t754 constant193)

(sp po*comprehend*assertion*pay-attention*.....6. if paying attention, and
              fixated-recently                    and WM contains an SP that's now hidden
              propose comprehending that SP
  (state <s> ^superstate nil
            ^wm.high-level-goal pay-attention
            ^wm.assertion <propose-return-operator>
            ^wm.fixated-recently <propose-return-operator>
      )
-->
(<s> ^operator <o>)
(<o> ^name comprehend
  ^goal <propose-return-operator>
  ^trace :assertion-fix-recent
  ^om <om>))

(sp a*state*important-objects..... creates important-objects table
  (state <s> ^superstate.operator)
-->
  (<s> ^important-object
    operator* + &,
    state + &,
    superstate + &,
    fixated-recently + &,
    imagined-but-seen + &,
    impasse* + &,
    current-context + &,
    target + &,
    object + &,
    assertion + &,
    retraction + &,
    condition + &,
    action + &,
    chunk + &,
    sp + &,))..... rule 7 binds ^sp propose-return-operator

(sp po*probe*with-important-object..... 7. if WM contains an SP,
  (state <g> ^important-object <object>
            ^wm.<object> <probe>
            ^superstate.operator)
-->
  (<g> ^operator <o> +)
  (<o> ^name comprehend ^goal <probe> ^probe t ^om <om>
    ^important-object <object>
    ^trace :imp-obj + &, <object> + &
    ^terminate-and-reject t))

497 O: 03213 (comprehend lhs :part)
498 O: 03218 (fixate bound-by type :bind-target)
499 O: 03211 (comprehend rhs :part)
500 O: 03241 (fixate :action s-model-constructor..... fixate reconsider
              reconsider)
501 O: 03250 (comprehend reconsider :imp-obj action)
502 O: 03238 (fixate :condition problem-space create-operator)
503 O: 03255 (comprehend create-operator :imp-obj condition)
504 O: 03237 (fixate :condition type s-model-constructor)
505 O: 03239 (comprehend type :imp-obj condition)
506 O: 03236 (fixate :condition annotation..... fixate construction-done

```

```

                                construction-done)
507 O: 03263 (comprehend construction-done :imp-obj condition)
508 O: 03225 (comprehend terminate-s-model-constructor :imp-obj sp)
509 O: 03223 (comprehend propose-return-operator..... probe: comprehend propose-return-operator
             :imp-obj sp)
510 O: 03227 (comprehend propose-return-operator.....comprehend propose-return-operator
             :assertion-fix-recent)

(sp po*display*scroll*to-sp.....8. if comprehending an SP that
  (state <g> ^superstate nil                               we've seen but is now hidden,
    ^operator.goal <apply-create-referent>                propose scrolling to it
    ^wm.<< fixated-recently imagined-but-seen >>
      <apply-create-referent>
    ^wm.<< assertion sp condition action >> <apply-create-referent>
    - ^wm.fixated (^ << sp assertion >>
      <apply-create-referent>
      - ^imagined-at)
  )
-->
(<g> ^operator <o>)
(<o> ^name display
  ^goal scroll:to-sp
  ^val <apply-create-referent>
  ^trace <apply-create-referent>
  ^terminate-and-reject t))

511 O: 03270 (display scroll:to-sp.....scroll to propose-return-operator
             propose-return-operator (t770))
512 O: 03227 (comprehend propose-return-operator.....comprehend propose-return-operator
             :assertion-fix-recent)
513 ==>S: S94 (operator no-change)
514 O: 03280 (attend not-newest constant193)
515 O: 03316 (comprehend terminate-s-model-constructor :imp-obj sp)

(sp p*fixate*fixated-recently-in-view*best..... 9. if a WM element was hidden but
  (state <s> ^operator <o> +                                is now visible again,
    ^wm.fixated-recently <val>                             prefer fixating it
    ^first-op <l>
    - ^fixate-meta-attribute <att>)
  (<o> ^name fixate ^<att> <val>)
  -->
  (<s> ^operator <o> >))

516 O: 03293 (fixate terminate-s-model-constructor :assertion)
517 O: 03292 (fixate propose-return-operator..... fixate propose-return-operator
             :assertion)

(sp po*display*print*high-level-goal.....10. if comprehending an asserted SP that
  (state <g> ^superstate nil                               was hidden but is now visible again,
    ^operator.goal <propose-return-operator>                propose printing it
    ^wm.fixated-recently <propose-return-operator>
    ^wm.fixated.<assertion> <propose-return-operator>
    ^wm.high-level-goal <return-new-pointer>
    ^wm.<return-new-pointer> <propose-return-operator>
  )
-->
(<g> ^operator <o>)
(<o> ^name display
  ^goal print:high-level-goal
  ^val <propose-return-operator>
  ^trace <propose-return-operator>
  ^terminate-and-reject t))

518 O: 03324 (display print:high-level-goal.....print propose-return-operator
             propose-return-operator (t774))
519 O: 03227 (comprehend propose-return-operator.....comprehend propose-return-operator
             :assertion-fix-recent)

```

D.5. Scrolling event 5 - Figure 25, p. 65

D.5.1. Encoding episode

```

136 O: 0806 (comprehend chunk-128 :chunk).....comprehend chunk-128
137 ==>S: S25 (operator no-change)
138 O: 0814 (attend t373 constant55)
139 O: 0818 (comprehend lhs :part)
140 O: 0834 (fixate :no-space)
141 O: 0820 (comprehend rhs :part)
142 O: 0811 (comprehend rhs :sp-parts).....comprehend right-hand sides
143 ==>S: S26 (operator no-change)
144 O: 0857 (comprehend lhs :part)
145 O: 0874 (fixate :action referent-of ul)
146 O: 0882 (comprehend ul :imp-obj action)
147 O: 0873 (fixate :action type s-model)
148 O: 0887 (comprehend s-model :imp-obj action)
149 O: 0872 (fixate :action ul referent)
150 O: 0891 (comprehend referent :imp-obj action)
151 O: 0879 (fixate :condition referent nil)
152 O: 0898 (comprehend nil :imp-obj condition)
153 O: 0878 (fixate :condition path-to-referent)
154 O: 0904 (comprehend path-to-referent :imp-obj condition)
155 O: 0877 (fixate :condition operator*.....fixate condition s-constructor16
      s-constructor16)

(sp chunk-475..... encode:
  :chunk                1. if WM contains condition s-constructor16,
  (state <s1> ^dummy d ^wm <w1> ^operator <o1>)                recall seeing it earlier
  (<w1> ^condition s-constructor16 ^fixated <f1>)
  (<f1> ^condition s-constructor16)                             (episodic memory)
  (<o1> ^name comprehend ^om <o2>)
-->
  (<o2> ^condition s-constructor16 + ^condition s-constructor16 &)
  (<f1> ^time constant58 + ^time constant58 &))

```

D.5.2. Recall episode

```

552 O: 03567 (comprehend new-operator :objects-att).....comprehend new-operator
553 ==>S: S103 (operator no-change)

(sp po*fixate*bind-op*id..... 2. if display shows and operator with an
  (state <s> ^superstate.operator                                identifier already in WM,
  ^wm.operator-id <o29>                                         propose fixing that operator
  ^wm.<o29> <o24>
  ^wm (- ^<o24> <s-constructor16>)
  ^wm.dp.<reg>.context (^op-id <o24> ^operator* <s-constructor16>))
-->
  (<s> ^operator <o> + >)
  (<o> ^name fixate ^region <reg> ^terminate-and-reject t
  ^<o24> <s-constructor16> + &
  ^operator* <s-constructor16> + &
  ^trace <o24> + &, operator* + &, <s-constructor16> + &,
  :bind-op-id + &))

554 O: 03577 (fixate o24 operator* s-constructor16..... fixate operator s-constructor16
      :bind-op-id)
555 O: 03580 (comprehend operator* :imp-obj condition)
556 O: 03586 (comprehend lhs :part)
557 O: 03584 (comprehend lhs :sp-parts).....comprehend left-hand sides
558 ==>S: S104 (operator no-change)
559 O: 03610 (comprehend lhs :part)
560 O: 03621 (fixate new-operator o24 :augmentation)
561 O: 03608 (comprehend new-operator |(previous-goal)|)
562 O: 03637 (fixate :condition problem-space create-operator)
563 O: 03642 (comprehend create-operator :imp-obj condition)

```

```

564 O: 03636 (fixate :condition type..... fixate condition s-model-constructor
          s-model-constructor)
565 O: 03650 (comprehend s-model-constructor :imp-obj condition)

(sp po*imagine*s-model-constructor..... 3. if WM contains an operator, and the
  (state <s> ^wm.condition s-model-constructor          s-model-constructor operator type
            ^wm.operator* <s-constructor16>)          as a condition,
  -->          propose imagining the operator as a condition
  (<s> ^operator <o> + >)
  (<o> ^name imagine ^terminate-and-reject t
    ^condition <s-constructor16>
    ^trace condition + &, <s-constructor16> + &))

566 O: 03648 (imagine condition s-constructor16)..... imagine condition s-constructor16

(sp chunk-475..... 1. if WM says condition s-constructor16,
  :chunk          recall seeing it earlier
  (state <s1> ^dummy d ^wm <w1> ^operator <o1>)
  (<w1> -^condition s-constructor16 ^fixated <f1>)
  (<f1> ^condition s-constructor16)
  (<o1> ^name comprehend ^om <o2>)
  -->
  (<o2> ^condition s-constructor16 + ^condition s-constructor16 &)
  (<f1> ^time constant58 + ^time constant58 &))

(sp po*comprehend*recalled-condition..... 4. if WM contains a condition
  (state <s> ^superstate nil          we recall having seen,
            ^wm.imagined-but-seen    propose comprehending it
            <s-model-constructor>
            ^wm.condition <s-model-constructor>)
  -->
  (<s> ^operator <o>)
  (<o> ^name comprehend
    ^goal <s-model-constructor>
    ^om <om>
    ^trace :recalled-condition))

567 O: 03656 (comprehend s-constructor16.....comprehend condition s-constructor16
          :recalled-condition)

(sp po*display*scroll*to-sp..... 5. if comprehending a condition
  (state <g> ^superstate nil          that we've seen but is now hidden,
            ^operator.goal <apply-create-referent>    propose scrolling to it
            ^wm.<< fixated-recently imagined-but-seen >>
            <apply-create-referent>
            ^wm.<< assertion sp condition action >> <apply-create-referent>
            - ^wm.fixated (^ << sp assertion >>
              <apply-create-referent>
              - ^imagined-at)
            )
  -->
  (<g> ^operator <o>)
  (<o> ^name display
    ^goal scroll:to-sp
    ^val <apply-create-referent>
    ^trace <apply-create-referent>
    ^terminate-and-reject t))

568 O: 03660 (display scroll:to-sp.....scroll to condition s-constructor16
          s-constructor16 (t845))
569 O: 03656 (comprehend s-constructor16.....comprehend condition s-constructor16
          :recalled-condition)

```

Appendix E

Display contents, protocol data, and model trace

This appendix presents the data from the programmer's display and the programmer's verbal and keystroke protocols. Section E.1 shows display contents and protocol data aligned with an operator trace of the model. Section E.2 (p. 225) presents a table that maps model commands to programmer keystrokes. This table serves as an index from model and programmer commands (Figure 26, p. 70) to the times that they occur in the protocol. It also contains a glossary of keystrokes that invoke Emacs commands like scrolling.

The display data consist of 300 lines of a GNU Emacs process buffer containing a running Soar process. The programmer ran her program interactively in this buffer, queried the process for the program's execution state, and printed out code. The 300 lines we are concerned with (d1003 to d1303) were visible at some point during the 10 minutes of behavior that we modeled. The window onto the process buffer was 60 lines high. When Emacs needed room for new output, it scrolled the window down on the buffer to accommodate the new output at the bottom. The scrolling distance was roughly 30 lines, depending on the size of the new output.

The protocol data consist of utterances and keystrokes, which we have arranged on a 1-second time course. To make the keystroke data more readable, we group nearby keystrokes M those typed less than 1 second apart M and present them together as one string. One second approximates \mathbf{M} , the mental-preparation operator in the Keystroke Level Model (Card et al., 1983, p. 264, put \mathbf{M} at 1.35 seconds). A 1-second threshold groups most keystrokes into identifiable commands.

E.1. Aligned display, protocol, and trace

Display, protocol, and trace are aligned at the commands issued by the programmer and the model. The tables on the next page illustrate the notation used in this alignment, using fictional data. The tables show display contents in the left column, protocol data in the middle column, and model trace (at the operator level; see Appendix D) in the right column. In the actual data, each table appears on a page by itself.

Aligning the three columns often requires a lot of white space in one or more columns. For example, on p. 201, the programmer spends a lot of time thinking about something she has recently printed out. The protocol column contains the utterances from this thinking time, but the display column is largely blank because she issues no new commands. One consequence of aligning the data in this way is that some continuity is lost. For example, the output shown on p. 201 is only part of what is on display during that

part of the protocol. The complete display visible at the time consists of all those regions generated since the most recent marker indicating the top of the window. This occurs on p. 198, three pages earlier. (The marker is an *italicized* line in the display column, indicating the top line of the window for a range of commands. Details of this marker are explained in the examples below.) The programmer generally talks about the most recent region on the display, so resolving references in the protocol seldom requires referring to previous pages. When the programmer clearly refers to older regions, as on p. 222 with the reference to "new operator o24" (t825), we include the referred-to output in the display column.

In the table below, the display column shows `command1 arg1`, a (fictional) command and argument issued by the programmer to the interpreter running in the process buffer. Below `command1` is its output. In the protocol column, the 1-second time course is on the left, utterances are in the middle, and `keystrokes` are aligned flush right. A double-underscore in the keystroke protocol (as in `command1__arg1`) is used to indicate a space (" "), to make spaces more visible. In the trace column, decision cycles are on the left and operators in the middle. `Command1` is horizontally aligned across all three columns. We emphasize alignment across all three columns by underlining the time of the command (t100). Alignment across only the display and protocol columns represents a programmer command that is missed or disregarded by the model, or that is part of a compound model command.

<u>Display contents</u>	<u>Verbal and keystroke protocol</u>	<u>Model trace</u>
d1000 Soar> <code>command1 arg1</code> <u>t100</u> d1001 d1002 output from <code>command1</code>	t100 <code>command1__arg1^M</code> t101 programmer's words t102 from t101 to t103 t103 (time in seconds)	1 0: 01 (display <code>command1</code> from model (<u>t100</u>))

The next two tables show the notation used for scrolling events. The table below shows *scrolling event 0*. (It also shows a dot (".") at the top right corner of the right column; dots at the top right of a column are artifacts of the typesetting software and should be ignored.)

d1100 d1101 output on display d1102 before scrolling d1103 <i>Scrolling event 0 (2 windows up): t203</i>	t200 utterances t201 before scrolling t202 command <u>t203</u> ^[v^[v	. 99 0: 039 (display scroll:to-something (<u>t203</u>))
---	--	---

A scrolling command is the last command to occur in a table.

The table after a scrolling command shows the command's effects on the display. In the table below, at the top of the scrolled-to window, is a message indicating which lines were redisplayed as a result of scrolling (*d1049 to d1054*). Within the scrolled-to window, each new-output command is still tagged with the time it was issued, but now the tag is parenthesized ("(t150)"). This means that the command's output has been redisplayed, as opposed to appearing on the display for the first time. Thus, parenthesized times do not align with times in the protocol column.

<p><i>Visible after t203 (scrolling event 0): d1049 to d1054</i> d1049 Soar> command2 (t150) d1050 d1051 output on display d1052 after scrolling, <i>Top of window from t170 until t190</i> d1053 generated earlier d1054 at t150</p>	<p>t204 utterances after t205 scrolling command</p>	<p>100 O: O40 (comprehend some-object) 101 ==>S: S20 (operator no-change) 102 O: O16 (attend not-newest constant44) 103 O: O15 (fixate feature)</p>
---	--	--

The table above also shows how we mark the top of the window, which changes when Emacs scrolls the window to accommodate new output. The *italicized* line in the display column says that a command at t170 generated output that caused Emacs to scroll the window. From t170 to t190, the display line d1053 (underneath the marker) was at the top of the window. At t190, another command caused the top line to change.

d1018 Soar> p s15	t164	t164	p__s15^M	Model not alive yet
d1019 (S15 ^annotation u-constructor-applied ^annotation u-model-success		t165		
d1020 ^name initial-state ^ordering-info O3 ^adjacency-info A3		t166 ok		
d1021 ^for-formatting F4 ^assigners A64 ^receivers R7 ^assigners2 A65		t167 u-constructor		
d1022 ^receivers2 R8)		t168 applied u-model		
d1023		t169 success		
		t170 order yeah yeah		
		t171 yeah all that stuff		
		t172 assigners receivers		
		t173		
		t174 assigners and		
		t175 receivers		
		t176 so now we		
		t177 start the really		
		t178 boring part		
d1024 Soar> ms	t179	t179	ms^M	
d1025 Assertions:		t180		
d1026 s-construct*propose*exhausted		t181 propose		
d1027 s-construct*propose-create-referent*noun		t182 exhausted propose		
d1028 Retractions:		t183 create referent noun		
d1029		t184 so there's the		
d1030 Soar>		t185 first production		
		t186		
		t187 that i care about		
		t188		
		t189		
		t190 now what i		
		t191 really it should be		
		t192 it would be like		
		t193 really nice if		
		t194 instead of having to do		
		t195 this now if i knew		
		t196 how to use the sde		

<pre> d1030 Soar> p s-construct*propose-create-referent*noun t197 d1031 (sp s-construct*propose-create-referent*noun d1032 (goal <g> ^problem-space <p> ^state <s>) d1033 (<p> ^name s-construct) d1034 (<s> ^{ << assigners assigners2 >> <a*1> } <v*1>) d1035 (<v*1> ^n <n*1>) d1036 (<n*1> ^max <m*1>) d1037 (<m*1> ^head <np>) <i>Top of window from t236 until t373</i> d1038 (<np> -^referent <r*1>) d1039 --> d1040 (<g> ^operator <o> + ^operator <o> =) d1041 (<o> ^name create-referent + ^for <np> +)) d1042 d1043 d1044 Soar> run 1 d1045 d1046 Soar> ms d1047 Assertions: d1048 Retractions: d1049 d1050 Soar> run 1 d1051 76: 0: 025 create-referent(cop) d1052 Soar> </pre>	<pre> t197 there's probably a way p__ s-construct*propose- t198 to do this create-referent*noun t199 t200 but i don't know how ^M t201 to use it so we're out t202 of luck t203 ok t204 so it's going to t205 take t206 t207 testing for the u-model t208 on the left hand side t209 that says t210 t211 you've got something out t212 there that's a noun t213 but doesn't have a t214 referent t215 create the operator t216 so that's cool t217 t218 run__1^Mms^M t219 now what's it going to t220 do ok so that's t221 the end of that decision t222 cycle t223 there's run__1^M t224 the operator </pre>	<p>Model not alive yet</p>
--	---	-----------------------------------

<pre> d1052 Soar> ms d1053 Assertions: d1054 s-construct*create-referent*touch-conjunct-symbol d1055 s-construct*create-referent d1056 Retractions: d1057 d1058 Soar> p s-construct*create-referent d1059 (sp s-construct*create-referent d1060 (goal <g> ^operator <o> ^problem-space <p> ^state <s>) d1061 (<o> ^name create-referent ^for <obj>) d1062 (<p> ^name s-construct) d1063 --> d1064 (<obj> ^referent <r> + ^referent <r> &) d1065 (<r> ^referent-of <obj> + ^type s-model +)) d1066 d1067 d1068 Soar> </pre>	<pre> t225 t226 ok t227 create referent t228 touch conjunct t229 symbol that's fine t230 that's just going to t231 string beads t232 and t233 this is the question t234 where is it sticking t235 this t236 p_s-construct* create-referent^M t237 so t238 it's sticking t239 t240 t241 t242 let's see t243 it's uh </pre>	<pre> Model issues its first command at t225 0 S: S1 1 O: O1 (display tinit (tinit)) 2 O: O2 (comprehend cinit) 3 ==>S: S2 (operator no-change) 4 O: O5 (attend tinit constant2) 5 O: O8 (fixate nil :assertion) 6 O: O11 (comprehend nil :imp-obj assertion) 7 O: O9 (comprehend nil :assertion) 8 ==>S: S3 (operator no-change) 9 O: O16 (comprehend cinit (previous-goal)) 10 O: O15 (fixate o25 :selected-id) 11 O: O14 (fixate create-referent :selected) 12 O: O26 (comprehend create-referent :selected-op) 13 O: O28 (display match-set:after-selection (t225)) 14 O: O26 (comprehend create-referent :selected-op) 15 ==>S: S5 (operator no-change) 16 O: O32 (attend t225 constant9) 17 O: O40 (fixate create-referent :argument cop) 18 O: O36 (fixate apply-create-referent :assertion) 19 O: O43 (comprehend apply-create-referent :assertion) 20 O: O47 (display print-sp:applies-operator apply-create-referent (t236)) 21 O: O43 (comprehend apply-create-referent :assertion) 22 ==>S: S8 (operator no-change) 23 O: O53 (attend t236 constant13) 24 O: O68 (comprehend create-referent :imp-obj operator*) 25 O: O67 (fixate condition for obj :bind-param) 26 O: O63 (comprehend rhs :part) 27 O: O50 (comprehend rhs :sp-parts) 28 ==>S: S9 (operator no-change) 29 O: O93 (comprehend lhs :part) 30 O: O101 (fixate :action referent-of obj) 31 O: O110 (comprehend obj :imp-obj action) 32 O: O100 (fixate :action type s-model) 33 O: O116 (comprehend s-model :imp-obj action) 34 O: O99 (fixate :action obj referent) 35 O: O121 (comprehend referent :imp-obj action) </pre>
---	---	--

<pre> d1068 Soar> p o25 d1069 (O25 ^name create-referent ^for U20) d1070 d1071 Soar> p u20 d1072 (U20 ^left-edge W8 ^right-edge W13 ^bar-level max ^word-id W13 ^category n d1073 ^annotation specified ^empty-node E15 ^spec U14 ^zero-head U15 d1074 ^head U17) d1075 d1076 Soar> </pre>	<pre> t244 p t245 o 25 p__o25^M t246 t247 ok the operator t248 has got t249 t250 what is u t251 20 t252 the operator has p__u20 t253 this for argument ^M t254 t255 the for argument is t256 the profile t257 in the u-model of t258 the thing you're creating t259 the referent for t260 t261 so what t262 rick's code is going to t263 do is it's t264 going to slap t265 it's going to generate t266 ok the production that t267 that's in the match set t268 now is t269 going to actually t270 create t271 create t272 referent for t273 right t274 this is the one that's t275 going to actually t276 create t277 the referent </pre>	<pre> 36 O: O107 (fixate operator* for create-referent :bind-obj) 37 O: O95 (comprehend for :imp-obj condition) 38 O: O106 (fixate :condition problem-space s-construct) 39 O: O135 (comprehend s-construct :imp-obj condition) 40 O: O105 (fixate :condition operator* create-referent) 41 O: O143 (comprehend operator* :op-cond) 42 ==>S: S10 (operator no-change) 43 O: O160 (comprehend lhs :part) 44 O: O162 (fixate condition for obj :bind-param) 45 O: O175 (comprehend referent-of :imp-obj action) 46 O: O154 (fixate o25 :selected-id) 47 O: O195 (display print-op (t245)) 48 O: O143 (comprehend operator* :op-cond) 49 ==>S: S11 (operator no-change) 50 O: O199 (attend t245 constant18) 51 O: O209 (comprehend lhs :part) 52 O: O206 (fixate for u20 :augmentation) 53 O: O244 (comprehend for :objects-att) 54 O: O247 (display print-object u20 (t252)) 55 O: O244 (comprehend for :objects-att) 56 ==>S: S12 (operator no-change) 57 O: O250 (attend t252 constant21) 58 O: O266 (comprehend lhs :part) 59 O: O263 (fixate head u17 :augmentation) 60 O: O278 (comprehend head :att-of-id) 61 O: O262 (fixate zero-head u15 :augmentation) 62 O: O284 (comprehend zero-head :att-of-id) 63 O: O261 (fixate spec u14 :augmentation) 64 O: O286 (comprehend spec :att-of-id) 65 O: O260 (fixate empty-node e15 :augmentation) 66 O: O288 (comprehend empty-node :att-of-id) 67 O: O259 (fixate right-edge w13 :augmentation) 68 O: O290 (comprehend right-edge :att-of-id) 69 O: O257 (fixate left-edge w8 :augmentation) 70 O: O292 (comprehend left-edge :att-of-id) 71 O: O277 (fixate operator* for create-referent :bind-obj) 72 O: O264 (comprehend operator* (previous-goal)) 73 O: O281 (fixate :no-referent) 74 O: O268 (comprehend rhs :part) 75 O: O303 (comprehend return-new-pointer (high-level-goal)) 76 O: O301 (comprehend return-new-pointer :high-lev-goal) </pre>
--	---	--

<pre> d1071 Soar> p u20 d1072 (U20 ^left-edge W8 ^right-edge W13 d1073 ^bar-level max ^word-id W13 ^category n ^annotation specified ^empty-node E15 ^spec U14 ^zero-head U15 d1074 ^head U17) d1075 d1076 Soar> </pre>	<pre> t252 t278 t279 t280 uh attribute t281 t282 t283 t284 t285 t286 and value t287 and stick it t288 on t289 this profile t290 t291 t292 so the question t293 is how long do i want to t294 leave it on there t295 [beep] t296 shut t297 up t298 t299 um t300 t301 t302 let's see t303 t304 t305 t306 because this is just t307 the bare node t308 it doesn't have any of the t309 properties on it t310 t311 so maybe t312 i'll leave it on t313 t314 t315 i could leave it t316 on t317 until t318 t319 exhaustion in this t320 space? t321 t322 what </pre>	<pre> 77 ==>S: S13 (operator no-change) 78 O: 0357 (imagine nil) 79 O: 0356 (imagine nil) 80 O: 0355 (imagine nil) 81 O: 0389 (imagine high-level-goal postpone) 82 O: 0398 (comprehend postpone :high-lev-goal) 83 ==>S: S14 (operator no-change) 84 O: 0404 (imagine operator* exhausted) 85 O: 0430 (comprehend current-context :sno) </pre>
---	---	---

d1076 Soar> pgs	t323	t323 space am i in?	pgs^M	86 O: 0432 (display print-stack (t323))
d1077 : ==>G: G1		t324 i'm in		87 O: 0430 (comprehend current-context :sno)
d1078 : P: P1 top-ps		t325 s-construct		88 ==>S: S16 (operator no-change)
d1079 : S: S1		t326		89 O: 0434 (attend t323 constant29)
<i>Top of window from t373 until t431</i>		t327		90 O: 0453 (comprehend exhausted :imp-obj operator*)
d1080 : O: O2 comprehend-input		t328		91 O: 0451 (fixate :current-context s-construct)
d1081 : ==>G: G2 operator no-change		t329		92 O: 0459 (comprehend s-construct :actual-context)
d1082 : P: P2 comprehension		t330 i'm going to run		93 ==>S: S17 (operator no-change)
d1083 : S: S4		t331 a little bit longer		94 O: 0489 (comprehend apply-chunks :imp-obj chunk)
d1084 : ==>G: G15 state no-change		t332		95 O: 0483 (fixate state s15 :state-id)
d1085 : P: P68 create-operator		t333		96 O: 0499 (comprehend lhs :part)
d1086 : S: S15		t334 letting it build the		97 O: 0497 (comprehend lhs :sp-parts)
d1087 : O: O24 s-structor16		t335 thing on the u-model		98 ==>S: S18 (operator no-change)
d1088 : ==>G: G16 operator no-change		t336 on the profile		99 O: 0527 (comprehend lhs :part)
d1089 : P: P85 s-construct		t337 in the		100 O: 0531 (fixate :current-context s-construct)
d1090 : S: S15		t338 u-model because		101 O: 0546 (comprehend s-construct :imp-obj
d1091 : O: O25 create-referent (cop)		t339 i don't think		current-context)
d1092		t340 i want		102 O: 0537 (comprehend apply-chunks :imp-obj sp)
d1093 Soar>		t341 it sitting around		103 O: 0543 (fixate :condition problem-space s-construct)
		t342 on the state		104 O: 0550 (comprehend problem-space :imp-obj condition)
		t343 because if i		105 O: 0542 (fixate :condition operator* create-referent)
		t344 do that i'm going to have to		106 O: 0556 (comprehend operator* :op-cond)
		t345 keep going through this		107 ==>S: S19 (operator no-change)
		t346 thing anyway following		108 O: 0578 (comprehend lhs :part)
		t347 pointers to get to it		109 O: 0582 (fixate state s15 :state-id)
		t348 so i think i'll		110 O: 0603 (comprehend s15 :imp-obj state)
		t349 i think i want to let		111 O: 0602 (fixate operator* for create-referent
		t350 it build the whole		:bind-obj)
		t351 thing and then		112 O: 0595 (comprehend for :imp-obj condition)
		t352		113 O: 0605 (comprehend for :objects-att)
		t353		114 ==>S: S20 (operator no-change)
		t354		115 O: 0664 (comprehend lhs :part)
		t355 and then take it		116 O: 0708 (comprehend apply-chunks :imp-obj chunk)
		t356 off this and put it		117 O: 0683 (comprehend referent-of :imp-obj action)
		t357 on the state but i'm not		118 O: 0681 (comprehend obj :imp-obj action)
		t358 sure in any event		119 O: 0679 (comprehend type :imp-obj action)
		t359 i have to run some		120 O: 0677 (comprehend s-model :imp-obj action)
		t360 more		121 O: 0675 (comprehend referent :imp-obj action)
		t361 so		122 O: 0671 (comprehend u-model :imp-obj object)
		t362 where was i that		123 O: 0642 (comprehend return-new-pointer
		t363 was the match set		(high-level-goal))
				124 O: 0716 (comprehend where-was-i :sno)
				125 ==>S: S21 (operator no-change)
				126 O: 0756 (fixate ms-t225 where-was-i:assertions)

d1093 Soar> run 1	<u>t364</u>	t364 run_1^Mm	127 O: 0757 (display run:where-was-i (t364))
d1094		t365	128 O: 0716 (comprehend where-was-i :sno)
d1095 Build: chunk-128		t366 ok what	129 ==>S: S23 (operator no-change)
d1096 Build: chunk-129		t367 rrr i	130 O: 0759 (attend t364 constant50)
		t368 just hit control h	131 O: 0797 (fixate builds-364 where-was-i:chunks)
		t369 again [beep]	132 O: 0795 (comprehend apply-chunks :imp-obj sp)
		t370	133 O: 0798 (fixate :chunk chunk-128)
		t371 fine let's see ^?	134 O: 0806 (comprehend chunk-128 :chunk)
		t372 what the chunks	
		t372 are doing	
d1097 Soar> p chunk-128	<u>t373</u>	t373 p_chunk-128^M	135 O: 0810 (display print-chunk chunk-128 (t373))
d1098 (sp chunk-128		t374	136 O: 0806 (comprehend chunk-128 :chunk)
d1099 :chunk		t375	137 ==>S: S25 (operator no-change)
d1100 (goal <gl> ^operator ^state <sl>)		t376	138 O: 0814 (attend t373 constant55)
d1101 (^name s-constructor16 ^type s-model- constructor)		t377	139 O: 0818 (comprehend lhs :part)
		t378	140 O: 0834 (fixate :no-space)
d1102 (<sl> ^assigners <al>)		t379 ok	141 O: 0820 (comprehend rhs :part)
d1103 (<al> ^n <n2>)		t380 this chunk	142 O: 0811 (comprehend rhs :sp-parts)
d1104 (<n2> ^max <n1>)		t381	143 ==>S: S26 (operator no-change)
d1105 (<n1> ^head)		t382 is test	144 O: 0857 (comprehend lhs :part)
d1106 (^referent <r*1>)		t383 ing for s	145 O: 0874 (fixate :action referent-of ul)
d1107 -->		t384 constructor	146 O: 0882 (comprehend ul :imp-obj action)
d1108 (^referent <r1> + ^referent <r1> &)		t385 16	147 O: 0873 (fixate :action type s-model)
d1109 (<r1> ^referent-of + ^type s-model +)		t386	148 O: 0887 (comprehend s-model :imp-obj action)
d1110		t387 so it's just	149 O: 0872 (fixate :action ul referent)
d1111		t388	150 O: 0891 (comprehend referent :imp-obj action)
d1112 Soar>		t389 where is this	151 O: 0879 (fixate :condition referent nil)
		t390	152 O: 0898 (comprehend nil :imp-obj condition)
		t391 uh	153 O: 0878 (fixate :condition path-to-referent)
		t392 operator	154 O: 0904 (comprehend path-to-referent :imp-obj condition)
		t393 is in	
		t394	155 O: 0877 (fixate :condition operator* s-constructor16)
		t395	
		t396	156 O: 0912 (comprehend operator* :op-cond)
		t397 huh?	157 ==>S: S27 (operator no-change)
		t398	158 O: 0926 (comprehend lhs :part)
		t399	159 O: 0959 (fixate :no-space)
		t400 it's in that slot	160 O: 0966 (comprehend problem-space :imp-obj condition)
		t401 as a state?	
		t402	161 O: 0958 (fixate :condition type s-model-constructor)
		t403 oh it's not testing	
		t404 for a problem space	162 O: 0953 (comprehend type :imp-obj action)
		t405 that's why ok	163 O: 0969 (imagine target top-context)

<p><i>Top of window from t373 until t431</i></p> <pre> d1080 : O: O2 comprehend-input d1081 : ==>G: G2 operator no-change d1082 : P: P2 comprehension d1083 : S: S4 d1084 : ==>G: G15 state no-change d1085 : P: P68 create-operator d1086 : S: S15 d1087 : O: O24 s-constructor16 d1088 : ==>G: G16 operator no-change d1089 : P: P85 s-construct d1090 : S: S15 d1091 : O: O25 create-referent(cop) d1092 d1093 Soar> run 1 t364 d1094 d1095 Build: chunk-128 d1096 Build: chunk-129 d1097 Soar> p chunk-128 t373 d1098 (sp chunk-128 d1099 :chunk d1100 (goal <g1> ^operator ^state <s1>) d1101 (^name s-constructor16 ^type s-model- constructor) d1102 (<s1> ^assigners <a1>) d1103 (<a1> ^n <n2>) d1104 (<n2> ^max <n1>) d1105 (<n1> ^head) d1106 (-^referent <r*1>) d1107 --> d1108 (^referent <r1> + ^referent <r1> &) d1109 (<r1> ^referent-of + ^type s-model +)) d1110 d1111 d1112 Soar> </pre> <p style="text-align: right;"><i>Scrolling event 1 (1 window up): t431</i></p>	<pre> t406 t407 so t408 t409 t410 assigners t411 a l t412 t413 t414 t415 t416 so it must t417 have just changed t418 it on t419 t420 on the sup t421 erstate? is that t422 oh these are shared t423 states t424 i t425 see t426 t427 no i don't see t428 t429 what t430 the hell built that t431 </pre> <p style="text-align: right;">^[v</p>	<pre> 164 O: 0976 (comprehend top-context :imp-obj target) 165 O: 0968 (imagine target superstate) 166 O: 0978 (comprehend superstate) 167 ==>S: S28 (operator no-change) 168 O: 0992 (comprehend lhs :part) 169 O: 0987 (imagine apply-return-operator) 170 O: 0989 (fixate superstate s15 :superstate-id) 171 O: 0986 (imagine apply-add-property) 172 O: 01020 (fixate state :shared-state) 173 O: 0985 (imagine implement-exhausted) 174 O: 01031 (comprehend implement-exhausted :imp-obj assertion) 175 O: 0984 (imagine apply-create-referent) 176 O: 01037 (comprehend apply-create-referent :imagined-assertion) 177 O: 01042 (display scroll:to-sp apply-create-referent (t431)) </pre>
---	---	---

<i>Visible after t431 (scrolling event 1): d1021 to d1081</i>		
<pre> d1021 ... d1030 Soar> p s-construct*propose-create-referent*noun (t197) d1031 (sp s-construct*propose-create-referent*noun d1032 (goal <g> ^problem-space <p> ^state <s>) d1033 (<p> ^name s-construct) d1034 (<s> ^{ << assigners assigners2 >> <a*1> } <v*1>) d1035 (<v*1> ^n <n*1>) d1036 (<n*1> ^max <m*1>) d1037 (<m*1> ^head <np>) d1038 (<np> -^referent <r*1>) d1039 --> d1040 (<g> ^operator <o> + ^operator <o> =) d1041 (<o> ^name create-referent + ^for <np> +)) d1042 d1043 d1044 Soar> run 1 d1045 d1046 Soar> ms (t218) ... d1050 Soar> run 1 (t223) d1051 76: O: 025 create-referent(cop) d1052 Soar> ms (t225) ... d1058 Soar> p s-construct*create-referent (t236) d1059 (sp s-construct*create-referent d1060 (goal <g> ^operator <o> ^problem-space <p> ^state <s>) d1061 (<o> ^name create-referent ^for <obj>) d1062 (<p> ^name s-construct) d1063 --> d1064 (<obj> ^referent <r> + ^referent <r> &) d1065 (<r> ^referent-of <obj> + ^type s-model +)) d1066 d1067 d1068 Soar> p o25 (t245) ... d1071 Soar> p u20 (t252) ... d1076 Soar> pgs (t323) ... d1081 : ==>G: G2 operator no-change </pre>	<pre> t432 t433 t434 t435 this said t436 if you're in t437 the s cons t438 truct problem space t439 t440 you slap t441 t442 t443 t444 that attribute t445 on the ^V </pre>	<pre> 178 O: 01037 (comprehend apply-create-referent :imagined-assertion) 179 ==>S: S32 (operator no-change) 180 O: 01047 (attend not-newest constant21) 181 O: 01069 (comprehend s15 :imp-obj superstate) 182 O: 01067 (comprehend shared-state :imp-obj state) 183 O: 01066 (fixate :current-context-ss s-construct) 184 O: 01097 (comprehend shared-state :imag-chunk-cause) 185 O: 01103 (display scroll:to-state shared-state (t445)) </pre>
<i>Scrolling event 2 (1 window down): t445</i>		

<p>Visible after t445 (scrolling event 2): d1080 to d1112 <i>Top of window until t564</i></p> <pre> d1080 : O: O2 comprehend-input d1081 : ==>G: G2 operator no-change d1082 : P: P2 comprehension d1083 : S: S4 d1084 : ==>G: G15 state no-change d1085 : P: P68 create-operator d1086 : S: S15 d1087 : O: O24 s-constructor16 d1088 : ==>G: G16 operator no-change d1089 : P: P85 s-construct d1090 : S: S15 d1091 : O: O25 create-referent(cop) d1092 d1093 Soar> run 1 (t364) d1094 d1095 Build: chunk-128 d1096 Build: chunk-129 d1097 Soar> p chunk-128 (t373) d1098 (sp chunk-128 d1099 :chunk d1100 (goal <g1> ^operator <o1> ^state <s1>) d1101 (<o1> ^name s-constructor16 ^type s-model-constructor) d1102 (<s1> ^assigners <a1>) d1103 (<a1> ^n <n2>) d1104 (<n2> ^max <n1>) d1105 (<n1> ^head <u1>) d1106 (<u1> ^referent <r*1>) d1107 --> d1108 (<u1> ^referent <r1> + ^referent <r1> &) d1109 (<r1> ^referent-of <u1> + ^type s-model +)) d1110 <i>Top of window from t564 until t596</i> d1111 d1112 Soar> </pre>	<pre> t446 object t447 so i'm in the t448 s-construct problem t449 space i'm going to slap t450 that thing on there t451 t452 and lo and behold t453 t454 t455 t456 i get this t457 chunk t458 because they share the state t459 t460 t461 t462 t463 i don't know t464 why it's testing for the t465 t466 t467 operator? t468 t469 why is it testing for the t470 operator t471 t472 t473 t474 t475 t476 t477 because this is an op t478 erator no change t479 t480 must be it t481 but i don't actually know that t482 t483 ok t484 t485 it t486 found all that crap t487 because it's all a copy t488 t489 ok </pre>	<pre> 186 O: O1097 (comprehend shared-state :imag-chunk-cause) 187 ==>S: S36 (operator no-change) 188 O: O1105 (attend not-newest constant55) 189 O: O1123 (comprehend for :imp-obj condition) 190 O: O1121 (comprehend s-construct :imp-obj current-context) 191 O: O1119 (comprehend s15 :imp-obj fixated-recently) 192 O: O1109 (comprehend lhs :part) 193 O: O1131 (comprehend obj :imp-obj fixated-recently) 194 O: O1129 (comprehend referent-of :imp-obj fixated-recently) 195 O: O1135 (comprehend where-was-i (where-was-i)) 196 O: O1142 (fixate :second chunk-129) 197 O: O1154 (comprehend chunk-129 :chunk) </pre>
--	--	--

<pre> d1112 Soar> p chunk-129 d1113 (sp chunk-129 d1114 :chunk d1115 (goal <gl> ^state <s1> ^operator) d1116 (<s1> ^name initial-state ^assigners <a1>) d1117 (^name s-constructor16 ^conjunct-symbol conjunct d1118 ^type s-model-constructor) d1119 (<a1> ^n <n2>) d1120 (<n2> ^max <n1>) d1121 (<n1> ^head) d1122 (-^referent <r*1>) d1123 --> d1124 (^conjunct-symbol conjunct - ^conjunct-symbol <c1> +)) d1125 d1126 d1127 Soar> </pre>	<pre> t490 t491 t492 alright rick t493 which means i now t494 have t495 t496 t497 i now have it sit t498 ting there t499 t500 and the second chunk t501 i know is just going t502 to be testing t503 for t504 t505 t506 um t507 the t508 bead right t509 yeah there's the con t510 junct symbol t511 fine t512 </pre>	<p>Goto-prompt disregarded</p> <pre> 198 O: 01158 (display print-chunk chunk-129 (t503)) 199 O: 01154 (comprehend chunk-129 :chunk) 200 ==>S: S38 (operator no-change) 201 O: 01164 (attend t503 constant73) 202 O: 01169 (comprehend lhs :part) 203 O: 01209 (fixate :no-space) 204 O: 01167 (comprehend rhs :part) 205 O: 01161 (comprehend rhs :sp-parts) 206 ==>S: S39 (operator no-change) 207 O: 01256 (comprehend lhs :part) 208 O: 01299 (fixate :action conjunct-symbol replacement) 209 O: 01314 (comprehend replacement :imp-obj action) 210 O: 01307 (fixate :condition conjunct-symbol conjunct) 211 O: 01312 (comprehend conjunct-symbol :imp-obj action) 212 O: 01306 (fixate :condition type s-model-constructor) 213 O: 01291 (comprehend type :imp-obj action) 214 O: 01343 (imagine action conjunct) 215 O: 01344 (comprehend conjunct :imp-obj action) 216 O: 01348 (imagine action s-model-constructor) 217 O: 01353 (comprehend s-model-constructor :imp-obj action) 218 O: 01310 (comprehend apply-chunks :imp-obj chunk) 219 O: 01362 (imagine action nil) 220 O: 01371 (comprehend nil :imp-obj action) 221 O: 01297 (comprehend chunk-129 :imp-obj sp) 222 O: 01293 (comprehend ul :imp-obj action) 223 O: 01289 (comprehend s-model :imp-obj action) 224 O: 01287 (comprehend referent :imp-obj action) </pre>
--	---	---

d1127 Soar> ms	<u>t513</u>	t513 ok	ms^M	225 O: 01408 (display match-set:asserted-sp (t513))
d1128 Assertions:		t514		226 O: 01137 (comprehend return-new-pointer :high-lev-goal)
d1129 s-construct*terminate*create-referent		t515 terminate		227 ==>S: S41 (operator no-change)
d1130 s-construct*propose*add-property*head-noun*cop*		t516 create referent		228 O: 01414 (attend t513 constant78)
animate		t517 fine		229 O: 01428 (imagine nil)
d1131 s-construct*propose*add-property*head-noun*cop		t518 propose		230 O: 01460 (comprehend nil :imp-obj action)
d1132 Retractions:		t519 add property head		231 O: 01429 (fixate terminate-create-referent :assertion)
d1133 chunk-129		t520 noun cop animate		232 O: 01482 (comprehend terminate-create-referent :assertion)
d1134 chunk-128		t521 propose		233 ==>S: S42 (operator no-change)
d1135 s-construct*create-referent*touch-conjunct-symbol		t522 add property		234 O: 01494 (comprehend lhs :part)
d1136 s-construct*propose-create-referent*noun		t523 head noun cop		235 O: 01499 (fixate head-noun-cop-animate :assertion)
d1137		t524		236 O: 01544 (comprehend head-noun-cop-animate :assertion)
d1138 Soar>		t525 ok		237 ==>S: S43 (operator no-change)
		t526		238 O: 01552 (comprehend lhs :part)
		t527		239 O: 01558 (fixate head-noun-cop :assertion)
		t528		240 O: 01585 (comprehend head-noun-cop :assertion)
		t529 why are		241 ==>S: S44 (operator no-change)
		t530 the chunks retracting		242 O: 01593 (comprehend lhs :part)
		t531		243 O: 01598 (fixate terminate-create-referent :assertion)
		t532		244 O: 01629 (comprehend terminate-create-referent :assertion)
		t533 oh because it has a		245 ==>S: S45 (operator no-change)
		t534 referent fine		246 O: 01639 (comprehend lhs :part)
		t535 um		247 O: 01644 (fixate head-noun-cop :assertion)
		t536		248 O: 01641 (comprehend rhs :part)
		t537		249 O: 01653 (comprehend apply-chunks :imp-obj chunk)
		t538 what's going to happen		250 O: 01729 (imagine condition add-property)
		t539 when it does this it's		251 O: 01741 (comprehend add-property :selected-op)
		t540 going to build some more		
		t541		
		t542 should build		
		t543 two more chunks		
		t544		
		t545		
		t546 which means that		
		t547 in both spaces		
		t548 it will have		
		t549		
		t550		
		t551 added the properties		

<pre>d1138 Soar> run 1 d1139 d1140 Soar></pre>	<pre> t552 to t553 t554 hm! t555 t556 why didn't it build a chunk t557 t558 t559 t560 t561 oh those are operators t562 it's proposing them fine t563 </pre>	<pre> run__1^M 252 O: 01745 (display run:to-builds (t552)) 253 O: 01741 (comprehend add-property :selected-op) n254 ==>S: S46 (operator no-change) 255 O: 01747 (attend t552 constant96) 256 O: 01757 (fixate :no-chunk) 257 O: 01751 (comprehend lhs :part) 258 O: 01810 (comprehend apply-chunks :imp-obj sp) 259 O: 01808 (comprehend terminate-create-referent :imp-obj sp) 260 O: 01806 (comprehend referent-of :imp-obj action) 261 O: 01804 (comprehend ul :imp-obj action) 262 O: 01802 (comprehend type :imp-obj action) 263 O: 01800 (comprehend s-model :imp-obj action) 264 O: 01798 (comprehend referent :imp-obj action) 265 O: 01749 (comprehend terminate-create-referent (previous-goal)) 266 O: 01796 (comprehend conjunct-symbol :imp-obj action) 267 O: 01794 (comprehend replacement :imp-obj action) 268 O: 01788 (comprehend nil :imp-obj condition) 269 O: 01786 (comprehend path-to-referent :imp-obj condition) 270 O: 01784 (comprehend operator* :imp-obj condition) 271 O: 01833 (imagine target top-context) 272 O: 01836 (comprehend top-context :imp-obj target) 273 O: 01782 (comprehend s-constructor16 :imp-obj condition) 274 O: 01778 (comprehend conjunct :imp-obj condition) 275 O: 01774 (comprehend s-model-constructor :imp-obj condition) 276 O: 01824 (comprehend apply-chunks :imp-obj chunk) 277 O: 01772 (comprehend head-noun-cop :imp-obj assertion) 278 O: 01792 (comprehend add-property :imp-obj condition) 279 O: 01768 (comprehend superstate :imp-obj target) 280 O: 01770 (comprehend head-noun-cop-animate :imp-obj assertion) 281 O: 01764 (comprehend add-property :imp-obj operator*) 282 O: 01760 (comprehend return-new-pointer (high-level-goal)) 283 O: 01753 (comprehend rhs :part) 284 O: 01856 (imagine action s-model-constructor) 285 O: 01860 (comprehend s-model-constructor :imp-obj action) 286 O: 01864 (imagine action nil) 287 O: 01869 (comprehend nil :imp-obj action) 288 O: 01871 (imagine action conjunct) 289 O: 01878 (comprehend conjunct :imp-obj action) 290 O: 01748 (fixate builds builds-364 g:builds) 291 O: 01887 (imagine add-property builds) </pre>
--	---	--

<pre> d1140 Soar> ms d1141 Assertions: d1142 Retractions: <i>Top of window from t596 until t639</i> d1143 d1144 Soar> run 1 d1145 77: 0: 028 add-property(t) </pre>	<p style="text-align: right;"><u>t564</u></p> <p style="text-align: right;"><u>t565</u></p> <pre> t566 fine t567 so we're going to do it one t568 at a time with an operator t569 slap them on t570 t571 ok so this is the t572 thing that's putting animate t573 true on t574 </pre>	<pre> ms^M 292 O: 01888 (display match-set:after-builds (t564)) 293 O: 01741 (comprehend add-property :selected-op) 294 ==>S: S47 (operator no-change) 295 O: 01893 (attend t564 constant101) 296 O: 01900 (fixate add-property :argument t) 297 O: 01896 (comprehend lhs :part) Miss 1 298 O: 01907 (fixate retraction nil) 299 O: 01990 (comprehend nil :imp-obj retraction) 300 O: 01906 (fixate o28 :selected-id) 301 O: 01989 (imagine referent retraction) 302 O: 01957 (comprehend referent :imp-obj action) 303 O: 01977 (comprehend terminate-create-referent :imp-obj sp) 304 O: 01908 (fixate nil :assertion) 305 O: 01994 (comprehend nil :assertion) 306 ==>S: S48 (operator no-change) 307 O: 02041 (comprehend head-noun-cop-animate :imp-obj sp) 308 O: 02005 (comprehend add-property :imp-obj operator*) 309 O: 02049 (comprehend lhs :part) 310 O: 02047 (comprehend lhs :sp-parts) 311 ==>S: S49 (operator no-change) 312 O: 02065 (comprehend lhs :part) 313 O: 02073 (fixate nil :assertion) 314 O: 02088 (comprehend nil :imp-obj assertion) 315 O: 02070 (fixate o28 :selected-id) 316 O: 02085 (fixate :no-space) 317 O: 02069 (fixate add-property :selected) 318 O: 02098 (comprehend add-property :selected-op) 319 O: 02100 (display match-set:after-selection (t575)) 320 O: 02098 (comprehend add-property :selected-op) 321 ==>S: S51 (operator no-change) 322 O: 02104 (attend t575 constant115) 323 O: 02105 (comprehend lhs (previous-goal)) 324 O: 02111 (fixate touch-conjunct-symbol :assertion) 325 O: 02132 (comprehend touch-conjunct-symbol :assertion) 326 ==>S: S52 (operator no-change) 327 O: 02145 (comprehend lhs :part) 328 O: 02151 (fixate apply-add-property :assertion) 329 O: 02177 (comprehend apply-add-property :assertion) </pre>
<pre> d1146 Soar> ms d1147 Assertions: d1148 s-construct*add-property*touch- conjunction-symbol d1149 s-construct*add-property d1150 Retractions: d1151 s-construct*terminate*create-referent d1152 s-construct*create-referent d1153 d1154 Soar> </pre>	<p style="text-align: right;"><u>t575</u></p> <p style="text-align: right;"><u>t576</u></p> <pre> t577 touch the t578 conjunct symbol t579 t580 add the properties so t581 i should get two more </pre>	<pre> ms^M 292 O: 01888 (display match-set:after-builds (t564)) 293 O: 01741 (comprehend add-property :selected-op) 294 ==>S: S47 (operator no-change) 295 O: 01893 (attend t564 constant101) 296 O: 01900 (fixate add-property :argument t) 297 O: 01896 (comprehend lhs :part) Miss 1 298 O: 01907 (fixate retraction nil) 299 O: 01990 (comprehend nil :imp-obj retraction) 300 O: 01906 (fixate o28 :selected-id) 301 O: 01989 (imagine referent retraction) 302 O: 01957 (comprehend referent :imp-obj action) 303 O: 01977 (comprehend terminate-create-referent :imp-obj sp) 304 O: 01908 (fixate nil :assertion) 305 O: 01994 (comprehend nil :assertion) 306 ==>S: S48 (operator no-change) 307 O: 02041 (comprehend head-noun-cop-animate :imp-obj sp) 308 O: 02005 (comprehend add-property :imp-obj operator*) 309 O: 02049 (comprehend lhs :part) 310 O: 02047 (comprehend lhs :sp-parts) 311 ==>S: S49 (operator no-change) 312 O: 02065 (comprehend lhs :part) 313 O: 02073 (fixate nil :assertion) 314 O: 02088 (comprehend nil :imp-obj assertion) 315 O: 02070 (fixate o28 :selected-id) 316 O: 02085 (fixate :no-space) 317 O: 02069 (fixate add-property :selected) 318 O: 02098 (comprehend add-property :selected-op) 319 O: 02100 (display match-set:after-selection (t575)) 320 O: 02098 (comprehend add-property :selected-op) 321 ==>S: S51 (operator no-change) 322 O: 02104 (attend t575 constant115) 323 O: 02105 (comprehend lhs (previous-goal)) 324 O: 02111 (fixate touch-conjunct-symbol :assertion) 325 O: 02132 (comprehend touch-conjunct-symbol :assertion) 326 ==>S: S52 (operator no-change) 327 O: 02145 (comprehend lhs :part) 328 O: 02151 (fixate apply-add-property :assertion) 329 O: 02177 (comprehend apply-add-property :assertion) </pre>

<pre>d1154 Soar> run 1 d1155 d1156 Build: chunk-130 d1157 Build: chunk-131</pre>	<p><u>t582</u></p>	<pre>t582 chunks run__1^M t583 yes t584 t585 and t586 then i should get</pre>	<pre>330 O: 02185 (display run:to-builds (t582)) 331 O: 02177 (comprehend apply-add-property :assertion) 332 ==>S: S54 (operator no-change) 333 O: 02187 (attend t582 constant124) 334 O: 02208 (comprehend touch-conjunct-symbol :imp-obj sp) 335 O: 02188 (fixate builds builds-582 g:builds) 336 O: 02191 (comprehend rhs :part) 337 O: 02193 (comprehend lhs :part) 338 O: 02212 (imagine apply-add-property builds) 339 O: 02239 (display match-set:after-builds (t587)) 340 O: 02177 (comprehend apply-add-property :assertion) 341 ==>S: S56 (operator no-change) 342 O: 02242 (attend t587 constant126) 343 O: 02249 (fixate :no-chunk) 344 O: 02283 (comprehend nil :imp-obj chunk) 345 O: 02257 (fixate terminate-add-property :assertion) 346 O: 02285 (comprehend terminate-add-property :assertion)</pre>
<pre>d1158 Soar> ms d1159 Assertions: d1160 s-construct*terminate*add-property d1161 Retractions: d1162 chunk-131 d1163 chunk-130 d1164 s-construct*add-property*touch-conjunct-symbol d1165 s-construct*propose*add-property*head-noun*cop* animate d1166</pre>	<p><u>t587</u></p>	<pre>t587 ms t588 reconsiders t589 yup ^M t590 t591 t592 and</pre>	<pre>330 O: 02185 (display run:to-builds (t582)) 331 O: 02177 (comprehend apply-add-property :assertion) 332 ==>S: S54 (operator no-change) 333 O: 02187 (attend t582 constant124) 334 O: 02208 (comprehend touch-conjunct-symbol :imp-obj sp) 335 O: 02188 (fixate builds builds-582 g:builds) 336 O: 02191 (comprehend rhs :part) 337 O: 02193 (comprehend lhs :part) 338 O: 02212 (imagine apply-add-property builds) 339 O: 02239 (display match-set:after-builds (t587)) 340 O: 02177 (comprehend apply-add-property :assertion) 341 ==>S: S56 (operator no-change) 342 O: 02242 (attend t587 constant126) 343 O: 02249 (fixate :no-chunk) 344 O: 02283 (comprehend nil :imp-obj chunk) 345 O: 02257 (fixate terminate-add-property :assertion) 346 O: 02285 (comprehend terminate-add-property :assertion)</pre>

<pre> d1195 Soar> pgs d1196 : ==>G: G1 d1197 : P: P1 top-ps d1198 : S: S1 d1199 : O: O2 comprehend-input d1200 : ==>G: G2 operator no-change d1201 : P: P2 comprehension d1202 : S: S4 d1203 : ==>G: G15 state no-change d1204 : P: P68 create-operator d1205 : S: S15 d1206 : O: O24 s-constructor16 d1207 : ==>G: G16 operator no-change d1208 : P: P85 s-construct d1209 : S: S15 d1210 : O: O26 exhausted d1211 d1212 Soar> </pre> <p style="text-align: right;"><i>Scrolling event 3 (2 windows up): <u>t646</u></i></p>	<pre> <u>t639</u> t639 where am i pgs^M t640 t641 s15 t642 now has t643 an utterance t644 model object t645 </pre> <p style="text-align: right;">t646 ^[v t647 u something ^[v</p>	<pre> 378 O: 02457 (display print-stack (t639)) 379 O: 02455 (comprehend current-context :sno) 380 ==>S: S65 (operator no-change) 381 O: 02459 (attend t639 constant148) 382 O: 02464 (comprehend lhs :part) 383 O: 02466 (fixate :current-context s-construct) 384 O: 02481 (comprehend s-construct :actual-context) 385 ==>S: S66 (operator no-change) 386 O: 02495 (comprehend lhs :part) 387 O: 02497 (fixate state s15 :state-id) 388 O: 02513 (comprehend s15 :imp-obj state) 389 O: 02509 (comprehend apply-chunks :imp-obj sp) 390 O: 02505 (comprehend u-model :imp-obj object) 391 O: 02511 (comprehend u-model) 392 ==>S: S67 (operator no-change) 393 O: 02529 (imagine :referent) 391 O: 02511 (comprehend u-model) 392 ==>S: S67 (operator no-change) 393 O: 02529 (imagine :referent) 394 O: 02545 (display scroll:to-object u-model (t646)) . (Compound command) </pre>
---	--	---

<p>Visible after t646 (scrolling event 3): d1064 to d1124</p> <pre> d1064 (<obj> ^referent <r> + ^referent <r> &) d1065 (<r> ^referent-of <obj> + ^type s-model +)) d1066 d1067 d1068 Soar> p o25 (t245) d1069 (O25 ^name create-referent ^for U20) d1070 d1071 Soar> p u20 (t252) d1072 (U20 ^left-edge W8 ^right-edge W13 ^bar-level max ^word-id W13 ^category n d1073 ^annotation specified ^empty-node E15 ^spec U14 ^zero-head U15 d1074 ^head U17) d1075 d1076 Soar> pgs (t323) ... d1093 Soar> run 1 (t364) ... d1097 Soar> p chunk-128 (t373) ... d1112 Soar> p chunk-129 (t503) d1113 (sp chunk-129 d1114 :chunk d1115 (goal <g1> ^state <s1> ^operator <o1>) d1116 (<s1> ^name initial-state ^assigners <al>) d1117 (<o1> ^name s-constructor16 ^conjunction-symbol conjunct d1118 ^type s-model-constructor) d1119 (<al> ^n <n2>) d1120 (<n2> ^max <n1>) d1121 (<n1> ^head <u1>) d1122 (<u1> -^referent <r*1>) d1123 --> d1124 (<o1> ^conjunction-symbol conjunct - ^conjunction-symbol <c1> +)) Goto-prompt: t649 </pre>	<pre> t648 u20 t649 let's look at u20 ^[>p__u20^M </pre>	<pre> 395 O: O2511 (comprehend u-model) 396 ==>S: S71 (operator no-change) 397 O: O2583 (fixate u20 :id-of-imagined-att) Goto-prompt disregarded </pre>
--	---	---

<p><i>Visible after t649 (goto-prompt): d1182 to d1212</i> <i>Top of window until t720</i> d1182 Retractions: d1183 chunk-132 d1184 s-construct*add-property*touch-conjunct-symbol d1185 s-construct*propose*add-property*head-noun*cop d1186 d1187 Soar> run 1 (t621) d1188 d1189 Soar> ms (t629) d1190 Assertions: d1191 Retractions: d1192 d1193 Soar> run 1 (t630) d1194 79: O: O26 exhausted d1195 Soar> pgs (t639) d1196 : ==>G: G1 d1197 : P: P1 top-ps d1198 : S: S1 d1199 : O: O2 comprehend-input d1200 : ==>G: G2 operator no-change d1201 : P: P2 comprehension d1202 : S: S4 d1203 : ==>G: G15 state no-change d1204 : P: P68 create-operator d1205 : S: S15 d1206 : O: O24 s-constructor16 d1207 : ==>G: G16 operator no-change d1208 : P: P85 s-construct d1209 : S: S15 d1210 : O: O26 exhausted d1211 d1212 Soar> p u20 (t649) <i>Top of window from t720 until t754</i> d1213 (U20 ^referent R9 ^left-edge W8 ^right-edge W13 ^bar-level max ^word-id W13 d1214 ^category n ^annotation specified ^empty-node E15 ^spec U14 d1215 ^zero-head U15 ^head U17) d1216 d1217 Soar></p>	<p><u>t649</u> let's look at u20 ^ [>p__u20^M t650 t651 right t652 which has referent r t653 9</p>	<p>398 O: O2587 (display print-object-fresh u20 (t649)) 399 O: O2511 (comprehend u-model) 400 ==>S: S73 (operator no-change) 401 O: O2591 (attend t649 constant157) 402 O: O2596 (comprehend lhs :part) 403 O: O2612 (fixate referent r9 :augmentation) 404 O: O2630 (comprehend referent :objects-att)</p>
---	--	--

d1217 Soar> p r9	<u>t654</u>	t654	p__r9^M	405 O: 02632 (display print-object r9 (t654))
d1218 (R9 ^properties P87 ^properties P88 ^referent-of U20 ^type s-model)		t655 it's sitting		406 O: 02630 (comprehend referent :objects-att)
d1219		t656 there		407 ==>S: S74 (operator no-change)
		t657 it		408 O: 02635 (attend t654 constant159)
		t658 has two properties		409 O: 02655 (fixate properties p88 :augmentation)
		t659 everything looks		410 O: 02653 (fixate referent-of u20 :augmentation)
		t660 good		411 O: 02651 (fixate head u17 :augmentation)
		t661		412 O: 02654 (fixate properties p87 :augmentation)
		t662 we're about to come		413 O: 02674 (comprehend properties :objects-att)
		t663		414 ==>S: S75 (operator no-change)
		t664 out of the s		415 O: 02699 (fixate referent r9 :augmentation)
		t665 constructor space		416 O: 02698 (fixate zero-head u15 :augmentation)
		t666 to build a pro		417 O: 02697 (fixate spec u14 :augmentation)
		t667 posal		418 O: 02696 (fixate empty-node e15 :augmentation)
		t668		419 O: 02695 (fixate right-edge w13 :augmentation)
		t669		420 O: 02693 (fixate left-edge w8 :augmentation)
		t670		421 O: 02691 (fixate :two properties)
		t671		422 O: 02692 (fixate nil :assertion)
		t672		423 O: 02714 (comprehend nil :assertion)
		t673 here's the big		424 ==>S: S76 (operator no-change)
		t674 question		425 O: 02739 (fixate properties p88 :augmentation)
		t675		426 O: 02737 (fixate referent-of u20 :augmentation)
		t676 do i want to let it		427 O: 02736 (fixate head u17 :augmentation)
		t677 return this		428 O: 02738 (fixate properties p87 :augmentation)
		t678 thing		429 O: 02719 (comprehend properties (previous-goal))
		t679		430 O: 02735 (fixate o26 :selected-id)
		t680		431 O: 02734 (fixate exhausted :selected)
		t681 how is it going to return		432 O: 02764 (comprehend exhausted :selected-op)
		t682 this thing		
		t683		
		t684		
d1220 Soar> ms	<u>t685</u>	t685	ms^M	433 O: 02766 (display match-set:after-selection (t685))
d1221 Assertions:		t686 implement exhaust		434 O: 02764 (comprehend exhausted :selected-op)
d1222 s-construct*implement*exhausted		t687 ed what does that do		435 ==>S: S78 (operator no-change)
d1223 Retractions:		t688		436 O: 02770 (attend t685 constant172)
d1224 s-construct*terminate*add-property		t689		437 O: 02794 (fixate implement-exhausted :assertion)
d1225 s-construct*add-property		t690		438 O: 02799 (comprehend implement-exhausted :imp-obj assertion)
d1226				
d1227 Soar>				439 O: 02797 (comprehend implement-exhausted :assertion)

d1227 Soar> p s-construct*implement*exhausted t691	t691 p__s-construct*implement*exhausted^M	440 O: 02811 (display print-sp:applies-operator implement-exhausted (t691))
d1228 (sp s-construct*implement*exhausted	t692	441 O: 02797 (comprehend implement-exhausted :assertion)
d1229 (goal <g> ^operator <o> ^problem-space <p> ^state <s> ^object <sg>)	t693	442 ==>S: S81 (operator no-change)
d1230 (<o> ^name exhausted)	t694	443 O: 02813 (attend t691 constant176)
d1231 (<p> ^name s-construct)	t695	444 O: 02838 (comprehend exhausted :imp-obj operator*)
d1232 (<sg> ^problem-space <p*1> ^operator <o*1> ^state <ss>)	t696 ok it	445 O: 02821 (comprehend return-new-pointer (high-level-goal))
d1233 (<p*1> ^name create-operator)	t697 puts this marvel	446 O: 02818 (comprehend rhs :part)
d1234 (<o*1> ^name <name>)	t698 ous construction	447 O: 02816 (comprehend lhs :part)
d1235 -->	t699 done	448 O: 02845 (fixate :action superstate construction-done)
d1236 (<g> ^operator <o>)	t700 flag	449 O: 02857 (comprehend superstate :superstate)
d1237 (<ss> ^annotation construction-done + ^annotation construction-done &))	t701 on	450 ==>S: S82 (operator no-change)
d1238	t702	451 O: 02879 (fixate superstate s15 :superstate-id)
d1239	t703	
	t704	
	t705 the	
	t706 superstate	
	t707	
	t708 course	
	t709 it's the same	
	t710 as the state but	
	t711 won't build a	
	t712 will build a chunk	
	t713	
	t714 um ok	
	t715 so it's going to slap	
	t716	
	t717 construction	
	t718 done on	
	t719 s15	
d1240 Soar> run 1 t720	t720	452 O: 02901 (display run:action-and-print (t720)) . (Compound command)
d1241	t721 un__1^Mp__s15^M	453 O: 02857 (comprehend superstate :superstate)
d1242 Build: chunk-134	t722	454 ==>S: S83 (operator no-change)
d1243 Soar> p s15	t723	455 O: 02912 (attend t720 constant179)
d1244 (S15 ^annotation construction-done ^annotation u-constructor-applied	t724	456 O: 02920 (fixate :annotations)
d1245 ^annotation u-model-success ^name initial-state ^ordering-info O3	t725	
d1246 ^adjacency-info A3 ^for-formatting F4 ^assigners A64 ^receivers R7	t726 s con	
d1247 ^assigners2 A65 ^receivers2 R8)	t727 struction done	
d1248	t728 u constructor applied	
d1249 Soar>	t729 u model success	

d1249 Soar> ms	t730	t730 this	ms^M	457 O: 02953 (display match-set:asserted-sp (t730))
d1250 Assertions:		t731 is going to let me		458 O: 02857 (comprehend superstate :superstate)
d1251 create-operator*propose-return-operator		t732 propose the return		459 ==>S: S85 (operator no-change)
<i>Top of window from t754 until t770</i>		t733 operator		460 O: 02961 (attend t730 constant180)
d1252 create-operator*terminate-s-model-		t734 where		461 O: 02965 (comprehend lhs :part)
constructor		t735		462 O: 02989 (fixate propose-return-operator :assertion)
d1253 Retractions:		t736		463 O: 03012 (comprehend propose-return-operator :assertion)
d1254		t737		464 O: 03016 (comprehend proposal-context :sno)
d1255 Soar> pgs	t738	t738	pgs^M	465 O: 03020 (display print-stack (t738))
d1256 : ==>G: G1		t739		466 O: 03016 (comprehend proposal-context :sno)
d1257 : P: P1 top-ps		t740 create		467 ==>S: S87 (operator no-change)
d1258 : S: S1		t741 in the create operator		468 O: 03022 (attend t738 constant184)
d1259 : O: O2 comprehend-input		t742 space		469 O: 03059 (fixate :proposal-context create-operator)
d1260 : ==>G: G2 operator no-change		t743		470 O: 03052 (fixate receivers2 r8 :augmentation)
d1261 : P: P2 comprehension		t744		471 O: 03051 (fixate assigners2 a65 :augmentation)
d1262 : S: S4		t745 propose		472 O: 03050 (fixate receivers r7 :augmentation)
d1263 : ==>G: G15 state no-change		t746 return		473 O: 03049 (fixate assigners a64 :augmentation)
d1264 : P: P68 create-operator		t747 operator		474 O: 03048 (fixate for-formatting f4 :augmentation)
d1265 : S: S15		t748		475 O: 03047 (fixate adjacency-info a3 :augmentation)
d1266 : O: O24 s-constructor16		t749		476 O: 03046 (fixate ordering-info o3 :augmentation)
d1267 : ==>G: G16 operator no-change		t750 i think we're getting		477 O: 03030 (fixate :current-context s-construct)
d1268 : P: P85 s-construct		t751 close to the right		478 O: 03074 (comprehend s-construct :actual-context)
d1269 : S: S15		t752 place		479 ==>S: S88 (operator no-change)
d1270 : O: O26 exhausted		t753 terminate s-model		480 O: 03087 (comprehend lhs :part)
d1271				481 O: 03091 (fixate state s15 :state-id)
d1272 Soar>				482 O: 03138 (comprehend s15 :imp-obj state)
				483 O: 03129 (comprehend apply-chunks :imp-obj sp)
				484 O: 03125 (comprehend proposal-build :imp-obj action)
				485 O: 03142 (comprehend return-new-pointer
				(high-level-goal))
				486 O: 03140 (comprehend return-new-pointer :high-lev-goal)
				487 ==>S: S89 (operator no-change)
				488 O: 03152 (comprehend lhs :part)
				489 O: 03181 (fixate propose-return-operator :assertion)
				490 O: 03196 (comprehend propose-return-operator
				:imp-obj assertion)
				491 O: 03182 (fixate terminate-s-model-constructor
				:assertion)
				492 O: 03202 (comprehend terminate-s-model-constructor
				:assertion-pay-attn)

<pre>d1272 Soar> p create-operator*terminate- t754 s-model-constructor d1273 (sp create-operator*terminate- s-model-constructor d1274 (goal <g> ^state <s*1> ^operator <o> ^problem-space <p>) d1275 (<s*1> ^annotation construction-done) d1276 (<o> ^type s-model-constructor) d1277 (<p> ^name create-operator) d1278 --> d1279 (<g> ^operator <o> d1280 d1281 d1282 Soar></pre> <p style="text-align: right;"><i>Scrolling event 4 (3 lines up): t770</i></p>	<pre>t754 constructor (mouse copy d1252) t755 t756 see what t757 that's doing t758 t759 p__create-operat t760 oh shut up or*terminate-s- model-constructor (mouse paste) t761 t762 t763 t764 gnu's doing t765 things i don't know what ^M t766 it's doing t767 ok t768 fine that t769 looked for the construc t770 tion done ^[z t771 to put out the recon ^[z t772 sider ^[z</pre>	<pre>493 O: O3206 (display print-sp:paying-attention terminate-s-model-constructor (t754)) 494 O: O3202 (comprehend terminate-s-model-constructor :assertion-pay-attn) 495 ==>S: S92 (operator no-change) 496 O: O3208 (attend t754 constant193) 497 O: O3213 (comprehend lhs :part) 498 O: O3218 (fixate bound-by type :bind-target) 499 O: O3211 (comprehend rhs :part) 500 O: O3241 (fixate :action s-model-constructor reconsider) 501 O: O3250 (comprehend reconsider :imp-obj action) 502 O: O3238 (fixate :condition problem-space create-operator) 503 O: O3255 (comprehend create-operator :imp-obj condition) 504 O: O3237 (fixate :condition type s-model-constructor) 505 O: O3239 (comprehend type :imp-obj condition) 506 O: O3236 (fixate :condition annotation construction-done) 507 O: O3263 (comprehend construction-done :imp-obj condition) 508 O: O3225 (comprehend terminate-s-model-constructor :imp-obj sp) 509 O: O3223 (comprehend propose-return-operator :imp-obj sp) 510 O: O3227 (comprehend propose-return-operator :assertion-fix-recent) 511 O: O3270 (display scroll:to-sp propose-return-operator (t770)) . . (Compound command)</pre>
--	--	---

<pre> d1255 Soar> pgs t738 ... d1266 : O: 024 s-constructor16 d1267 : ==>G: G16 operator no-change d1268 : P: P85 s-construct d1269 : S: S15 d1270 : O: 026 exhausted d1271 d1272 Soar> p create-operator*terminate- t754 s-model-constructor ... d1282 Soar> p create-operator*propose- t774 return-operator d1283 (sp create-operator*propose-return-operator d1284 (goal <g> ^state <s*1> ^problem-space <p> ^operator <op>) d1285 (<s*1> ^annotation construction-done) d1286 (<p> ^name create-operator) d1287 (<op> ^type { << u-model-constructor s-model-constructor >> <t*1> }) d1288 --> d1289 (<g> ^operator <o> + ^operator <o> >) d1290 (<o> ^name return-operator + ^new-operator <op> +)) d1291 d1292 d1293 Soar> run 1 t811 d1294 d1295 Soar> ms d1296 Assertions: d1297 Retractions: d1298 d1299 Soar> run 1 d1300 80: O: 029 return-operator d1301 Soar> p o29 t817 d1302 (029 ^name return-operator ^new-operator 024) d1303 d1304 Soar> </pre>	<pre> t797 huh? t798 where's it picking that t799 up from t800 t801 t802 operator t803 op t804 t805 t806 oh i see t807 so it's going to put s-con t808 structor 16 t809 out there t810 so let's do that t811 run__1^Mms^Mrun__1^M t812 t813 t814 there's the re t815 turn operator t816 so far so t817 good t818 t819 ^?p__o29^M t820 t821 ok t822 it t823 says t824 t825 new operator t826 o24 which t827 is in fact s-con t828 structor 16 </pre>	<pre> 532 O: 03424 (comprehend operator* :op-cond) 533 ==>S: S99 (operator no-change) 534 O: 03433 (comprehend lhs :part) 535 O: 03455 (fixate :condition operator* op) 536 O: 03461 (comprehend op :new-operator) 537 ==>S: S100 (operator no-change) 538 O: 03473 (comprehend lhs :part) 539 O: 03478 (fixate condition operator* op :bind-param) 540 O: 03497 (display run:to-expected-op (t811)) (Compound command) 541 O: 03461 (comprehend op :new-operator) 542 ==>S: S101 (operator no-change) 543 O: 03502 (attend t811 constant214) 544 O: 03505 (comprehend lhs :part) 545 O: 03512 (fixate o29 :selected-id) 546 O: 03531 (display print-op (t817)) 547 O: 03461 (comprehend op :new-operator) 548 ==>S: S102 (operator no-change) 549 O: 03536 (attend t817 constant215) 550 O: 03539 (comprehend lhs :part) 551 O: 03548 (fixate new-operator o24 :augmentation) 552 O: 03567 (comprehend new-operator :objects-att) 553 ==>S: S103 (operator no-change) 554 O: 03577 (fixate o24 operator* s-constructor16 :bind-op-id) 547 O: 03461 (comprehend op :new-operator) 548 ==>S: S102 (operator no-change) 549 O: 03536 (attend t817 constant215) </pre>
--	---	---

<p style="text-align: center;"><i>Scrolling event 5 (3 windows up): <u>t845</u></i></p>	<p>t829 t830 t831 t832 t833 t834 eeuuu t835 t836 t837 rats t838 t839 t840 t841 t842 t843 i think all those chunks t844 i built t845 ^[v t846 test for opera t847 tor t848 t849 six ^[v t850 uh t851 s whatever it is t852 i think t853 they test for the s-cons t854 tructor ^[v</p>	<p>550 O: 03539 (comprehend lhs :part) 551 O: 03548 (fixate new-operator o24 :augmentation) 552 O: 03567 (comprehend new-operator :objects-att) 553 ==>S: S103 (operator no-change) 554 O: 03577 (fixate o24 operator* s-constructor16 :bind-op-id) 555 O: 03580 (comprehend operator* :imp-obj condition) 556 O: 03586 (comprehend lhs :part) 557 O: 03584 (comprehend lhs :sp-parts) 558 ==>S: S104 (operator no-change) 559 O: 03610 (comprehend lhs :part) 560 O: 03621 (fixate new-operator o24 :augmentation) 561 O: 03608 (comprehend new-operator (previous-goal)) 562 O: 03637 (fixate :condition problem-space create-operator) 563 O: 03642 (comprehend create-operator :imp-obj condition) 564 O: 03636 (fixate :condition type s-model-constructor) 565 O: 03650 (comprehend s-model-constructor :imp-obj condition) 566 O: 03648 (imagine condition s-constructor16) 567 O: 03656 (comprehend s-constructor16 :recalled-condition) 568 O: 03660 (display scroll:to-sp s-constructor16 (t845)) . (Compound . command)</p>
---	--	--

<p><i>Visible after t845 (scrolling event 5): d1072 to d1132</i></p> <pre> d1072 (U20 ^left-edge W8 ^right-edge W13 ... d1076 Soar> pgs (t323) ... d1093 Soar> run 1 (t364) d1094 d1095 Build: chunk-128 d1096 Build: chunk-129 d1097 Soar> p chunk-128 (t373) d1098 (sp chunk-128 d1099 :chunk d1100 (goal <g1> ^operator ^state <s1>) d1101 (^name s-constructor16 ^type s-model-constructor) d1102 (<s1> ^assigners <al>) d1103 (<al> ^n <n2>) d1104 (<n2> ^max <n1>) d1105 (<n1> ^head) d1106 (-^referent <r*1>) d1107 --> d1108 (^referent <r1> + ^referent <r1> &) d1109 (<r1> ^referent-of + ^type s-model +)) d1110 d1111 d1112 Soar> p chunk-129 (t503) d1113 (sp chunk-129 d1114 :chunk d1115 (goal <g1> ^state <s1> ^operator) d1116 (<s1> ^name initial-state ^assigners <al>) d1117 (^name s-constructor16 ^conjunct-symbol conjunct ^type s-model-constructor) d1118 d1119 (<al> ^n <n2>) d1120 (<n2> ^max <n1>) d1121 (<n1> ^head) d1122 (-^referent <r*1>) d1123 --> d1124 (^conjunct-symbol conjunct - ^conjunct-symbol <c1> +)) d1125 d1126 d1127 Soar> ms (t513) d1128 Assertions: ... d1132 Retractions: </pre>	<pre> t855 t856 let's t857 see shall we t858 yeah s construc t859 tor 16 t860 there it is </pre>	<pre> 569 0: 03656 (comprehend s-constructor16 :recalled-condition) End of model </pre>
--	--	--

E.2. Table of model commands mapped to programmer's keystrokes

The table below maps model commands (Figure 26, p. 70) to programmer keystrokes and when they occur. It also contains a glossary of keystrokes that invoked Emacs commands.

unit	commands	onset of keystroke sequence	programmer keystrokes
v	v v slips/disregarded/missed v v model commands	v	
1	match-set after-selection	t225 ms^M	
2	print-sp create-referent	t236 p__s-construct*create-referent^M	
3	print-operator o25	t245 p__o25^M	
4	print-object u20	t252 p__u20.^M	
5	print-stack	t323 pgs^M	
6	run where-was-i	t364 run__1^Mm	
	s	t371 ^?	<i>erases preceding "m"</i>
7	print-chunk chunk-128	t373 p__chunk-128^M	
8	scroll to-sp	t431 ^[v	scrolling event 1
9	scroll to-state	t445 ^V	scrolling event 2
	d	t499 ^[>	
10	print-chunk chunk-129	t503 p__chunk-...129..^M	
11	match-set asserted-sp	t513 ms^M	
12	run to-builds	t552 run__1^M	
13	match-set after-builds	t564 ms^M	
	m	t565 run__1^M	
14	match-set after-selection	t575 ms^M	
15	run to-builds	t582 run__1^M	
16	match-set after-builds	t587 ms..^M	
	1	t593 run...1^M.ms^M..run.{x14}__1^M	
	m	t616 run__1^M	
17	match-set after-selection	t618 ms^M	
	2	t621 run...__...1^M.ms^M.run__1^M	
18	print-stack	t639 pgs^M	
	3	t646 ^[v.^[v	scrolling event 3
	d	t649 ^[>	
19	print-object-fresh u20	t649 p__u20^M	
20	print-object r9	t654 p__r9^M	
21	match-set after-selection	t685 ms^M	
22	print-sp impl't-exhausted	t691 p__s-construct*implement*exhausted^M	
	4	t720 r.un__1^Mmp__s15^M	
23	match-set asserted-sp	t730 ms^M	
24	print-stack	t738 pgs^M	
25	print-sp term-s-model-con	t754 {mc}....p__{mp}^M	
	5	t770 ^[z.^[z.^[z	scrolling event 4
26	print propose-retrn-op	t774 {mc}..p__{mp}^M	
	6	t811 run__1^Mms^Mrun__1^M	
	s	t817 o..^?	
27	print-operator o29	t819 p__o29^M	
	7	t845 ^[v....^[v.....^[v	scrolling event 5

Meta- characters used above:	.	One-second latency between keystrokes
	.{xN}	N-second gap between keystrokes
	—	Space (" ")
	{mc}	Mouse copy
	{mp}	Mouse paste
Keystrokes that invoke GNU Emacs commands:	^[v	scroll window up (ESC v)
	^[z	scroll up one line (ESC z)
	^V	scroll window down (Control-v)
	^[>	goto prompt (ESC >)
	^M	return
	^?	delete

Appendix F

Rule-level trace of the model

The trace in this appendix covers the life of the model, and shows all operator selections, rule firings, and rule encodings. Decision cycles (DCs) and operators are as they were in the operator traces in Appendix D.

Rule firings are grouped into "paragraphs" that correspond to firing cycles (also known as *elaboration cycles* (Laird et al., 1993)). For example, below DC 0 are 2 firing cycles, each with 2 firings. Where a rule fires multiple times in one cycle, the instances compressed and counted. For example, below DC 3 (second cycle) the rule `po*attend` fires twice, so is followed by `x2`. At the end of a firing cycle, Soar carries out the WM changes proposed by the firings that occurred in that cycle. These WM changes may cause a new firing cycle, but if not Soar makes a new decision (incrementing the DC count).

Encoded rules are emphasized by being underlined at the time that Soar builds them, and **bolded** when they fire. For example, below DC 4 (first cycle) Soar first encodes `chunk-1` and then fires it immediately afterwards. An encoded rule is added to LTM immediately, and fires immediately if its conditions match WM elements other than those from which the rule was built. (Laird et al., 1993)

The prefix of a rule name describes the rule's general role. "po*" proposes an operator; "p*" indicates a preference for selecting a proposed operator; "ao*" applies a selected operator, changing the state according to the operator's specifications; and "a*" augments a data structure, changing the structure based not on an operator but on elements of the data structure itself. The prefix "f:" is a special case, denoting a fact. Appendix C translates rule names to page numbers in the code.

```

0 S: S1
  ao*display*po*init a*topstate*create-display-wm-dp-time-dummy
  a*display*tinit p*generic*indifferent
1 O: O1 (display tinit (tinit))
  p*generic*terminate-and-reject ao*display*tinit ao*display*emulator*first
  p*generic*reject d*print*tinit po*comprehend*init
  a*topstate*clean-up-old-comprehends-and-displays ao*goal-select*comprehend*create-token
  p*generic*indifferent
2 O: O2 (comprehend cinit)
  ao*comprehend*applied*first ao*comprehend*create-dp-on-om
  a*state*applied-newer
3 ==>S: S2 (operator no-change)
  a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend x2 a*state*fixate-meta-attributes
  a*substate*initialize-goal-set a*state*important-objects a*substate*create-time

```

po*attend x2
 p*generic*indifferent x2
 4 O: 05 (attend tinit constant2)
 ao*attend chunk-1 **chunk-1** ao*attend*mark-locally p*generic*terminate-and-reject
 a*dp*unpack-applied-om x4 p*generic*reject
 a*wm*newest-from-not-newest
 po*fixate*selected-operator po*fixate*selected-id po*fixate*assertion
 a*fixate*newest x3 p*generic*indifferent x3
 p*fixate*newest*interleave-best x3
 5 O: 08 (fixate nil :assertion)
 ao*fixate chunk-2 p*generic*terminate-and-reject ao*substate*count-first
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-3 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*comprehend*assertion*real po*probe*with-important-object
 a*goal-select*proposed-during ao*goal-select*comprehend*create-token p*probe*new-important-object*best
 p*generic*indifferent x2
 6 O: 011 (comprehend nil :imp-obj assertion)
 p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*substate*count-second
 ao*comprehend*applied*first
 p*probe*repeated-goal*worst p*generic*reject ao*goal-select*new-goal*probe chunk-4
 a*state*applied-newer
 7 O: 09 (comprehend nil :assertion)
 ao*comprehend*applied*second ao*comprehend*create-dp-on-om a*goal-select*proposed-during
 a*state*applied-newer
 8 ==>S: S3 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*fixate*selected-operator po*fixate*selected-id po*probe*with-previous-goal
 po*probe*with-important-object p*attend*old-regions*reject p*generic*indifferent
 a*fixate*newest x2 p*probe*repeated-goal*worst p*generic*indifferent x4
 p*fixate*newest*interleave-best x2
 9 O: 016 (comprehend cinit |(previous-goal)|)
 p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 ao*substate*count-first
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 10 O: 015 (fixate o25 :selected-id)
 ao*fixate chunk-5 ao*fixate chunk-6 ao*fixate chunk-7 p*generic*terminate-and-reject
 ao*substate*count-second
 a*wm*unpack-applied-om x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-8 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-9 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-10 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x3
 11 O: 014 (fixate create-referent :selected)
 ao*fixate chunk-11 ao*fixate chunk-12 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x2 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-13 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-14 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x2

- po*probe*with-important-object po*comprehend*selected-operator x3
p*probe*new-important-object*best ao*goal-select*comprehend*create-token a*goal-select*proposed-during
ao*goal-select*comprehend*create-token a*goal-select*proposed-during
ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent x4
ao*goal-select*new-goal*fixate/imagine chunk-15 ao*goal-select*new-goal*fixate/imagine x2
chunk-17 ao*goal-select*new-goal*fixate/imagine x2 chunk-19
ao*goal-select*new-goal*fixate/imagine
- 12 O: 026 (comprehend create-referent :selected-op)
ao*comprehend*applied f:high-level-goal-cues f:operator ao*comprehend*create-dp-on-om
po*display*match-set*after-selection
a*state*applied-newer a*wm*unpack-applied-om x3 p*display*dunk-comprehend a*display*t225
p*generic*indifferent
chunk-1 x2 po*comprehend*high-level-goal
a*dp*unpack-applied-om x4 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
p*generic*indifferent
a*wm*newest-from-not-newest
- 13 O: 028 (display match-set:after-selection (t225))
ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
a*wm*unpack-applied-om p*generic*reject ao*display*t225
d*tinit*t225
- 14 O: 026 (comprehend create-referent :selected-op)
f:high-level-goal-cues f:operator po*display*match-set*after-selection
p*display*reject-duplicates p*generic*indifferent
- 15 ==>S: S5 (operator no-change)
a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
po*attend
p*generic*indifferent
- 16 O: 032 (attend t225 constant9)
ao*attend chunk-21 ao*attend*mark-locally p*generic*terminate-and-reject
a*dp*unpack-applied-om x3 p*generic*reject
a*wm*newest-from-not-newest
ao*attend*previous-not-newest chunk-22 po*probe*with-high-level-goal po*fixate*assertion x3
po*probe*with-previous-goal po*fixate*argument po*probe*with-important-object
a*dp*unpack-applied-om p*probe*repeated-goal*worst p*generic*indifferent x7
a*fixate*newest x4
p*fixate*newest*interleave-best x2
p*fixate*top-down
- 17 O: 040 (fixate create-referent :argument cop)
ao*fixate chunk-23 ao*fixate chunk-24 p*fixate*interleave-best x2
p*generic*terminate-and-reject ao*substate*count-first p*probe*new-attribute*best
a*wm*unpack-applied-om x2 p*generic*reject
ao*fixate*unpack-fixation-object chunk-25 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-26 ao*comprehend*unpack-fixation-object
a*wm*unpack-applied-om x2
- 18 O: 036 (fixate apply-create-referent :assertion)
ao*fixate chunk-27 p*generic*terminate-and-reject ao*substate*count-second
a*wm*unpack-applied-om p*generic*reject
ao*fixate*unpack-fixation-object chunk-28 ao*comprehend*unpack-fixation-object

a*wm*unpack-applied-om
 po*comprehend*assertion*real po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*generic*indifferent x2
 ao*goal-select*new-goal*fixate/imagine chunk-29

19 O: 043 (comprehend apply-create-referent :assertion)
 ao*comprehend*applied f:apply-sps f:sp ao*comprehend*create-dp-on-om
 a*state*applied-newer a*wm*unpack-applied-om x2 **chunk-22**
 f:sp-parts po*display*print-sp*applies-operator a*dp*unpack-applied-om
 a*wm*unpack-applied-om x2 p*display*dunk-comprehend a*display*t236 p*generic*indifferent
 po*comprehend*sp-parts x2
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent x2

20 O: 047 (display print-sp:applies-operator apply-create-referent (t236))
 ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator ao*display*t225
 a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
 a*wm*unpack-applied-om p*generic*reject ao*display*print*t236
 d*print*t236

21 O: 043 (comprehend apply-create-referent :assertion)
 f:apply-sps f:sp f:sp-parts po*display*print-sp*applies-operator **chunk-22**
 p*display*reject-duplicates p*generic*indifferent

22 ==>S: S8 (operator no-change)
 a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
 a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
 po*attend
 p*generic*indifferent

23 O: 053 (attend t236 constant13)
 ao*attend chunk-30 ao*attend*mark-locally p*generic*terminate-and-reject
 a*dp*unpack-applied-om x3 p*generic*reject
 a*wm*newest-from-not-newest
 ao*attend*previous-not-newest chunk-31 po*probe*with-high-level-goal
 po*fixate*binding-attribute*target po*fixate*selected-operator po*fixate*selected-id po*fixate*assertion
 x2 po*probe*with-previous-goal po*probe*with-part x2 po*fixate*binding-attribute*param
 po*probe*with-important-object x3
 a*dp*unpack-applied-om p*probe*rhs-better-when-apply-sp p*probe*lhs-best p*probe*repeated-goal*worst
 x2 p*generic*indifferent
 a*fixate*newest x6
 p*fixate*newest*interleave-best x2

24 O: 068 (comprehend create-referent :imp-obj operator*)
 p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:high-level-goal-cues f:operator ao*substate*count-first
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
 p*probe*new-high-level-goal*best

25 O: 067 (fixate condition for obj :bind-param)
 ao*fixate chunk-32 ao*fixate chunk-33 ao*fixate chunk-34 ao*fixate chunk-35
 ao*fixate chunk-36 ao*fixate chunk-37 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om x6 p*generic*reject p*probe*rhs-when-sp
 ao*fixate*unpack-fixation-object chunk-38 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-39 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-40 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-41 ao*comprehend*unpack-fixation-object

ao*fixate*unpack-fixation-object chunk-42 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-43 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x6
 po*probe*with-important-object
 p*probe*new-important-object*best p*generic*indifferent
 26 O: 063 (comprehend rhs :part)
 ao*goal-select*new-goal*probe chunk-44 ao*probe*goal p*generic*terminate-and-reject
 ao*comprehend*applied*second f:high-level-goal-cues
 po*fixate*action x3 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 p*probe*new-high-level-goal*best ao*probe*unpack-probe-om-to-superop-om chunk-45
 a*fixate*newest x3 a*wm*unpack-applied-om p*generic*indifferent x3
 p*fixate*newest*interleave-best x3 po*probe*with-high-level-goal
 p*fixate*interleave-best p*fixate*bottom-up x2 p*fixate*interleave-best x2
 p*probe*new-high-level-goal*best p*generic*indifferent
 27 O: 050 (comprehend rhs :sp-parts)
 ao*comprehend*applied f:high-level-goal-cues ao*comprehend*create-dp-on-om
 a*state*applied-newer a*wm*unpack-applied-om x2 **chunk-22 chunk-31**
chunk-21 chunk-1 x2 a*dp*unpack-applied-om x2
 a*dp*unpack-applied-om x7
chunk-22 chunk-31
 28 ==>S: S9 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*probe*with-high-level-goal po*fixate*selected-operator po*fixate*selected-id po*fixate*assertion
 x3 po*probe*with-previous-goal po*probe*with-part x2 po*probe*with-important-object x2
 po*fixate*action x3 p*attend*old-regions*reject p*generic*indifferent
 a*fixate*newest x5 p*probe*repeated-goal*worst p*probe*lhs-best a*fixate*newest x3
 p*generic*indifferent
 p*fixate*newest*interleave-best x3
 p*fixate*bottom-up x2
 29 O: 093 (comprehend lhs :part)
 p*fixate*interleave-best x3 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 po*fixate*condition x5 po*fixate*binding-context p*probe*repeated-goal*worst p*generic*reject
 a*state*applied-newer
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x5 p*fixate*interleave-best
 p*generic*indifferent x6
 a*fixate*dont-interleave-best p*fixate*newest*interleave-best a*fixate*dont-interleave-best
 p*fixate*newest*interleave-best x5
 p*fixate*interleave-best p*fixate*bottom-up p*fixate*interleave-best p*fixate*bottom-up
 p*fixate*interleave-best p*fixate*bottom-up
 30 O: 0101 (fixate :action referent-of obj)
 ao*fixate chunk-46 ao*fixate chunk-47 ao*fixate chunk-48 p*generic*terminate-and-reject
 ao*substate*count-second
 a*wm*unpack-applied-om x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-49 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-50 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x2
 po*probe*with-important-object x2
 p*probe*new-important-object*best p*probe*new-attribute*best p*probe*new-important-object*best
 p*generic*indifferent x2

- 31 O: O110 (comprehend obj :imp-obj action)
 p*fixate*interleave-best x6 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*second
 po*imagine*action x2
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer a*fixate*newest p*imagine*refract
 a*fixate*newest p*generic*indifferent x2
- 32 O: O100 (fixate :action type s-model)
 ao*fixate chunk-51 ao*fixate chunk-52 ao*fixate chunk-53 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-54 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-55 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-56 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x3
 po*probe*with-important-object x2
 p*probe*new-important-object*best p*probe*new-attribute*best p*probe*new-important-object*best
 p*generic*indifferent x2
- 33 O: O116 (comprehend s-model :imp-obj action)
 p*fixate*interleave-best x5 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 po*imagine*action x3
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer a*fixate*newest p*imagine*refract
 a*fixate*newest p*imagine*refract a*fixate*newest p*generic*indifferent x3
- 34 O: O99 (fixate :action obj referent)
 ao*fixate chunk-57 ao*fixate chunk-58 ao*fixate chunk-59 p*generic*terminate-and-reject
 p*probe*new-attribute*best p*probe*new-important-object*best
 a*wm*unpack-applied-om x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-60 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-61 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x2
 po*probe*with-important-object
 p*probe*new-important-object*best p*generic*indifferent
- 35 O: O121 (comprehend referent :imp-obj action)
 p*fixate*interleave-best x4 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 po*imagine*action x4
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer a*fixate*newest p*imagine*refract
 a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest
 p*generic*indifferent x4
- 36 O: O107 (fixate operator* for create-referent :bind-obj)
 ao*fixate chunk-62 ao*fixate chunk-63 ao*fixate chunk-64 p*generic*terminate-and-reject
 p*probe*new-attribute*best
 a*wm*unpack-applied-om x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-65 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-66 ao*comprehend*unpack-fixation-object **chunk-13**
 ao*fixate*unpack-fixation-object chunk-67 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x3
 po*probe*with-important-object
 p*probe*new-important-object*best p*generic*indifferent
- 37 O: O95 (comprehend for :imp-obj condition)
 p*fixate*interleave-best x3 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 f:recall-for-part-of-u-model po*imagine*action x4
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om
chunk-68 a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest
 p*imagine*refract a*fixate*newest p*generic*indifferent x4
 a*wm*unpack-applied-om
- 38 O: O106 (fixate :condition problem-space s-construct)

ao*fixate chunk-69 ao*fixate chunk-70 ao*fixate chunk-71 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-72 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-73 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-74 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x3
 po*probe*with-important-object x2
 p*probe*new-important-object*best p*probe*new-attribute*best p*probe*new-important-object*best
 p*generic*indifferent x2
 39 O: O135 (comprehend s-construct :imp-obj condition)
 p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 f:s-construct-builds-chunk po*imagine*action x4
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer a*state*new-important-object*best
 ao*probe*unpack-probe-om-to-superop-om chunk-75 ao*probe*unpack-probe-om-to-superop-om chunk-76
 a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract
 a*fixate*newest p*generic*indifferent x4
 a*wm*unpack-applied-om x2
 po*probe*with-important-object
 p*probe*retrieved-by-probe*best p*generic*indifferent
 40 O: O105 (fixate :condition operator* create-referent)
 ao*fixate chunk-77 ao*fixate chunk-78 ao*fixate chunk-79 p*generic*terminate-and-reject
 p*probe*new-important-object*best
 a*wm*unpack-applied-om x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-80 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-81 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x2
 po*comprehend*operator-condition po*probe*with-important-object x2
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*probe*new-attribute*best p*probe*new-important-object*best p*generic*indifferent x3
 ao*goal-select*new-goal*fixate/imagine chunk-82
 41 O: O143 (comprehend operator* :op-cond)
 ao*comprehend*applied f:operator-condition*part-of-lhs ao*comprehend*create-dp-on-om
 a*state*applied-newer a*wm*unpack-applied-om **chunk-31** **chunk-22**
chunk-40 ao*comprehend*unpack-fixation-object **chunk-30** a*dp*unpack-applied-om x2
 a*wm*unpack-applied-om a*dp*unpack-applied-om x3
chunk-31 a*wm*newest-from-not-newest
 42 ==>S: S10 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*probe*with-high-level-goal po*fixate*binding-attribute*target po*fixate*selected-operator
 po*fixate*selected-id po*fixate*assertion x3 po*probe*with-previous-goal po*probe*with-part
 po*fixate*binding-attribute*param po*probe*with-important-object p*attend*old-regions*reject
 p*generic*indifferent
 a*fixate*newest x2 p*fixate*operator-id a*fixate*newest x4 p*probe*lhs-best a*fixate*newest
 p*probe*repeated-goal*worst p*generic*indifferent
 p*fixate*newest*interleave-best x2
 43 O: O160 (comprehend lhs :part)
 p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 po*fixate*condition x3 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 ao*probe*unpack-probe-om-to-superop-om chunk-83
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest

p*fixate*invariant-feature*dont-interleave-best a*fixate*newest.x2 a*wm*unpack-applied-om
 p*generic*indifferent.x3
 a*fixate*dont-interleave-best p*fixate*newest*interleave-best a*fixate*dont-interleave-best
 p*fixate*newest*interleave-best.x2 po*comprehend*sp-parts po*probe*with-part
 p*fixate*interleave-best ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 p*generic*indifferent.x2
 44 O: 0162 (fixate condition for obj :bind-param)
 ao*fixate chunk-84 ao*fixate chunk-85 ao*fixate chunk-86 ao*fixate chunk-87
 ao*fixate chunk-88 ao*fixate chunk-89 p*generic*terminate-and-reject ao*substate*count-second
 p*probe*new-attribute*best.x2
 a*wm*unpack-applied-om.x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-90 ao*comprehend*unpack-fixation-object **chunk-38**
 ao*fixate*unpack-fixation-object chunk-91 ao*comprehend*unpack-fixation-object **chunk-39**
 ao*fixate*unpack-fixation-object chunk-92 ao*comprehend*unpack-fixation-object **chunk-41**
 ao*fixate*unpack-fixation-object chunk-93 ao*comprehend*unpack-fixation-object **chunk-42**
 ao*fixate*unpack-fixation-object chunk-94 ao*comprehend*unpack-fixation-object **chunk-43**
 a*wm*unpack-applied-om.x5
 po*probe*with-important-object
 p*probe*new-important-object*best p*generic*indifferent
 45 O: 0175 (comprehend referent-of :imp-obj action)
 p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*second
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 46 O: 0154 (fixate o25 :selected-id)
 ao*fixate chunk-95 ao*fixate chunk-96 ao*fixate chunk-97 p*fixate*interleave-best
 p*generic*terminate-and-reject
 a*wm*unpack-applied-om.x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-98 **chunk-8** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-99 **chunk-9** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-100 **chunk-10** ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om.x3
 po*display*print-operator.x2
 p*display*dunk-comprehend a*display*t245 p*display*dunk-comprehend a*display*t245 p*generic*indifferent
 x2
 47 O: 0195 (display print-op (t245))
 ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
 a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
 a*wm*unpack-applied-om p*generic*reject ao*display*print*t245
 d*print-id*t245
 48 O: 0143 (comprehend operator* :op-cond)
 f:operator-condition*part-of-lhs **chunk-22** **chunk-31** po*display*print-operator.x2
 p*display*reject-duplicates.x2 p*generic*indifferent.x2
 49 ==>S: S11 (operator no-change)
 a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
 a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
 po*attend
 p*generic*indifferent
 50 O: 0199 (attend t245 constant18)
 ao*attend chunk-101 ao*attend*mark-locally p*generic*terminate-and-reject
 a*dp*unpack-applied-om.x3 p*generic*reject
 a*wm*newest-from-not-newest
 ao*attend*previous-not-newest chunk-102 po*probe*with-high-level-goal po*fixate*selected-operator
 po*fixate*assertion.x3 po*fixate*augmentation po*probe*with-previous-goal po*probe*with-part.x2

po*probe*with-important-object
 a*dp*unpack-applied-om p*probe*lhs-best p*probe*repeated-goal*worst p*generic*indifferent
 a*fixate*newest x5
 p*fixate*newest*interleave-best
 51 O: O209 (comprehend lhs :part)
 p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 po*fixate*condition x3 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x2 p*generic*indifferent x3
 a*fixate*dont-interleave-best x2
 52 O: O206 (fixate for u20 :augmentation)
 ao*fixate chunk-103 ao*fixate chunk-104 ao*fixate chunk-105 ao*fixate chunk-106
 ao*fixate chunk-107 ao*fixate chunk-108 p*generic*terminate-and-reject
 ao*substate*count-second p*probe*new-attribute*best
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-109 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-110 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-111 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-112 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-113 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-114 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x6
 po*probe*with-attribute po*comprehend*objects-attribute x2
 p*probe*new-attribute*best ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent x3
 ao*goal-select*new-goal*fixate/imagine chunk-115 ao*goal-select*new-goal*fixate/imagine
chunk-116
 53 O: O244 (comprehend for :objects-att)
 ao*comprehend*applied f:recall-for-part-of-u-model ao*comprehend*create-dp-on-om po*display*print-object
 x2
 a*state*applied-newer a*wm*unpack-applied-om **chunk-31 chunk-22 chunk-102**
 p*display*dunk-comprehend a*display*t252 p*display*dunk-comprehend a*display*t252 p*generic*indifferent
 x2
chunk-21 chunk-1 x2 a*dp*unpack-applied-om x3
 a*dp*unpack-applied-om x7
chunk-31 chunk-22
 54 O: O247 (display print-object u20 (t252))
 ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
 a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
 a*wm*unpack-applied-om p*generic*reject ao*display*print*t252
 d*print-id*t252 p*generic*indifferent
 55 O: O244 (comprehend for :objects-att)
 f:recall-for-part-of-u-model **chunk-22 chunk-31 chunk-102** po*display*print-object
 p*display*reject-duplicates p*generic*indifferent
 56 ==>S: S12 (operator no-change)
 a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
 a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
 po*attend
 p*generic*indifferent
 57 O: O250 (attend t252 constant21)
 ao*attend chunk-117 ao*attend*mark-locally p*generic*terminate-and-reject

a*dp*unpack-applied-om x3 p*generic*reject
 a*wm*newest-from-not-newest
 ao*attend*previous-not-newest chunk-118 po*probe*with-attribute po*fixate*selected-operator
 po*fixate*assertion x3 po*fixate*augmentation x7 po*probe*with-previous-goal po*probe*with-part
 x2 po*probe*with-important-object
 a*dp*unpack-applied-om p*probe*repeated-goal*worst p*probe*lhs-best p*probe*repeated-goal*worst
 p*generic*indifferent
 a*fixate*newest
 p*fixate*newest*interleave-best x7
 58 O: 0266 (comprehend lhs :part)
 p*fixate*interleave-best x7 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 po*fixate*condition x5 po*fixate*binding-context p*probe*repeated-goal*worst p*generic*reject
 a*state*applied-newer
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x5 p*fixate*interleave-best
 p*generic*indifferent x6
 a*fixate*dont-interleave-best x2
 59 O: 0263 (fixate head u17 :augmentation)
 ao*fixate chunk-119 ao*fixate chunk-120 ao*fixate chunk-121 ao*fixate chunk-122
 ao*fixate chunk-123 ao*fixate chunk-124 p*generic*terminate-and-reject
 ao*substate*count-second
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-125 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-126 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-127 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-135 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-136 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-137 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x6
 po*probe*with-attribute
 p*probe*new-attribute*best p*generic*indifferent
 60 O: 0278 (comprehend head :att-of-id)
 p*fixate*interleave-best x7 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*second
 f:recognize-u-model-object
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer a*state*new-important-object*best
 ao*probe*unpack-probe-om-to-superop-om chunk-138 ao*probe*unpack-probe-om-to-superop-om
chunk-139
 a*wm*unpack-applied-om x2
 po*fixate*no-referent x2 po*probe*with-important-object
 a*fixate*newest x2 p*probe*retrieved-by-probe*best p*generic*indifferent x3
 p*fixate*newest*interleave-best
 p*fixate*interleave-best
 61 O: 0262 (fixate zero-head u15 :augmentation)
 ao*fixate chunk-140 ao*fixate chunk-141 ao*fixate chunk-142 ao*fixate chunk-143
 ao*fixate chunk-144 ao*fixate chunk-145 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-146 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-147 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-148 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-149 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-150 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x5
 po*probe*with-attribute

p*probe*new-attribute*best p*generic*indifferent

62 O: 0284 (comprehend zero-head :att-of-id)

p*fixate*interleave-best x7 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
f:recognize-u-model-object

p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer a*state*new-important-object*best
ao*probe*unpack-probe-om-to-superop-om chunk-151

p*probe*retrieved-by-probe*best a*wm*unpack-applied-om

63 O: 0261 (fixate spec u14 :augmentation)

ao*fixate chunk-152 ao*fixate chunk-153 ao*fixate chunk-154 ao*fixate chunk-155
ao*fixate chunk-156 ao*fixate chunk-157 p*generic*terminate-and-reject

a*wm*unpack-applied-om x6 p*generic*reject

ao*fixate*unpack-fixation-object chunk-158 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-159 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-160 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-161 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-162 ao*comprehend*unpack-fixation-object

a*wm*unpack-applied-om x5

po*probe*with-attribute

p*probe*new-attribute*best p*generic*indifferent

64 O: 0286 (comprehend spec :att-of-id)

p*fixate*interleave-best x6 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
f:recognize-u-model-object

p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer a*state*new-important-object*best
ao*probe*unpack-probe-om-to-superop-om chunk-163

p*probe*retrieved-by-probe*best a*wm*unpack-applied-om

65 O: 0260 (fixate empty-node e15 :augmentation)

ao*fixate chunk-164 ao*fixate chunk-165 ao*fixate chunk-166 ao*fixate chunk-167
ao*fixate chunk-168 ao*fixate chunk-169 p*generic*terminate-and-reject

a*wm*unpack-applied-om x6 p*generic*reject

ao*fixate*unpack-fixation-object chunk-170 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-171 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-172 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-173 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-174 ao*comprehend*unpack-fixation-object

a*wm*unpack-applied-om x5

po*probe*with-attribute

p*probe*new-attribute*best p*generic*indifferent

66 O: 0288 (comprehend empty-node :att-of-id)

p*fixate*interleave-best x5 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
f:recognize-u-model-object

p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer a*state*new-important-object*best
ao*probe*unpack-probe-om-to-superop-om chunk-175

p*probe*retrieved-by-probe*best a*wm*unpack-applied-om

67 O: 0259 (fixate right-edge w13 :augmentation)

ao*fixate chunk-176 ao*fixate chunk-177 ao*fixate chunk-178 ao*fixate chunk-179
ao*fixate chunk-180 ao*fixate chunk-181 p*generic*terminate-and-reject

a*wm*unpack-applied-om x6 p*generic*reject

ao*fixate*unpack-fixation-object chunk-182 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-183 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-184 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-185 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-186 ao*comprehend*unpack-fixation-object

a*wm*unpack-applied-om x5

po*probe*with-attribute
 p*probe*new-attribute*best p*generic*indifferent
 68 O: 0290 (comprehend right-edge :att-of-id)
 p*fixate*interleave-best x3 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 f:recognize-u-model-object
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer a*state*new-important-object*best
 ao*probe*unpack-probe-om-to-superop-om chunk-187
 p*probe*retrieved-by-probe*best a*wm*unpack-applied-om
 69 O: 0257 (fixate left-edge w8 :augmentation)
 ao*fixate chunk-188 ao*fixate chunk-189 ao*fixate chunk-190 ao*fixate chunk-191
 ao*fixate chunk-192 ao*fixate chunk-193 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-194 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-195 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-196 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-197 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-198 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x5
 po*probe*with-attribute
 p*probe*new-attribute*best p*generic*indifferent
 70 O: 0292 (comprehend left-edge :att-of-id)
 p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 f:recognize-u-model-object
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer a*state*new-important-object*best
 ao*probe*unpack-probe-om-to-superop-om chunk-199
 p*probe*retrieved-by-probe*best a*wm*unpack-applied-om
 71 O: 0277 (fixate operator* for create-referent :bind-obj)
 ao*fixate chunk-200 ao*fixate chunk-201 ao*fixate chunk-202
 p*generic*terminate-and-reject p*probe*new-attribute*best x3
 a*wm*unpack-applied-om x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-203 **chunk-65** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-204 **chunk-13** **chunk-66**
 ao*comprehend*unpack-fixation-object ao*fixate*unpack-fixation-object chunk-205 **chunk-67**
 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x3
 po*comprehend*objects-attribute po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*generic*indifferent x2
 72 O: 0264 (comprehend operator* |(previous-goal)|)
 p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 73 O: 0281 (fixate :no-referent)
 ao*fixate chunk-206 ao*fixate chunk-207 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x2 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-208 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-209 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x2
 74 O: 0268 (comprehend rhs :part)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:high-level-goal-cues **chunk-46**
chunk-47 **chunk-48** **chunk-51** **chunk-52** **chunk-53** **chunk-57** **chunk-58**
chunk-59 **chunk-69** **chunk-70** **chunk-71** **chunk-77** **chunk-78** **chunk-79**
 po*fixate*action x3 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer

ao*probe*unpack-probe-om-to-superop-om chunk-210 ao*probe*unpack-probe-om-to-superop-om
chunk-211 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-212
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-213
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-215
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-217
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-218
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-219
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-221
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-223
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-224
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-225
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-227
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-229
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-230
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-231
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-233
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-235
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-236
 ao*probe*unpack-probe-om-to-superop-om*fixated x3 chunk-239
 a*fixate*newest x3 a*wm*unpack-applied-om p*generic*indifferent x3
 po*comprehend*high-level-goal po*probe*with-high-level-goal po*imagine*action x3
 po*probe*with-important-object po*imagine*action po*probe*with-important-object x3 po*imagine*action
 po*probe*with-important-object x2 **chunk-76 chunk-75** po*probe*with-important-object
 po*comprehend*operator-condition po*probe*with-important-object x2
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-high-level-goal*best
 p*imagine*refract p*imagine*worst-after-new-output a*fixate*newest p*imagine*refract
 p*imagine*worst-after-new-output a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output
 a*fixate*newest p*imagine*worst-after-new-output a*fixate*newest p*imagine*worst-after-new-output
 a*fixate*newest ao*probe*unpack-probe-om-to-superop-om chunk-240 a*state*new-important-object*best
 ao*probe*unpack-probe-om-to-superop-om chunk-241 ao*goal-select*comprehend*create-token
 a*goal-select*proposed-during p*probe*repeated-goal*worst p*generic*indifferent
 a*wm*unpack-applied-om x2
 po*probe*with-important-object
 p*probe*retrieved-by-probe*best p*generic*indifferent
 75 O: O303 (comprehend return-new-pointer |(high-level-goal)|)
 ao*goal-select*new-goal*probe chunk-242 ao*probe*goal p*generic*terminate-and-reject
 ao*comprehend*applied po*imagine*action x5
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer p*imagine*worst-after-new-output
 a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output a*fixate*newest
 p*imagine*worst-after-new-output a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output
 a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output a*fixate*newest p*generic*indifferent
 x5
 76 O: O301 (comprehend return-new-pointer :high-lev-goal)
 ao*comprehend*applied ao*comprehend*create-dp-on-om
 a*state*applied-newer **chunk-102 chunk-31 chunk-22 chunk-118**
chunk-101 chunk-30 a*dp*unpack-applied-om x4
 a*dp*unpack-applied-om x6
chunk-102 chunk-118 chunk-31
 77 ==>S: S13 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*probe*with-high-level-goal po*probe*with-attribute x6 po*fixate*binding-attribute*target
 po*fixate*selected-operator po*fixate*selected-id po*imagine*nil-object x3 po*fixate*assertion x3
 po*fixate*augmentation po*probe*with-previous-goal po*fixate*binding-attribute*param
 po*probe*with-important-object p*attend*old-regions*reject p*generic*indifferent
 p*probe*repeated-goal*worst a*fixate*newest p*generic*indifferent
 78 O: O357 (imagine nil)
 ao*imagine chunk-243 ao*imagine chunk-244 p*probe*non-important-after-imagine*worst x8
 p*generic*terminate-and-reject ao*substate*count-first

a*wm*unpack-applied-om x4 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-245 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-246 ao*comprehend*unpack-fixation-object
 ao*fixate*mark-imagined-object x2 chunk-247 ao*fixate*mark-imagined-object x2 chunk-248
 a*wm*unpack-applied-om x2
 po*imagine*postpone p*imagine*refract po*probe*with-important-object p*imagine*refract
 a*fixate*newest p*probe*new-important-object*best p*generic*indifferent x2
 79 O: 0356 (imagine nil)
 ao*imagine chunk-249 ao*imagine chunk-250 p*probe*non-important-after-imagine*worst x8
 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om x4 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-251 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-252 ao*comprehend*unpack-fixation-object
 ao*fixate*mark-imagined-object x2 chunk-253 ao*fixate*mark-imagined-object x2 chunk-254
 a*wm*unpack-applied-om x2
 po*imagine*postpone p*imagine*refract po*probe*with-important-object p*imagine*refract
 a*fixate*newest p*probe*new-important-object*best p*generic*indifferent x2
 80 O: 0355 (imagine nil)
 ao*imagine chunk-255 ao*imagine chunk-256 p*probe*non-important-after-imagine*worst x8
 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x4 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-257 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-258 ao*comprehend*unpack-fixation-object
 ao*fixate*mark-imagined-object x2 chunk-259 ao*fixate*mark-imagined-object x2 chunk-260
 a*wm*unpack-applied-om x2
 po*imagine*postpone p*imagine*refract po*probe*with-important-object p*imagine*refract
 a*fixate*newest p*probe*new-important-object*best p*probe*new-attribute*best p*generic*indifferent x2
 81 O: 0389 (imagine high-level-goal postpone)
 ao*imagine chunk-261 p*probe*non-important-after-imagine*worst x8 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x2 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-262 ao*comprehend*unpack-fixation-object
 ao*fixate*mark-imagined-object x2 chunk-263
 a*wm*unpack-applied-om
 po*comprehend*high-level-goal p*imagine*refract x3
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent
 ao*goal-select*new-goal*fixate/imagine chunk-264
 82 O: 0398 (comprehend postpone :high-lev-goal)
 ao*comprehend*applied ao*comprehend*create-dp-on-om
 a*state*applied-newer **chunk-22 chunk-31 chunk-118 chunk-102**
chunk-117 chunk-21 chunk-1 x2 a*dp*unpack-applied-om x4
 a*dp*unpack-applied-om
chunk-118 chunk-22 chunk-31 a*wm*newest-from-not-newest
 83 ==>S: S14 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects po*imagine*operators a*substate*create-time
 a*fixate*newest x4 po*fixate*binding-attribute*target po*fixate*selected-operator
 po*fixate*selected-id po*fixate*augmentation po*fixate*assertion x3 po*fixate*augmentation x7
 po*probe*with-previous-goal po*fixate*binding-attribute*param po*probe*with-important-object x3
 p*attend*old-regions*reject p*generic*indifferent x5
 a*fixate*newest p*generic*indifferent

p*fixate*newest*interleave-best x7

84 O: O404 (imagine operator* exhausted)

p*fixate*interleave-best x7 ao*imagine chunk-265 p*probe*non-important-after-imagine*worst
p*generic*terminate-and-reject ao*substate*count-first

a*wm*unpack-applied-om x2 p*generic*reject

ao*fixate*unpack-fixation-object chunk-266 ao*comprehend*unpack-fixation-object
ao*fixate*mark-imagined-object x2 chunk-267

a*wm*unpack-applied-om

f:select-exhausted po*probe*with-important-object p*imagine*refract

a*wm*unpack-applied-om p*probe*new-important-object*best p*generic*indifferent

po*comprehend*current-context-when-exhausted

ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent

ao*goal-select*select-now

85 O: O430 (comprehend current-context :sno)

ao*comprehend*applied ao*comprehend*create-dp-on-om po*display*print-stack

a*state*applied-newer **chunk-102 chunk-31 chunk-22 chunk-118** p*display*dunk-comprehend
a*display*t323 p*generic*indifferent

chunk-101 chunk-30 a*dp*unpack-applied-om x4

a*dp*unpack-applied-om x6

chunk-102 chunk-118 chunk-31

86 O: O432 (display print-stack (t323))

ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator

a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
a*wm*unpack-applied-om p*generic*reject ao*display*print*t323

d*print*t323

87 O: O430 (comprehend current-context :sno)

po*display*print-stack **chunk-22 chunk-31 chunk-102 chunk-118**

p*display*reject-duplicates p*generic*indifferent

88 ==>S: S16 (operator no-change)

a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
a*substate*initialize-goal-set a*state*important-objects a*substate*create-time

po*attend

p*generic*indifferent

89 O: O434 (attend t323 constant29)

ao*attend chunk-268 ao*attend*mark-locally p*generic*terminate-and-reject

a*dp*unpack-applied-om x3 p*generic*reject

a*wm*newest-from-not-newest

ao*attend*previous-not-newest chunk-269 po*fixate*binding-attribute*target
po*fixate*selected-operator po*fixate*selected-id po*fixate*assertion x3 po*fixate*augmentation
x8 po*probe*with-previous-goal po*fixate*current-context po*fixate*binding-attribute*param
po*probe*with-important-object

a*dp*unpack-applied-om p*generic*indifferent

a*fixate*newest

p*fixate*newest*interleave-best

90 O: O453 (comprehend exhausted :imp-obj operator*)

p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
f:high-level-goal-cues f:operator f:exhausted-builds-proposal ao*substate*count-first

p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om
chunk-270 ao*probe*unpack-probe-om-to-superop-om chunk-271
ao*probe*unpack-probe-om-to-superop-om chunk-272

a*wm*unpack-applied-om x3
 po*comprehend*high-level-goal po*probe*with-high-level-goal
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-high-level-goal*best
 p*generic*indifferent x2

91 O: 0451 (fixate :current-context s-construct)
 ao*fixate chunk-273 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-274 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*comprehend*actual-context po*fixate*state-id po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during a*fixate*newest
 p*probe*new-important-object*best p*generic*indifferent x3
 ao*goal-select*new-goal*fixate/imagine chunk-275 p*fixate*newest*interleave-best

92 O: 0459 (comprehend s-construct :actual-context)
 ao*comprehend*applied f:s-construct-builds-chunk f:recall-u-model ao*comprehend*create-dp-on-om
 a*state*applied-newer a*wm*unpack-applied-om x3 **chunk-118 chunk-102 chunk-31**
chunk-22 chunk-269
chunk-117 chunk-21 chunk-1 x2 a*dp*unpack-applied-om x5
 a*dp*unpack-applied-om
chunk-118 chunk-269 chunk-31 chunk-22

93 ==>S: S17 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*probe*with-high-level-goal po*fixate*binding-attribute*target po*fixate*selected-operator
 po*fixate*selected-id po*fixate*augmentation po*fixate*assertion x3 po*fixate*augmentation x7
 po*probe*with-previous-goal po*fixate*state-id po*fixate*binding-attribute*param
 po*probe*with-important-object x3 p*attend*old-regions*reject p*generic*indifferent
 a*fixate*newest p*probe*repeated-goal*worst p*generic*indifferent
 p*fixate*newest*interleave-best

94 O: 0489 (comprehend apply-chunks :imp-obj chunk)
 p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:abstract-sp ao*substate*count-first
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om
chunk-276
 a*wm*unpack-applied-om
 f:sp-parts po*probe*with-important-object
 ao*probe*unpack-probe-om-to-superop-om chunk-277 ao*probe*unpack-probe-om-to-superop-om
chunk-278 p*probe*repeated-goal*worst p*generic*indifferent
 a*wm*unpack-applied-om x2
 po*comprehend*sp-parts po*probe*with-part po*comprehend*sp-parts po*probe*with-part
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*lhs-best
 p*generic*indifferent x4

95 O: 0483 (fixate state s15 :state-id)
 ao*fixate chunk-279 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-280 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*comprehend*u-model po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*generic*indifferent x2

96 O: 0499 (comprehend lhs :part)
 ao*goal-select*new-goal*probe chunk-281 ao*probe*goal p*generic*terminate-and-reject
 ao*comprehend*applied*second f:operator-condition*lhs-means-rhs
 po*fixate*condition x5 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x4 p*generic*indifferent x5
 a*fixate*dont-interleave-best x2

97 O: 0497 (comprehend lhs :sp-parts)
 ao*comprehend*applied f:operator-condition*lhs-means-rhs ao*comprehend*create-dp-on-om
 a*state*applied-newer a*wm*unpack-applied-om **chunk-102 chunk-22 chunk-31 chunk-269**
chunk-118
chunk-268 chunk-101 chunk-30 a*dp*unpack-applied-om x5
 a*dp*unpack-applied-om x9
chunk-269 chunk-102 chunk-118 chunk-31 a*wm*newest-from-not-newest

98 ==>S: S18 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*fixate*binding-attribute*target po*fixate*selected-operator po*fixate*selected-id po*fixate*assertion
 x3 po*fixate*augmentation x8 po*probe*with-previous-goal po*probe*with-part x2
 po*fixate*current-context po*fixate*binding-attribute*param po*probe*with-important-object x3
 po*fixate*condition x5 p*attend*old-regions*reject p*generic*indifferent
 a*fixate*newest p*probe*lhs-best p*probe*repeated-goal*worst a*fixate*newest x2
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x4 p*generic*indifferent
 p*fixate*newest*interleave-best a*fixate*dont-interleave-best x2

99 O: 0527 (comprehend lhs :part)
 p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 p*generic*reject a*state*applied-newer

100 O: 0531 (fixate :current-context s-construct)
 ao*fixate chunk-282 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-283 ao*comprehend*unpack-fixation-object **chunk-274**
 a*wm*unpack-applied-om
 po*comprehend*actual-context po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*generic*indifferent x2

101 O: 0546 (comprehend s-construct :imp-obj current-context)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*second f:s-construct-builds-chunk
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
 a*state*new-important-object*best ao*probe*unpack-probe-om-to-superop-om chunk-284
 ao*probe*unpack-probe-om-to-superop-om chunk-285
 p*probe*retrieved-by-probe*best a*wm*unpack-applied-om x2
 po*probe*with-important-object
 p*probe*retrieved-by-probe*best p*generic*indifferent

102 O: 0537 (comprehend apply-chunks :imp-obj sp)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:abstract-sp f:sp-parts
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer

103 O: 0543 (fixate :condition problem-space s-construct)
 ao*fixate chunk-286 ao*fixate chunk-287 ao*fixate chunk-288
 p*generic*terminate-and-reject

a*wm*unpack-applied-om x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-289 **chunk-72** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-290 **chunk-73** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-291 **chunk-74** ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x3
 po*probe*with-important-object x2
 p*probe*new-important-object*best p*probe*new-attribute*best p*probe*new-important-object*best
 p*probe*repeated-goal*worst p*generic*indifferent x2
 104 O: 0550 (comprehend problem-space :imp-obj condition)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 105 O: 0542 (fixate :condition operator* create-referent)
 ao*fixate chunk-292 ao*fixate chunk-293 ao*fixate chunk-294
 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-295 **chunk-13** **chunk-66** **chunk-204**
 ao*comprehend*unpack-fixation-object ao*fixate*unpack-fixation-object chunk-296 **chunk-80**
 ao*comprehend*unpack-fixation-object ao*fixate*unpack-fixation-object chunk-297 **chunk-81**
 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x3
 po*probe*with-important-object po*comprehend*operator-condition po*probe*with-important-object x2
 p*probe*new-important-object*best ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 p*probe*new-important-object*best p*probe*new-attribute*best p*probe*new-important-object*best
 p*generic*indifferent x4
 ao*goal-select*new-goal*fixate/imagine chunk-298
 106 O: 0556 (comprehend operator* :op-cond)
 ao*comprehend*applied f:operator-condition*part-of-lhs ao*comprehend*create-dp-on-om **chunk-84**
chunk-85 **chunk-86** **chunk-87** **chunk-88** **chunk-89** **chunk-95** **chunk-96**
chunk-97 **chunk-103** **chunk-104** **chunk-105** **chunk-106** **chunk-107** **chunk-108**
 a*state*applied-newer a*wm*unpack-applied-om **chunk-22** **chunk-31** **chunk-118** **chunk-102**
chunk-269
chunk-117 **chunk-21** **chunk-1** x2 **chunk-114** ao*comprehend*unpack-fixation-object
chunk-113 ao*comprehend*unpack-fixation-object **chunk-112** ao*comprehend*unpack-fixation-object
chunk-111 ao*comprehend*unpack-fixation-object **chunk-110** ao*comprehend*unpack-fixation-object
chunk-109 ao*comprehend*unpack-fixation-object **chunk-10** **chunk-100**
 ao*comprehend*unpack-fixation-object **chunk-9** **chunk-99** ao*comprehend*unpack-fixation-object
chunk-8 **chunk-98** ao*comprehend*unpack-fixation-object **chunk-43** **chunk-94**
 ao*comprehend*unpack-fixation-object **chunk-42** **chunk-93** ao*comprehend*unpack-fixation-object
chunk-41 **chunk-92** ao*comprehend*unpack-fixation-object **chunk-40**
 ao*comprehend*unpack-fixation-object **chunk-39** **chunk-91** ao*comprehend*unpack-fixation-object
chunk-38 **chunk-90** ao*comprehend*unpack-fixation-object a*dp*unpack-applied-om x5
 a*dp*unpack-applied-om a*wm*unpack-applied-om
chunk-118 **chunk-269** **chunk-22** **chunk-31**
 107 ==>S: S19 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*probe*with-attribute po*fixate*selected-operator po*fixate*assertion x3 po*fixate*augmentation
 x7 po*probe*with-previous-goal po*probe*with-part x2 po*fixate*state-id
 po*probe*with-important-object x8 p*attend*old-regions*reject p*generic*indifferent
 a*fixate*newest p*probe*lhs-best x2 a*fixate*newest p*probe*repeated-goal*worst
 p*generic*indifferent
 p*fixate*newest*interleave-best
 108 O: 0578 (comprehend lhs :part)
 p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs **chunk-286** **chunk-287** **chunk-288** **chunk-292**

chunk-293 chunk-294 ao*substate*count-first

po*fixate*condition x3 po*fixate*binding-context p*probe*repeated-goal*worst x2 p*generic*reject
a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om*fixated chunk-299
ao*probe*unpack-probe-om-to-superop-om*fixated x3 chunk-302
ao*probe*unpack-probe-om-to-superop-om*fixated x2

p*fixate*invariant-feature*dont-interleave-best a*fixate*newest
p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x3 p*fixate*interleave-best
a*wm*unpack-applied-om p*generic*indifferent x4

a*fixate*dont-interleave-best x2

109 O: 0582 (fixate state s15 :state-id)

ao*fixate chunk-305 p*generic*terminate-and-reject ao*substate*count-second

a*wm*unpack-applied-om p*generic*reject

ao*fixate*unpack-fixation-object chunk-306 ao*comprehend*unpack-fixation-object **chunk-280**

a*wm*unpack-applied-om

po*probe*with-important-object

p*probe*new-important-object*best p*generic*indifferent

110 O: 0603 (comprehend s15 :imp-obj state)

p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*second

p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer

111 O: 0602 (fixate operator* for create-referent :bind-obj)

ao*fixate chunk-307 ao*fixate chunk-308 ao*fixate chunk-309
p*generic*terminate-and-reject p*probe*new-attribute*best x3 p*probe*new-important-object*best
a*wm*unpack-applied-om x3 p*generic*reject

ao*fixate*unpack-fixation-object chunk-310 **chunk-65 chunk-203**
ao*comprehend*unpack-fixation-object ao*fixate*unpack-fixation-object chunk-311 **chunk-67**
chunk-205 ao*comprehend*unpack-fixation-object

a*wm*unpack-applied-om x2

po*comprehend*objects-attribute

ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent

112 O: 0595 (comprehend for :imp-obj condition)

ao*goal-select*new-goal*probe chunk-312 ao*probe*goal p*generic*terminate-and-reject
ao*comprehend*applied frecall-for-part-of-u-model **chunk-119 chunk-120 chunk-121**
chunk-122 chunk-123 chunk-124 chunk-140 chunk-141 chunk-142 chunk-143
chunk-144 chunk-145 chunk-152 chunk-153 chunk-154 chunk-155 chunk-156
chunk-157 chunk-164 chunk-165 chunk-166 chunk-167 chunk-168 chunk-169
chunk-176 chunk-177 chunk-178 chunk-179 chunk-180 chunk-181 chunk-188
chunk-189 chunk-190 chunk-191 chunk-192 chunk-193 chunk-211 chunk-217
chunk-223 chunk-227 chunk-221 chunk-219 chunk-215 chunk-225 chunk-213
chunk-210

p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
ao*probe*unpack-probe-om-to-superop-om chunk-313 ao*probe*unpack-probe-om-to-superop-om*fixated
chunk-314 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-315
ao*probe*unpack-probe-om-to-superop-om*fixated chunk-316
ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-318
ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-320
ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-322
ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-324
ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-326
ao*probe*unpack-probe-om-to-superop-om*fixated chunk-327
ao*probe*unpack-probe-om-to-superop-om*fixated chunk-328
ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-330
ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-332
ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-334
ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-336
ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-338
ao*probe*unpack-probe-om-to-superop-om*fixated chunk-339
ao*probe*unpack-probe-om-to-superop-om*fixated chunk-340
ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-342

ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-344
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-346
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-348
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-350
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-351
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-352
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-354
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-356
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-358
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-360
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-362
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-363
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-364
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-366
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-368
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-370
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-372
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-374
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-375
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-376
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-378
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-380
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-382
 ao*probe*unpack-probe-om-to-superop-om*fixated x3 chunk-385
 ao*probe*unpack-probe-om-to-superop-om chunk-386 ao*probe*unpack-probe-om-to-superop-om
chunk-387 ao*probe*unpack-probe-om-to-superop-om chunk-388 a*state*new-important-object*best
 ao*probe*unpack-probe-om-to-superop-om chunk-389 a*state*new-important-object*best
 ao*probe*unpack-probe-om-to-superop-om chunk-390 a*state*new-important-object*best
 ao*probe*unpack-probe-om-to-superop-om chunk-391 a*state*new-important-object*best
 ao*probe*unpack-probe-om-to-superop-om chunk-392 a*state*new-important-object*best
 ao*probe*unpack-probe-om-to-superop-om chunk-393 ao*probe*unpack-probe-om-to-superop-om
chunk-394
 a*wm*unpack-applied-om
 po*comprehend*objects-attribute **chunk-138 chunk-139** po*probe*with-attribute **chunk-151**
 po*probe*with-attribute **chunk-163** po*probe*with-attribute **chunk-175** po*probe*with-attribute
chunk-187 po*probe*with-attribute **chunk-199** po*probe*with-attribute
 po*probe*with-high-level-goal po*probe*with-important-object x5 po*comprehend*high-level-goal
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during a*state*new-important-object*best
 ao*probe*unpack-probe-om-to-superop-om chunk-395 ao*probe*unpack-probe-om-to-superop-om
chunk-396 ao*probe*unpack-probe-om-to-superop-om chunk-397
 ao*probe*unpack-probe-om-to-superop-om chunk-398 ao*probe*unpack-probe-om-to-superop-om
chunk-399 ao*probe*unpack-probe-om-to-superop-om chunk-400
 ao*probe*unpack-probe-om-to-superop-om chunk-401 p*probe*new-high-level-goal*best
 p*probe*retrieved-by-probe*best x5 ao*goal-select*comprehend*create-token
 a*goal-select*proposed-during p*generic*indifferent
 ao*goal-select*new-goal*probe chunk-402 a*wm*unpack-applied-om x7
chunk-206 chunk-207 po*fixate*no-referent x2 po*probe*with-important-object
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-403
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-404 a*fixate*newest x2
 p*probe*retrieved-by-probe*best p*generic*indifferent x3
 a*wm*unpack-applied-om x2
 113 O: O605 (comprehend for :objects-att)
 ao*comprehend*applied f:recall-for-part-of-u-model ao*comprehend*create-dp-on-om
 a*state*applied-newer a*wm*unpack-applied-om **chunk-102 chunk-31 chunk-22 chunk-269**
chunk-118
chunk-13 chunk-66 chunk-204 chunk-295 ao*comprehend*unpack-fixation-object
chunk-268 chunk-101 chunk-30 a*dp*unpack-applied-om x5
 po*display*print-object*fresh x2 a*wm*unpack-applied-om a*dp*unpack-applied-om x9
 p*display*dunk-comprehend x2 **chunk-269 chunk-102 chunk-118 chunk-31**
 a*wm*newest-from-not-newest p*generic*indifferent x2
 114 ==>S: S20 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set

a*state*important-objects a*substate*create-time

po*probe*with-high-level-goal po*probe*with-attribute x7 po*fixate*selected-operator
 po*fixate*assertion x3 po*probe*with-previous-goal po*probe*with-part po*fixate*current-context
 po*probe*with-important-object x9 p*attend*old-regions*reject p*generic*indifferent

p*probe*repeated-goal*worst a*fixate*newest x4 p*probe*lhs-best a*fixate*newest
 p*probe*repeated-goal*worst p*generic*indifferent

p*fixate*newest*interleave-best

115 O: 0664 (comprehend lhs :part)

p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs **chunk-286 chunk-287 chunk-288 chunk-292**
chunk-293 chunk-294 chunk-282 ao*substate*count-first

po*fixate*condition x5 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 ao*probe*unpack-probe-om-to-superop-om chunk-405 ao*probe*unpack-probe-om-to-superop-om*fixated
chunk-406 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-407
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-408
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-409
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-410
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-411
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-412

p*fixate*invariant-feature*dont-interleave-best a*fixate*newest
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x4 a*wm*unpack-applied-om x8
 p*generic*indifferent x5

a*fixate*dont-interleave-best x2 **chunk-241 chunk-240** po*comprehend*sp-parts
 po*probe*with-part po*probe*with-important-object x2 po*comprehend*operator-condition
 po*probe*with-important-object x2 **chunk-285 chunk-284** po*comprehend*actual-context
 po*probe*with-important-object

a*wm*unpack-applied-om x2 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 ao*probe*unpack-probe-om-to-superop-om chunk-413 a*state*new-important-object*best
 ao*probe*unpack-probe-om-to-superop-om chunk-414 ao*goal-select*comprehend*create-token
 a*goal-select*proposed-during p*generic*indifferent x9

po*probe*with-important-object

p*probe*retrieved-by-probe*best p*generic*indifferent

116 O: 0708 (comprehend apply-chunks :imp-obj chunk)

ao*probe*goal p*generic*terminate-and-reject ao*substate*count-second ao*comprehend*applied*second
 f:abstract-sp

p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om
chunk-415

a*wm*unpack-applied-om

f:sp-parts po*probe*with-important-object

p*probe*repeated-goal*worst p*generic*indifferent

117 O: 0683 (comprehend referent-of :imp-obj action)

ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied

p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer

118 O: 0681 (comprehend obj :imp-obj action)

ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied

p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer

119 O: 0679 (comprehend type :imp-obj action)

ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied

p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer

120 O: 0677 (comprehend s-model :imp-obj action)

ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied

p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer

121 O: 0675 (comprehend referent :imp-obj action)

ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 122 O: 0671 (comprehend u-model :imp-obj object)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:high-level-goal-cues f:u-model
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer p*probe*new-high-level-goal*best
 ao*probe*unpack-probe-om-to-superop-om chunk-416 ao*probe*unpack-probe-om-to-superop-om
chunk-417
 a*wm*unpack-applied-om x2
 po*probe*with-high-level-goal
 p*probe*new-high-level-goal*best p*generic*indifferent
 123 O: 0642 (comprehend return-new-pointer |(high-level-goal)|)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:postpone-return-new-pointer
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
 ao*probe*unpack-probe-om-to-superop-om chunk-418
 a*wm*unpack-applied-om
 po*comprehend*where-was-i x2
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent x2
 ao*goal-select*select-now x2
 124 O: 0716 (comprehend where-was-i :sno)
 ao*comprehend*applied ao*comprehend*create-dp-on-om
 a*state*applied-newer **chunk-22 chunk-31 chunk-118 chunk-102 chunk-269**
chunk-117 chunk-21 chunk-1 x2 a*dp*unpack-applied-om x5
 a*dp*unpack-applied-om
chunk-118 chunk-269 chunk-22 chunk-31
 125 ==>S: S21 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*fixate*binding-attribute*target po*fixate*selected-operator po*fixate*selected-id
 po*fixate*augmentation po*fixate*assertion x3 po*fixate*augmentation x7
 po*probe*with-previous-goal po*probe*with-part po*fixate*state-id po*fixate*binding-attribute*param
 po*probe*with-important-object x8 po*fixate*where-was-i*assertions x2 p*attend*old-regions*reject
 p*generic*indifferent
 a*fixate*newest p*generic*indifferent
 p*fixate*newest*interleave-best
 126 O: 0756 (fixate ms-t225 where-was-i:assertions)
 ao*fixate chunk-419 p*fixate*interleave-best p*generic*terminate-and-reject ao*substate*count-first
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-420 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*display*run*where-was-i
 p*display*dunk-comprehend a*display*t364 p*generic*indifferent
 127 O: 0757 (display run:where-was-i (t364))
 ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
 a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
 a*wm*unpack-applied-om p*generic*reject ao*display*print*t364
 d*print*t364
 128 O: 0716 (comprehend where-was-i :sno)
 po*display*run*where-was-i **chunk-22 chunk-31 chunk-102 chunk-118 chunk-269**
 p*display*reject-duplicates p*generic*indifferent

129 ==>S: S23 (operator no-change)
a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
po*attend
p*generic*indifferent

130 O: 0759 (attend t364 constant50)
ao*attend chunk-421 ao*attend*mark-locally p*generic*terminate-and-reject
a*dp*unpack-applied-om x3 p*generic*reject
a*wm*newest-from-not-newest
ao*attend*previous-not-newest chunk-422 po*fixate*builds po*fixate*binding-attribute*target
po*fixate*selected-operator po*fixate*selected-id po*fixate*augmentation po*fixate*assertion x3
po*fixate*augmentation x7 po*probe*with-previous-goal po*probe*with-part po*fixate*state-id
po*fixate*binding-attribute*param po*probe*with-important-object x8 po*fixate*where-was-i*chunks-built
a*dp*unpack-applied-om p*generic*indifferent
a*fixate*newest
p*fixate*newest*interleave-best x2

131 O: 0797 (fixate builds-364 where-was-i:chunks)
ao*fixate chunk-423 p*generic*terminate-and-reject ao*substate*count-first
a*wm*unpack-applied-om p*generic*reject
ao*fixate*unpack-fixation-object chunk-424 ao*comprehend*unpack-fixation-object
a*wm*unpack-applied-om
po*fixate*chunk x2 po*comprehend*builds
a*fixate*newest x2 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
p*generic*indifferent x3
p*fixate*newest*interleave-best x2
p*fixate*top-down

132 O: 0795 (comprehend apply-chunks :imp-obj sp)
p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*substate*count-second
ao*comprehend*applied*first f:abstract-sp f:sp-parts
p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
ao*probe*unpack-probe-om-to-superop-om chunk-425
a*wm*unpack-applied-om
po*comprehend*sp-parts po*probe*with-part
ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*lhs-best
p*generic*indifferent x2

133 O: 0798 (fixate :chunk chunk-128)
ao*fixate chunk-426 p*generic*terminate-and-reject
a*wm*unpack-applied-om p*generic*reject
ao*fixate*unpack-fixation-object chunk-427 ao*comprehend*unpack-fixation-object
a*wm*unpack-applied-om
po*comprehend*chunk po*probe*with-important-object
ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
p*generic*indifferent x2
ao*goal-select*new-goal*fixate/imagine chunk-428

134 O: 0806 (comprehend chunk-128 :chunk)
ao*comprehend*applied f:sp ao*comprehend*create-dp-on-om po*display*print-chunk
a*state*applied-newer a*wm*unpack-applied-om **chunk-269 chunk-118 chunk-102 chunk-31**
chunk-22 chunk-422 p*display*dunk-comprehend a*display*t373 p*generic*indifferent
chunk-268 chunk-101 chunk-30 f:sp-parts a*dp*unpack-applied-om x6
a*dp*unpack-applied-om x9 a*wm*unpack-applied-om x2

chunk-269 chunk-422 chunk-118 chunk-102 chunk-31 po*comprehend*sp-parts
ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent

135 O: 0810 (display print-chunk chunk-128 (t373))
ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
a*wm*unpack-applied-om p*generic*reject ao*display*print*t373
d*print*t373

136 O: 0806 (comprehend chunk-128 :chunk)
f:sp f:sp-parts ao*comprehend*cleanup-naked-region-pointers po*display*print-chunk **chunk-269
chunk-422**
p*display*reject-duplicates p*generic*indifferent

137 ==>S: S25 (operator no-change)
a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
po*attend
p*generic*indifferent

138 O: 0814 (attend t373 constant55)
ao*attend chunk-429 ao*attend*mark-locally p*generic*terminate-and-reject
a*dp*unpack-applied-om x3 p*generic*reject
a*wm*newest-from-not-newest
ao*attend*previous-not-newest chunk-430 ao*fixated-recently chunk-431 po*fixate*chunk
po*probe*with-previous-goal po*probe*with-part x2 po*fixate*current-context
po*probe*with-important-object x2
a*dp*unpack-applied-om a*wm*unpack-applied-om p*probe*lhs-best p*probe*repeated-goal*worst x2
p*generic*indifferent x7
po*probe*with-important-object
a*fixate*newest x2 p*probe*fixated-recently*worst p*generic*indifferent

139 O: 0818 (comprehend lhs :part)
ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
f:operator-condition*lhs-means-rhs **chunk-282** ao*substate*count-first **chunk-431**
po*fixate*condition x5 po*fixate*no-problem-space p*probe*repeated-goal*worst p*generic*reject
a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om*fixated chunk-432
ao*probe*unpack-probe-om-to-superop-om*fixated chunk-433 a*state*new-important-object*best
p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x6 a*wm*unpack-applied-om x2
p*probe*retrieved-by-probe*best p*generic*indifferent x6
a*fixate*dont-interleave-best p*fixate*newest*interleave-best x6 **chunk-285 chunk-284**
po*comprehend*actual-context po*fixate*state-id po*probe*with-important-object
p*fixate*interleave-best x3 p*fixate*top-down p*fixate*interleave-best x2
ao*probe*unpack-probe-om-to-superop-om chunk-434 a*state*new-important-object*best
ao*probe*unpack-probe-om-to-superop-om chunk-435 ao*goal-select*comprehend*create-token
a*goal-select*proposed-during a*fixate*newest p*generic*indifferent x3
a*wm*unpack-applied-om x2
po*probe*with-important-object
p*probe*retrieved-by-probe*best p*generic*indifferent

140 O: 0834 (fixate :no-space)
ao*fixate chunk-436 ao*fixate chunk-437 p*generic*terminate-and-reject
ao*substate*count-second
a*wm*unpack-applied-om x2 p*generic*reject p*probe*rhs-when-sp
ao*fixate*unpack-fixation-object chunk-438 **chunk-73 chunk-290**
ao*comprehend*unpack-fixation-object ao*fixate*unpack-fixation-object chunk-439
ao*comprehend*unpack-fixation-object
a*wm*unpack-applied-om x2

po*probe*with-important-object
 p*probe*new-important-object*best p*probe*new-attribute*best p*generic*indifferent

141 O: 0820 (comprehend rhs :part)
 p*fixate*interleave-best x4 ao*goal-select*new-goal*probe chunk-440 ao*probe*goal
 p*generic*terminate-and-reject ao*comprehend*applied*second f:high-level-goal-cues **chunk-431**
 p*fixate*bottom-up po*fixate*action x3 p*probe*repeated-goal*worst p*generic*reject
 a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om chunk-441
 ao*probe*unpack-probe-om-to-superop-om chunk-442 a*state*new-important-object*best
 a*fixate*newest x3 a*wm*unpack-applied-om x2 p*probe*retrieved-by-probe*best
 p*generic*indifferent x3
 p*fixate*newest*interleave-best x3 po*comprehend*high-level-goal po*probe*with-high-level-goal
 p*fixate*interleave-best p*fixate*bottom-up x3 p*fixate*interleave-best x2
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-high-level-goal*best
 p*generic*indifferent x2

142 O: 0811 (comprehend rhs :sp-parts)
 ao*comprehend*applied f:high-level-goal-cues ao*comprehend*create-dp-on-om **chunk-431**
 a*state*applied-newer a*wm*unpack-applied-om x3 **chunk-422 chunk-430**
chunk-421 a*dp*unpack-applied-om x2
 a*dp*unpack-applied-om x3
chunk-422 chunk-430

143 ==>S: S26 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*fixate*builds po*probe*with-high-level-goal po*probe*with-previous-goal po*probe*with-part x2
 po*fixate*state-id po*probe*with-important-object x5 po*fixate*action x3
 p*attend*old-regions*reject p*generic*indifferent
 a*fixate*newest p*probe*lhs-best p*probe*repeated-goal*worst a*fixate*newest
 p*probe*fixated-recently*worst a*fixate*newest x3 p*generic*indifferent
 p*fixate*newest*interleave-best x3
 p*fixate*bottom-up x2

144 O: 0857 (comprehend lhs :part)
 p*fixate*interleave-best x3 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 po*fixate*condition x5 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x5 p*generic*indifferent x5
 a*fixate*dont-interleave-best p*fixate*newest*interleave-best x5
 p*fixate*interleave-best x3 p*fixate*bottom-up p*fixate*interleave-best p*fixate*bottom-up

145 O: 0874 (fixate :action referent-of u1)
 ao*fixate chunk-443 ao*fixate chunk-444 ao*fixate chunk-445
 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-446 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-447 **chunk-49** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-448 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x3
 po*probe*with-important-object x2
 p*probe*new-important-object*best p*probe*new-attribute*best p*probe*new-important-object*best
 p*generic*indifferent x2

146 O: 0882 (comprehend u1 :imp-obj action)
 p*fixate*interleave-best x6 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*second
 po*imagine*action
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer p*imagine*refract a*fixate*newest

- p*generic*indifferent
- 147 O: 0873 (fixate :action type s-model)
- ao*fixate chunk-449 ao*fixate chunk-450 ao*fixate chunk-451
p*generic*terminate-and-reject
a*wm*unpack-applied-om x3 p*generic*reject
ao*fixate*unpack-fixation-object chunk-452 **chunk-54** ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-453 **chunk-55** ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-454 **chunk-56** ao*comprehend*unpack-fixation-object
a*wm*unpack-applied-om x3
po*probe*with-important-object x2
p*probe*new-important-object*best p*probe*new-attribute*best p*probe*new-important-object*best
p*generic*indifferent x2
- 148 O: 0887 (comprehend s-model :imp-obj action)
- p*fixate*interleave-best x5 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
po*imagine*action x2
p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer p*imagine*refract a*fixate*newest
p*imagine*refract a*fixate*newest p*generic*indifferent x2
- 149 O: 0872 (fixate :action ul referent)
- ao*fixate chunk-455 ao*fixate chunk-456 ao*fixate chunk-457
p*generic*terminate-and-reject p*probe*new-attribute*best p*probe*new-important-object*best
a*wm*unpack-applied-om x3 p*generic*reject
ao*fixate*unpack-fixation-object chunk-458 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-459 **chunk-61** ao*comprehend*unpack-fixation-object
a*wm*unpack-applied-om x2
po*probe*with-important-object
p*probe*new-important-object*best p*generic*indifferent
- 150 O: 0891 (comprehend referent :imp-obj action)
- p*fixate*interleave-best x4 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
po*imagine*action x3
p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer p*imagine*refract a*fixate*newest
p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest p*generic*indifferent x3
- 151 O: 0879 (fixate :condition referent nil)
- ao*fixate chunk-460 ao*fixate chunk-461 ao*fixate chunk-462
p*generic*terminate-and-reject p*probe*new-attribute*best
a*wm*unpack-applied-om x3 p*generic*reject
ao*fixate*unpack-fixation-object chunk-463 **chunk-208** ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-464 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-465 ao*comprehend*unpack-fixation-object
a*wm*unpack-applied-om x3
po*probe*with-important-object x2
p*probe*new-important-object*best p*probe*repeated-goal*worst p*probe*new-attribute*best
p*probe*new-important-object*best p*generic*indifferent x2
- 152 O: 0898 (comprehend nil :imp-obj condition)
- p*fixate*interleave-best x3 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
po*imagine*action x4
p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer p*imagine*refract a*fixate*newest
p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest x2 p*generic*indifferent x4
- 153 O: 0878 (fixate :condition path-to-referent)
- ao*fixate chunk-466 ao*fixate chunk-467 p*generic*terminate-and-reject
a*wm*unpack-applied-om x2 p*generic*reject
ao*fixate*unpack-fixation-object chunk-468 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-469 ao*comprehend*unpack-fixation-object

a*wm*unpack-applied-om x2
 po*probe*with-important-object
 p*probe*new-important-object*best p*probe*new-attribute*best p*generic*indifferent
 154 O: 0904 (comprehend path-to-referent :imp-obj condition)
 p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 po*imagine*action x4
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer p*imagine*refract a*fixate*newest
 p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest x2 p*generic*indifferent x4
 155 O: 0877 (fixate :condition operator* s-structor16)
 ao*fixate chunk-470 ao*fixate chunk-471 ao*fixate chunk-472
 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-473 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-474 **chunk-80 chunk-296**
 ao*comprehend*unpack-fixation-object ao*fixate*unpack-fixation-object chunk-475
 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x3
 po*probe*with-important-object po*comprehend*operator-condition po*probe*with-important-object x2
 p*probe*new-important-object*best ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 p*probe*new-important-object*best p*probe*new-attribute*best p*probe*new-important-object*best
 p*generic*indifferent x4
 ao*goal-select*new-goal*fixate/imagine chunk-476
 156 O: 0912 (comprehend operator* :op-cond)
 ao*comprehend*applied f:operator-condition*part-of-lhs ao*comprehend*create-dp-on-om **chunk-305**
 a*state*applied-newer a*wm*unpack-applied-om x2 **chunk-430 chunk-422**
chunk-429 chunk-268 chunk-83 chunk-280 chunk-306
 ao*comprehend*unpack-fixation-object a*dp*unpack-applied-om x2
 a*dp*unpack-applied-om x6 a*wm*unpack-applied-om x2
chunk-430 chunk-422 a*wm*newest-from-not-newest po*comprehend*sp-parts
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent
 157 ==>S: S27 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*fixate*builds po*probe*with-high-level-goal po*probe*with-previous-goal po*probe*with-part x2
 po*fixate*current-context po*probe*with-important-object p*attend*old-regions*reject
 p*generic*indifferent
 a*fixate*newest p*probe*lhs-best a*fixate*newest p*probe*fixated-recently*worst
 p*probe*repeated-goal*worst p*generic*indifferent
 158 O: 0926 (comprehend lhs :part)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs **chunk-282** ao*substate*count-first
 po*fixate*condition x2 po*fixate*no-problem-space p*probe*repeated-goal*worst p*generic*reject
 a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om*fixated chunk-477
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-478
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x3 a*wm*unpack-applied-om x2
 p*generic*indifferent x3
 a*fixate*dont-interleave-best p*fixate*newest*interleave-best x3 **chunk-284 chunk-285**
 po*comprehend*actual-context po*probe*with-important-object
 p*fixate*interleave-best x2 a*state*new-important-object*best ao*probe*unpack-probe-om-to-superop-om
chunk-479 ao*probe*unpack-probe-om-to-superop-om chunk-480
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent x2
 a*wm*unpack-applied-om x2
 po*probe*with-important-object

p*probe*retrieved-by-probe*best p*generic*indifferent

159 O: 0959 (fixate :no-space)

ao*fixate chunk-481 ao*fixate chunk-482 p*generic*terminate-and-reject
ao*substate*count-second

a*wm*unpack-applied-om x2 p*generic*reject

ao*fixate*unpack-fixation-object chunk-483 ao*comprehend*unpack-fixation-object **chunk-73**
chunk-290 **chunk-438** ao*fixate*unpack-fixation-object chunk-484
ao*comprehend*unpack-fixation-object **chunk-439**

a*wm*unpack-applied-om x2

po*probe*with-important-object po*imagine*operator-targets

p*probe*new-important-object*best p*probe*new-attribute*best a*fixate*newest x2 p*generic*indifferent
x3

160 O: 0966 (comprehend problem-space :imp-obj condition)

p*fixate*interleave-best x3 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*second
p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer

161 O: 0958 (fixate :condition type s-model-constructor)

ao*fixate chunk-485 ao*fixate chunk-486 ao*fixate chunk-487
p*generic*terminate-and-reject p*probe*new-attribute*best

a*wm*unpack-applied-om x3 p*generic*reject

ao*fixate*unpack-fixation-object chunk-488 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-489 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-490 ao*comprehend*unpack-fixation-object

a*wm*unpack-applied-om x3

po*probe*with-important-object po*imagine*s-model-constructor x2 po*probe*with-important-object
p*probe*new-important-object*best p*probe*new-attribute*best p*imagine*refract a*fixate*newest
p*imagine*refract a*fixate*newest p*probe*new-important-object*best p*generic*indifferent x4

162 O: 0953 (comprehend type :imp-obj action)

p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer

163 O: 0969 (imagine target top-context)

ao*imagine chunk-491 p*probe*non-important-after-imagine*worst x4 p*generic*terminate-and-reject
a*wm*unpack-applied-om x2 p*generic*reject

ao*fixate*unpack-fixation-object chunk-492 ao*comprehend*unpack-fixation-object
ao*fixate*mark-imagined-object x2 chunk-493

a*wm*unpack-applied-om

po*probe*with-important-object p*imagine*refract

p*probe*new-important-object*best p*generic*indifferent

164 O: 0976 (comprehend top-context :imp-obj target)

p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer

165 O: 0968 (imagine target superstate)

ao*imagine chunk-494 p*probe*non-important-after-imagine*worst x4 p*generic*terminate-and-reject
a*wm*unpack-applied-om x2 p*generic*reject

ao*fixate*unpack-fixation-object chunk-495 ao*comprehend*unpack-fixation-object
ao*fixate*mark-imagined-object x2 chunk-496

a*wm*unpack-applied-om

po*comprehend*superstate-target po*fixate*superstate-id po*probe*with-important-object p*imagine*refract
ao*goal-select*comprehend*create-token a*goal-select*proposed-during a*fixate*newest
p*probe*new-important-object*best p*generic*indifferent x3

ao*goal-select*new-goal*fixate/imagine chunk-497

166 O: 0978 (comprehend superstate)
 ao*comprehend*applied ao*comprehend*create-dp-on-om
 a*state*applied-newer **chunk-422 chunk-430**
chunk-421 a*dp*unpack-applied-om x2
 a*dp*unpack-applied-om x3
chunk-422 chunk-430

167 ==>S: S28 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects po*imagine*assertions a*substate*create-time
 a*fixate*newest x4 po*fixate*builds po*fixate*superstate-id po*probe*with-previous-goal
 po*probe*with-part x2 po*probe*with-important-object x8 p*attend*old-regions*reject
 p*generic*indifferent x5
 a*fixate*newest p*fixate*superstate a*fixate*newest p*probe*lhs-best p*probe*repeated-goal*worst
 p*generic*indifferent

168 O: 0992 (comprehend lhs :part)
 p*fixate*interleave-best x4 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 po*fixate*condition x4 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 a*fixate*newest x4 p*generic*indifferent x4
 p*fixate*newest*interleave-best x4
 p*fixate*interleave-best p*fixate*top-down p*fixate*interleave-best x2 p*fixate*top-down
 p*fixate*interleave-best

169 O: 0987 (imagine apply-return-operator)
 ao*imagine chunk-498 ao*imagine chunk-499 p*probe*non-important-after-imagine*worst x3
 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om x4 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-500 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-501 ao*comprehend*unpack-fixation-object
 ao*fixate*mark-imagined-object x2 chunk-502 ao*fixate*mark-imagined-object x2 chunk-503
 a*wm*unpack-applied-om x2
 po*probe*with-important-object p*imagine*refract po*probe*with-important-object p*imagine*refract
 p*probe*new-important-object*best x2 p*generic*indifferent x2

170 O: 0989 (fixate superstate s15 :superstate-id)
 ao*fixate chunk-504 ao*fixate chunk-505 p*fixate*interleave-best x8
 p*generic*terminate-and-reject p*probe*new-attribute*best
 a*wm*unpack-applied-om x2 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-506 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-507 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x2
 po*fixate*shared-state po*probe*with-important-object
 a*fixate*newest p*probe*new-important-object*best p*generic*indifferent x2

171 O: 0986 (imagine apply-add-property)
 ao*imagine chunk-508 ao*imagine chunk-509 p*probe*non-important-after-imagine*worst x3
 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x4 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-510 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-511 ao*comprehend*unpack-fixation-object
 ao*fixate*mark-imagined-object x2 chunk-512 ao*fixate*mark-imagined-object x2 chunk-513
 a*wm*unpack-applied-om x2
 po*probe*with-important-object p*imagine*refract po*probe*with-important-object p*imagine*refract
 p*probe*new-important-object*best x2 p*generic*indifferent x2

- 172 O: 01020 (fixate state :shared-state)
 ao*fixate chunk-514 p*fixate*interleave-best x8 p*generic*terminate-and-reject
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-515 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*probe*with-important-object
 p*probe*new-important-object*best p*generic*indifferent
- 173 O: 0985 (imagine implement-exhausted)
 ao*imagine chunk-516 ao*imagine chunk-517 p*probe*non-important-after-imagine*worst x3
 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x4 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-518 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-519 ao*comprehend*unpack-fixation-object
 ao*fixate*mark-imagined-object x2 chunk-520 ao*fixate*mark-imagined-object x2 chunk-521
 a*wm*unpack-applied-om x2
 po*probe*with-important-object p*imagine*refract po*probe*with-important-object p*imagine*refract
 p*probe*new-important-object*best x2 p*generic*indifferent x2
- 174 O: 01031 (comprehend implement-exhausted :imp-obj assertion)
 p*fixate*interleave-best x8 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*second
 f:apply-sps f:sp f:sp-parts
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
 ao*probe*unpack-probe-om-to-superop-om chunk-522
 a*wm*unpack-applied-om
 p*probe*rhs-better-when-apply-sp
- 175 O: 0984 (imagine apply-create-referent)
 ao*imagine chunk-523 ao*imagine chunk-524 p*probe*non-important-after-imagine*worst x3
 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x4 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-525 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-526 **chunk-28** ao*comprehend*unpack-fixation-object
 ao*fixate*mark-imagined-object x2 chunk-527 ao*fixate*mark-imagined-object x2 chunk-528
 a*wm*unpack-applied-om ao*imagine*imagined-but-seen chunk-529 a*wm*unpack-applied-om
 po*probe*with-important-object p*imagine*refract a*wm*unpack-applied-om po*probe*with-important-object
 p*imagine*refract
 p*probe*new-important-object*best po*comprehend*assertion*imagined po*probe*with-important-object
 p*probe*new-important-object*best p*generic*indifferent x2
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent x2
 ao*goal-select*new-goal*fixate/imagine chunk-530 ao*goal-select*new-goal*fixate/imagine
- 176 O: 01037 (comprehend apply-create-referent :imagined-assertion)
 ao*comprehend*applied f:apply-sps f:sp f:sp-parts ao*comprehend*create-dp-on-om po*display*scroll*to-sp
 x2 **chunk-45**
 a*state*applied-newer a*wm*unpack-applied-om x5 **chunk-430 chunk-422**
 p*display*dunk-comprehend a*display*t431 p*display*dunk-comprehend a*display*t431 p*generic*indifferent
 x2
chunk-429 chunk-268 a*dp*unpack-applied-om x2
 a*dp*unpack-applied-om x6
chunk-430 chunk-422 a*wm*newest-from-not-newest
- 177 O: 01042 (display scroll:to-sp apply-create-referent (t431))
 ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
 a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
 a*wm*unpack-applied-om p*generic*reject ao*display*print*t431

d*print-id*t252 d*print-id*t245 d*print*t236 d*tinit*t225 d*print*tinit

178 O: O1037 (comprehend apply-create-referent :imagined-assertion)

chunk-1 x2 **chunk-21** **chunk-101** **chunk-117** f:apply-sps f:sp f:sp-parts
ao*comprehend*cleanup-naked-region-pointers x8 po*display*scroll*to-sp x2
a*dp*unpack-applied-om p*display*reject-duplicates x2 p*generic*indifferent x2

chunk-22 **chunk-31** **chunk-102** **chunk-32** **chunk-33** **chunk-34** **chunk-35**
chunk-36 **chunk-37** **chunk-118** a*wm*newest-from-not-newest x5
a*wm*unpack-applied-om x6 a*dp*unpack-applied-om x4

chunk-38 **chunk-90** ao*comprehend*unpack-fixation-object **chunk-39** **chunk-91**
ao*comprehend*unpack-fixation-object **chunk-40** ao*comprehend*unpack-fixation-object **chunk-41**
chunk-92 ao*comprehend*unpack-fixation-object **chunk-42** **chunk-93**
ao*comprehend*unpack-fixation-object **chunk-43** **chunk-94** ao*comprehend*unpack-fixation-object
a*wm*unpack-applied-om x6

179 ==>S: S32 (operator no-change)

a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
a*state*important-objects a*substate*create-time

ao*fixated-recently chunk-532 ao*fixated-recently chunk-533 ao*fixated-recently
chunk-534 po*fixate*selected-operator po*fixate*selected-id po*fixate*augmentation x8
po*fixate*assertion x2 po*probe*with-previous-goal po*probe*with-part x2
po*fixate*current-context*shared-state po*probe*with-important-object p*generic*indifferent
a*wm*unpack-applied-om x2 a*fixate*newest p*probe*lhs-best p*probe*rhs-better-when-apply-sp
a*fixate*newest p*probe*repeated-goal*worst x3 p*generic*indifferent
p*probe*fixated-recently*worst po*probe*with-important-object p*probe*fixated-recently*worst
po*probe*with-important-object p*fixate*newest*interleave-best x7
p*probe*fixated-recently*worst x2 p*generic*indifferent x2

180 O: O1047 (attend not-newest constant21)

ao*attend*old-regions chunk-535 p*generic*terminate-and-reject
a*dp*unpack-applied-om p*generic*reject
po*probe*where-was-i p*probe*fixated-recently*best x4
p*probe*where-was-i*best p*generic*indifferent

181 O: O1069 (comprehend s15 :imp-obj superstate)

p*fixate*interleave-best x7 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
ao*substate*count-first **chunk-532** **chunk-533** **chunk-534**

p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
p*fixate*fixated-recently-in-view*best a*state*new-important-object*best
p*probe*retrieved-by-probe*best x2

182 O: O1067 (comprehend shared-state :imp-obj state)

p*fixate*interleave-best x7 ao*probe*goal p*generic*terminate-and-reject ao*substate*count-second
ao*comprehend*applied*second f:shared-states-build-chunks f:superstate-is-shared-too **chunk-532**
chunk-533 **chunk-534**

p*probe*repeated-goal*worst x2 p*generic*reject p*probe*rhs-when-sp a*state*applied-newer
ao*probe*unpack-probe-om-to-superop-om chunk-536 ao*probe*unpack-probe-om-to-superop-om
chunk-537 a*state*new-important-object*best
a*wm*unpack-applied-om x2 p*probe*retrieved-by-probe*best x2
po*comprehend*imagined-chunk-cause po*probe*with-important-object
ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*repeated-goal*worst
p*probe*fixated-recently*best p*generic*indifferent x2

183 O: O1066 (fixate :current-context-ss s-construct)

ao*fixate chunk-538 ao*fixate chunk-539 p*fixate*interleave-best x7
p*generic*terminate-and-reject ao*goal-select*new-goal*fixate/imagine chunk-540
a*wm*unpack-applied-om x2 p*generic*reject
ao*fixate*unpack-fixation-object chunk-541 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-542 **chunk-274** **chunk-283**

ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x2
 po*probe*with-important-object
 p*probe*new-important-object*best p*probe*new-attribute*best p*generic*indifferent

184 O: O1097 (comprehend shared-state :imag-chunk-cause)
 ao*comprehend*applied f:shared-states-build-chunks f:superstate-is-shared-too
 ao*comprehend*create-dp-on-om **chunk-532 chunk-533 chunk-534**
 a*state*applied-newer a*wm*unpack-applied-om x5 **chunk-118 chunk-102 chunk-31 chunk-22**
 po*display*scroll*to-state a*dp*unpack-applied-om x4
 p*display*dunk-comprehend a*display*t445 p*generic*indifferent

185 O: O1103 (display scroll:to-state shared-state (t445))
 ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
 a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
 a*wm*unpack-applied-om p*generic*reject ao*display*t445
 d*print*t373 d*print*t364 d*print*t323

186 O: O1097 (comprehend shared-state :imag-chunk-cause)
chunk-268 chunk-421 chunk-429 f:shared-states-build-chunks
 ao*comprehend*cleanup-naked-region-pointers
 a*dp*unpack-applied-om x9 a*wm*unpack-applied-om x2
chunk-422 chunk-430 a*wm*newest-from-not-newest x2 f:superstate-is-shared-too
 po*display*scroll*to-state
 a*dp*unpack-applied-om x2 p*display*reject-duplicates p*generic*indifferent

187 ==>S: S36 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*fixate*builds po*probe*with-previous-goal po*probe*with-part x2 ao*fixated-recently
chunk-543 ao*fixated-recently chunk-544 ao*fixated-recently chunk-545
 ao*fixated-recently chunk-546 ao*fixated-recently chunk-547 ao*fixated-recently
chunk-548 ao*fixated-recently chunk-549 ao*fixated-recently chunk-550
 po*probe*with-important-object x7 p*generic*indifferent
 a*fixate*newest p*probe*lhs-best a*wm*unpack-applied-om x4 p*probe*repeated-goal*worst
 p*probe*fixated-recently*worst p*probe*repeated-goal*worst p*probe*fixated-recently*worst
 p*probe*repeated-goal*worst p*probe*fixated-recently*worst x2 p*generic*indifferent
 p*probe*fixated-recently*worst po*probe*with-important-object x3 p*probe*fixated-recently*worst
 po*probe*with-important-object
 p*probe*fixated-recently*worst x4 p*generic*indifferent x4

188 O: O1105 (attend not-newest constant55)
 ao*attend*old-regions chunk-551 p*generic*terminate-and-reject
 a*dp*unpack-applied-om p*generic*reject
 po*probe*where-was-i p*probe*fixated-recently*best
 p*probe*where-was-i*best p*generic*indifferent

189 O: O1123 (comprehend for :imp-obj condition)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first f:recall-for-part-of-u-model
chunk-210 chunk-414 chunk-413 ao*substate*count-first **chunk-543 chunk-544**
chunk-545 chunk-546 chunk-547 chunk-548 chunk-549 chunk-550
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
 ao*probe*unpack-probe-om-to-superop-om chunk-552 ao*probe*unpack-probe-om-to-superop-om
chunk-553 a*state*new-important-object*best ao*probe*unpack-probe-om-to-superop-om chunk-554
 ao*probe*unpack-probe-om-to-superop-om chunk-555 a*state*new-important-object*best x4
 a*wm*unpack-applied-om x4 p*probe*retrieved-by-probe*best x7
 po*comprehend*high-level-goal po*probe*with-high-level-goal **chunk-415** po*fixate*chunk*second x2
 po*probe*with-important-object

ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-high-level-goal*best
 a*state*new-important-object*best ao*probe*unpack-probe-om-to-superop-om chunk-556
 p*probe*retrieved-by-probe*best p*generic*indifferent x5
 a*wm*unpack-applied-om
 po*probe*with-important-object
 p*probe*retrieved-by-probe*best p*generic*indifferent

190 O: 01121 (comprehend s-construct :imp-obj current-context)
 ao*probe*goal p*generic*terminate-and-reject ao*substate*count-second ao*comprehend*applied*second
 f:s-construct-builds-chunk **chunk-543 chunk-544 chunk-545 chunk-546 chunk-547**
chunk-548 chunk-549 chunk-550
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
 a*state*new-important-object*best x5
 p*probe*retrieved-by-probe*best x9

191 O: 01119 (comprehend s15 :imp-obj fixated-recently)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied **chunk-543 chunk-544**
chunk-545 chunk-546 chunk-547 chunk-548 chunk-549 chunk-550
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer a*state*new-important-object*best
 x5
 p*probe*retrieved-by-probe*best x9

192 O: 01109 (comprehend lhs :part)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:operator-condition*lhs-means-rhs
chunk-543 chunk-544 chunk-545 chunk-546 chunk-547 chunk-548 chunk-549
chunk-550
 po*fixate*condition x5 po*fixate*no-problem-space p*probe*repeated-goal*worst p*generic*reject
 a*state*applied-newer a*state*new-important-object*best x5
 p*fixate*invariant-feature*dont-interleave-best p*probe*retrieved-by-probe*best x9
 p*generic*indifferent x6
 a*fixate*dont-interleave-best

193 O: 01131 (comprehend obj :imp-obj fixated-recently)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied **chunk-543 chunk-544**
chunk-545 chunk-546 chunk-547 chunk-548 chunk-549 chunk-550
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer a*state*new-important-object*best
 x4
 p*probe*retrieved-by-probe*best x8

194 O: 01129 (comprehend referent-of :imp-obj fixated-recently)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied **chunk-543 chunk-544**
chunk-545 chunk-546 chunk-547 chunk-548 chunk-549 chunk-550
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer a*state*new-important-object*best
 x4
 p*probe*retrieved-by-probe*best x8

195 O: 01135 (comprehend where-was-i |(where-was-i)|)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied **chunk-423 chunk-543**
chunk-544 chunk-545 chunk-546 chunk-547 chunk-548 chunk-549 chunk-550
 po*fixate*where-was-i*chunks-built p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-557
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-558 a*state*new-important-object*best x5
 a*wm*unpack-applied-om x2 p*probe*retrieved-by-probe*best x9 p*generic*indifferent

196 O: 01142 (fixate :second chunk-129)
 ao*fixate chunk-559 p*generic*terminate-and-reject
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-560 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om

po*comprehend*chunk po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*generic*indifferent x2
 ao*goal-select*new-goal*fixate/imagine chunk-561

197 O: 01154 (comprehend chunk-129 :chunk)
 ao*comprehend*applied f:sp ao*comprehend*create-dp-on-om po*display*print-chunk **chunk-543**
chunk-544 chunk-545 chunk-546 chunk-547 chunk-548 chunk-549 chunk-550
 a*state*applied-newer a*wm*unpack-applied-om x6 **chunk-430 chunk-422**
 p*display*dunk-comprehend a*display*t503 p*generic*indifferent
 f:sp-parts a*dp*unpack-applied-om x2
 a*wm*unpack-applied-om x2
 po*comprehend*sp-parts x2
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent x2

198 O: 01158 (display print-chunk chunk-129 (t503))
 ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
 a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
 a*wm*unpack-applied-om p*generic*reject ao*display*print*t503
 d*print*t503

199 O: 01154 (comprehend chunk-129 :chunk)
 f:sp f:sp-parts po*display*print-chunk **chunk-422 chunk-430**
 p*display*reject-duplicates p*generic*indifferent

200 ==>S: S38 (operator no-change)
 a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
 a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
 po*attend
 p*generic*indifferent

201 O: 01164 (attend t503 constant73)
 ao*attend chunk-562 ao*attend*mark-locally p*generic*terminate-and-reject
 a*dp*unpack-applied-om x3 p*generic*reject
 a*wm*newest-from-not-newest
 ao*attend*previous-not-newest chunk-563 po*probe*with-previous-goal po*probe*with-part x2
 po*fixate*current-context po*probe*with-important-object
 a*dp*unpack-applied-om p*probe*fixated-recently*worst p*probe*lhs-best a*fixate*newest
 p*probe*fixated-recently*worst x8 p*probe*repeated-goal*worst x2 p*generic*indifferent

202 O: 01169 (comprehend lhs :part)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs **chunk-282** ao*substate*count-first
 po*fixate*condition po*fixate*no-problem-space x2 p*probe*repeated-goal*worst p*generic*reject
 a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om*fixated chunk-564
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-565 p*fixate*fixated-recently-in-view*best
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x5
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x8 a*wm*unpack-applied-om x2
 p*generic*indifferent
 a*fixate*dont-interleave-best x2 p*fixate*newest*interleave-best a*fixate*dont-interleave-best
 p*fixate*newest*interleave-best x7 po*comprehend*actual-context po*probe*with-important-object
 p*fixate*interleave-best p*fixate*top-down p*fixate*interleave-best p*fixate*top-down
 p*fixate*interleave-best p*fixate*top-down p*fixate*interleave-best p*fixate*top-down
 p*fixate*interleave-best x2 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 p*probe*fixated-recently*worst p*generic*indifferent x2

203 O: 01209 (fixate :no-space)

ao*fixate chunk-566 ao*fixate chunk-567 p*generic*terminate-and-reject
 ao*substate*count-second
 a*wm*unpack-applied-om x2 p*generic*reject p*probe*rhs-when-sp
 ao*fixate*unpack-fixation-object chunk-568 **chunk-73** **chunk-290** **chunk-438** **chunk-483**
 ao*comprehend*unpack-fixation-object ao*fixate*unpack-fixation-object chunk-569 **chunk-439**
chunk-484 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x2
 po*probe*with-important-object
 p*probe*new-important-object*best p*probe*new-attribute*best p*generic*indifferent
 204 O: O1167 (comprehend rhs :part)
 p*fixate*interleave-best x5 ao*goal-select*new-goal*probe chunk-570 ao*probe*goal
 p*generic*terminate-and-reject ao*comprehend*applied*second f:high-level-goal-cues **chunk-443**
chunk-444 **chunk-445** **chunk-449** **chunk-450** **chunk-451** **chunk-455** **chunk-456**
chunk-457 **chunk-460** **chunk-461** **chunk-462** **chunk-466** **chunk-467** **chunk-470**
chunk-471 **chunk-472**
 p*fixate*bottom-up x4 po*fixate*action x4 p*probe*repeated-goal*worst p*generic*reject
 a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om chunk-571
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-572
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-573
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-574
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-576
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-578
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-579
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-580
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-582
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-584
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-585
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-586
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-588
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-590
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-591
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-592
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-594
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-596
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-597
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-598
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-600
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-601
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-602
 ao*probe*unpack-probe-om-to-superop-om*fixated x3 chunk-605
 a*fixate*newest x2 p*fixate*fixated-recently-in-view*best a*fixate*newest x2
 a*wm*unpack-applied-om p*generic*indifferent x4
 p*fixate*newest*interleave-best po*probe*with-high-level-goal po*imagine*action x3
 po*probe*with-important-object x4 po*imagine*action po*probe*with-important-object x5
 po*comprehend*operator-condition po*probe*with-important-object x2
 p*fixate*interleave-best p*probe*new-high-level-goal*best p*imagine*refract
 p*imagine*worst-after-new-output a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output
 a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output a*fixate*newest
 p*probe*fixated-recently*worst p*imagine*worst-after-new-output a*fixate*newest
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent
 205 O: O1161 (comprehend rhs :sp-parts)
 ao*comprehend*applied f:high-level-goal-cues ao*comprehend*create-dp-on-om **chunk-443** **chunk-444**
chunk-445 **chunk-449** **chunk-450** **chunk-451** **chunk-455** **chunk-456** **chunk-457**
chunk-460 **chunk-461** **chunk-462** **chunk-466** **chunk-467** **chunk-470** **chunk-471**
chunk-472
 a*state*applied-newer a*wm*unpack-applied-om **chunk-430** **chunk-422** **chunk-563**
chunk-268 **chunk-421** **chunk-429** a*dp*unpack-applied-om x3
 a*dp*unpack-applied-om x9
chunk-422 **chunk-430** **chunk-563** **chunk-460** **chunk-461** **chunk-462** **chunk-466**
chunk-467 **chunk-470** **chunk-471** **chunk-472** **chunk-443** **chunk-444** **chunk-445**
chunk-449 **chunk-450** **chunk-451** **chunk-455** **chunk-456** **chunk-457**

a*wm*unpack-applied-om

206 ==>S: S39 (operator no-change)

a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
a*state*important-objects a*substate*create-time

po*fixate*builds po*probe*with-previous-goal po*probe*with-part x2 po*fixate*state-id
po*probe*with-high-level-goal po*probe*with-important-object po*fixate*action p*attend*old-regions*reject
p*generic*indifferent

a*fixate*newest p*probe*repeated-goal*worst p*probe*lhs-best a*fixate*newest
p*probe*fixated-recently*worst x7 a*fixate*newest p*generic*indifferent

p*fixate*newest*interleave-best

207 O: O1256 (comprehend lhs :part)

p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
f:operator-condition*lhs-means-rhs **chunk-285 chunk-284** ao*substate*count-first po*imagine*action
x4

po*fixate*condition x6 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
ao*probe*unpack-probe-om-to-superop-om chunk-606 a*state*new-important-object*best
ao*probe*unpack-probe-om-to-superop-om chunk-607 a*fixate*newest p*imagine*refract a*fixate*newest
p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest p*generic*indifferent x4

p*fixate*invariant-feature*dont-interleave-best a*fixate*newest
p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x3
p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x2 a*wm*unpack-applied-om x2
p*generic*indifferent x6

a*fixate*dont-interleave-best p*fixate*newest*interleave-best a*fixate*dont-interleave-best
p*fixate*newest*interleave-best x3 a*fixate*dont-interleave-best po*probe*with-important-object

p*fixate*interleave-best x2 p*probe*retrieved-by-probe*best p*generic*indifferent

208 O: O1299 (fixate :action conjunct-symbol replacement)

ao*fixate chunk-608 ao*fixate chunk-609 ao*fixate chunk-610
p*generic*terminate-and-reject ao*substate*count-second

a*wm*unpack-applied-om x3 p*generic*reject

ao*fixate*unpack-fixation-object chunk-611 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-612 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-613 ao*comprehend*unpack-fixation-object

a*wm*unpack-applied-om x3

po*probe*with-important-object x2

p*probe*new-important-object*best p*probe*new-attribute*best p*probe*new-important-object*best
p*generic*indifferent x2

209 O: O1314 (comprehend replacement :imp-obj action)

p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*second
po*imagine*action x5

p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer a*fixate*newest p*imagine*refract
a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract
a*fixate*newest p*generic*indifferent x5

210 O: O1307 (fixate :condition conjunct-symbol conjunct)

ao*fixate chunk-614 ao*fixate chunk-615 ao*fixate chunk-616
p*generic*terminate-and-reject p*probe*new-attribute*best

a*wm*unpack-applied-om x3 p*generic*reject

ao*fixate*unpack-fixation-object chunk-617 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-618 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-619 ao*comprehend*unpack-fixation-object

a*wm*unpack-applied-om x3

po*probe*with-important-object x2

p*probe*new-important-object*best p*probe*new-attribute*best p*probe*new-important-object*best
p*generic*indifferent x2

211 O: O1312 (comprehend conjunct-symbol :imp-obj action)

p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 po*imagine*action x6

p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer a*fixate*newest
 p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest
 p*imagine*refract a*fixate*newest x2 p*generic*indifferent x6

212 O: 01306 (fixate :condition type s-model-constructor)

ao*fixate chunk-620 ao*fixate chunk-621 ao*fixate chunk-622
 p*generic*terminate-and-reject p*probe*new-attribute*best

a*wm*unpack-applied-om x3 p*generic*reject

ao*fixate*unpack-fixation-object chunk-623 **chunk-488** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-624 **chunk-489** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-625 **chunk-490** ao*comprehend*unpack-fixation-object

a*wm*unpack-applied-om x3

po*probe*with-important-object po*imagine*s-model-constructor x2 po*probe*with-important-object
 p*probe*new-important-object*best p*probe*new-attribute*best p*imagine*refract a*fixate*newest
 p*imagine*refract a*fixate*newest p*probe*new-important-object*best p*generic*indifferent x4

213 O: 01291 (comprehend type :imp-obj action)

ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied po*imagine*action x7

p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer a*fixate*newest
 p*imagine*refract a*fixate*newest x2 p*imagine*refract a*fixate*newest p*imagine*refract
 a*fixate*newest p*imagine*refract a*fixate*newest x2 p*generic*indifferent x7

214 O: 01343 (imagine action conjunct)

ao*imagine chunk-626 p*probe*non-important-after-imagine*worst x4 p*generic*terminate-and-reject

a*wm*unpack-applied-om x2 p*generic*reject

ao*fixate*unpack-fixation-object chunk-627 ao*comprehend*unpack-fixation-object
 ao*fixate*mark-imagined-object x2 chunk-628

a*wm*unpack-applied-om

po*probe*with-important-object

p*probe*new-important-object*best p*generic*indifferent

215 O: 01344 (comprehend conjunct :imp-obj action)

ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied po*imagine*action x7

p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer a*fixate*newest
 p*imagine*refract a*fixate*newest x2 p*imagine*refract a*fixate*newest p*imagine*refract
 a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest
 p*generic*indifferent x7

216 O: 01348 (imagine action s-model-constructor)

ao*imagine chunk-629 p*probe*non-important-after-imagine*worst x4 p*generic*terminate-and-reject

a*wm*unpack-applied-om x2 p*generic*reject

ao*fixate*unpack-fixation-object chunk-630 ao*comprehend*unpack-fixation-object
 ao*fixate*mark-imagined-object x2 chunk-631

a*wm*unpack-applied-om

po*probe*with-important-object

p*probe*new-important-object*best p*generic*indifferent

217 O: 01353 (comprehend s-model-constructor :imp-obj action)

ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:s-construct-builds-chunk
 po*imagine*action x7

p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
 a*state*new-important-object*best a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract
 a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract
 a*fixate*newest p*imagine*refract a*fixate*newest p*generic*indifferent x7

p*probe*retrieved-by-probe*best

218 O: 01310 (comprehend apply-chunks :imp-obj chunk)

ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:abstract-sp po*imagine*action
 x7
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om
chunk-632 a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest
 p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest
 p*imagine*refract a*fixate*newest p*generic*indifferent x7
 a*wm*unpack-applied-om
 f:sp-parts po*probe*with-important-object
 p*probe*repeated-goal*worst p*generic*indifferent
 219 O: 01362 (imagine action nil)
 ao*imagine chunk-633 p*probe*non-important-after-imagine*worst x4 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x2 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-634 ao*comprehend*unpack-fixation-object
 ao*fixate*mark-imagined-object x2 chunk-635
 a*wm*unpack-applied-om
 po*probe*with-important-object
 p*probe*new-important-object*best p*generic*indifferent
 220 O: 01371 (comprehend nil :imp-obj action)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied po*imagine*action x7
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer p*imagine*refract
 a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract
 a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract
 a*fixate*newest p*generic*indifferent x7
 221 O: 01297 (comprehend chunk-129 :imp-obj sp)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:sp f:sp-parts po*imagine*action
 x7
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer p*imagine*refract
 a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract
 a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract
 a*fixate*newest p*generic*indifferent x7
 222 O: 01293 (comprehend u1 :imp-obj action)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied po*imagine*action x7
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer p*imagine*refract a*fixate*newest
 p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest
 p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest
 p*generic*indifferent x7
 223 O: 01289 (comprehend s-model :imp-obj action)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied po*imagine*action x7
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer p*imagine*refract a*fixate*newest
 p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest
 p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest
 p*generic*indifferent x7
 224 O: 01287 (comprehend referent :imp-obj action)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:terminate-create-referent
 po*imagine*action x7
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
 a*state*new-important-object*best ao*probe*unpack-probe-om-to-superop-om chunk-636
 ao*probe*unpack-probe-om-to-superop-om chunk-637 p*imagine*refract a*fixate*newest
 p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest
 p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest
 p*generic*indifferent x7
 a*wm*unpack-applied-om x2
 po*display*match-set*asserted-sp po*probe*with-important-object
 p*display*dunk-comprehend a*display*t513 p*probe*retrieved-by-probe*best p*generic*indifferent x2

225 O: 01408 (display match-set:asserted-sp (t513))
 ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
 p*display*reject-duplicates a*wm*unpack-applied-om p*generic*reject ao*display*print*t513
 d*print*t513

226 O: 01137 (comprehend return-new-pointer :high-lev-goal)
 ao*comprehend*applied ao*comprehend*create-dp-on-om
 a*state*applied-newer **chunk-563 chunk-430 chunk-422**
chunk-446 x2 ao*comprehend*unpack-fixation-object x2 **chunk-54 chunk-452 chunk-54**
chunk-452 ao*comprehend*unpack-fixation-object x2 **chunk-458** x2
 ao*comprehend*unpack-fixation-object x2 **chunk-61 chunk-459 chunk-61 chunk-459**
 ao*comprehend*unpack-fixation-object x2 **chunk-56 chunk-454 chunk-56 chunk-454**
 ao*comprehend*unpack-fixation-object x2 **chunk-55 chunk-453 chunk-55 chunk-453**
 ao*comprehend*unpack-fixation-object x2 **chunk-448** x4 ao*comprehend*unpack-fixation-object
 x4 **chunk-49 chunk-447 chunk-49 chunk-447** ao*comprehend*unpack-fixation-object x2
chunk-208 chunk-463 chunk-208 chunk-463 ao*comprehend*unpack-fixation-object x2
chunk-468 x2 ao*comprehend*unpack-fixation-object x2 **chunk-473** x2
 ao*comprehend*unpack-fixation-object x2 **chunk-475** x2 ao*comprehend*unpack-fixation-object
 x2 **chunk-80 chunk-296 chunk-474 chunk-80 chunk-296 chunk-474**
 ao*comprehend*unpack-fixation-object x2 **chunk-469** x2 ao*comprehend*unpack-fixation-object
 x2 **chunk-465** x2 ao*comprehend*unpack-fixation-object x2 **chunk-464** x2
 ao*comprehend*unpack-fixation-object x2 **chunk-562** a*dp*unpack-applied-om x3
 a*wm*unpack-applied-om a*dp*unpack-applied-om x3
 po*imagine*s-model-constructor po*comprehend*operator-condition **chunk-563** a*wm*newest-from-not-newest
 p*imagine*refract a*fixate*newest ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 p*generic*indifferent x2

227 ==>S: S41 (operator no-change)
 a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
 a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
 po*attend
 p*generic*indifferent

228 O: 01414 (attend t513 constant78)
 ao*attend **chunk-638** ao*attend*mark-locally p*generic*terminate-and-reject
 a*dp*unpack-applied-om x3 p*generic*reject
 a*wm*newest-from-not-newest
 ao*attend*previous-not-newest **chunk-639** po*fixate*builds po*probe*with-previous-goal
 po*fixate*current-context po*imagine*s-model-constructor po*probe*with-high-level-goal
 po*imagine*nil-object x7 po*fixate*assertion x3 po*probe*with-important-object
 a*dp*unpack-applied-om p*imagine*refract p*probe*repeated-goal*worst p*generic*indifferent
 a*fixate*newest
 p*fixate*newest*interleave-best x3
 p*fixate*top-down x2

229 O: 01428 (imagine nil)
 p*fixate*interleave-best x3 ao*imagine **chunk-640** ao*imagine **chunk-641**
 p*probe*non-important-after-imagine*worst x2 p*generic*terminate-and-reject ao*substate*count-first
 p*probe*new-attribute*best x2
 a*wm*unpack-applied-om x4 p*generic*reject
 ao*fixate*unpack-fixation-object **chunk-642 chunk-257** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object **chunk-643 chunk-258** ao*comprehend*unpack-fixation-object
 ao*fixate*mark-imagined-object x2 **chunk-644 chunk-259** ao*fixate*mark-imagined-object x2
chunk-645 chunk-260
 a*wm*unpack-applied-om x2
 po*imagine*postpone po*imagine*nil-object p*imagine*refract x2 po*probe*with-important-object
 a*fixate*newest p*imagine*refract x2 a*fixate*newest p*probe*new-important-object*best
 p*probe*new-attribute*best p*generic*indifferent x3

230 O: 01460 (comprehend nil :imp-obj action)
 p*fixate*interleave-best x3 ao*probe*goal p*generic*terminate-and-reject ao*substate*count-second
 ao*comprehend*applied*first
 p*probe*repeated-goal*worst x3 p*generic*reject a*state*applied-newer

231 O: 01429 (fixate terminate-create-referent :assertion)
 ao*fixate chunk-646 p*generic*terminate-and-reject
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-647 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*comprehend*assertion*real po*probe*with-important-object
 a*topstate*clean-up-old-comprehends-and-displays ao*goal-select*comprehend*create-token
 a*goal-select*proposed-during p*probe*new-important-object*best p*generic*indifferent x2
 ao*goal-select*new-goal*fixate/imagine chunk-648

232 O: 01482 (comprehend terminate-create-referent :assertion)
 ao*comprehend*applied f:sp f:terminating-sps f:sp-parts ao*comprehend*create-dp-on-om
 a*state*applied-newer a*wm*unpack-applied-om x4 **chunk-430 chunk-422 chunk-563**
chunk-639
chunk-429 chunk-421 chunk-268 po*comprehend*sp-parts x2 a*dp*unpack-applied-om x4
 a*dp*unpack-applied-om x9 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent x2
chunk-430 chunk-563 chunk-422

233 ==>S: S42 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*fixate*builds po*probe*with-previous-goal po*probe*with-part x2 po*fixate*current-context
 po*fixate*assertion x2 po*probe*with-important-object p*attend*old-regions*reject
 p*generic*indifferent
 a*fixate*newest p*probe*lhs-best a*fixate*newest x3 p*probe*repeated-goal*worst x2
 p*generic*indifferent
 p*fixate*newest*interleave-best x2
 p*fixate*top-down

234 O: 01494 (comprehend lhs :part)
 p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs **chunk-282** ao*substate*count-first
 po*fixate*condition x6 po*fixate*no-problem-space x2 p*probe*repeated-goal*worst p*generic*reject
 a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om*fixated chunk-649
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-650
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x3
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x4 a*wm*unpack-applied-om x2
 p*generic*indifferent x8
 a*fixate*dont-interleave-best x3 **chunk-285 chunk-284** po*comprehend*actual-context
 po*fixate*state-id po*probe*with-important-object
 ao*probe*unpack-probe-om-to-superop-om chunk-651 a*state*new-important-object*best
 ao*probe*unpack-probe-om-to-superop-om chunk-652 ao*goal-select*comprehend*create-token
 a*goal-select*proposed-during a*fixate*newest p*generic*indifferent x3
 a*wm*unpack-applied-om x2
 po*probe*with-important-object
 p*probe*retrieved-by-probe*best p*generic*indifferent

235 O: 01499 (fixate head-noun-cop-animate :assertion)
 ao*fixate chunk-653 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om p*generic*reject p*probe*rhs-when-sp

- ao*fixate*unpack-fixation-object [chunk-654](#) ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*comprehend*assertion*real po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*generic*indifferent x2
 ao*goal-select*new-goal*fixate/imagine [chunk-655](#)
- 236 O: 01544 (comprehend head-noun-cop-animate :assertion)
 ao*comprehend*applied f:sp f:sps-propose-add-property ao*comprehend*create-dp-on-om
 a*state*applied-newer a*wm*unpack-applied-om x3 **chunk-639 chunk-422 chunk-563
 chunk-430**
chunk-638 chunk-562 f:sp-parts a*dp*unpack-applied-om x4
 a*dp*unpack-applied-om x6 a*wm*unpack-applied-om x2
chunk-639 chunk-563 a*wm*newest-from-not-newest
- 237 ==>S: S43 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*fixate*builds po*probe*with-previous-goal po*probe*with-part x2 po*fixate*state-id
 po*fixate*assertion x2 po*probe*with-important-object x6 p*attend*old-regions*reject
 p*generic*indifferent
 a*fixate*newest p*probe*lhs-best a*fixate*newest x3 p*probe*repeated-goal*worst x2
 p*generic*indifferent
 p*fixate*newest*interleave-best x2
- 238 O: 01552 (comprehend lhs :part)
 p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 po*fixate*condition po*fixate*no-problem-space x2 p*probe*repeated-goal*worst p*generic*reject
 a*state*applied-newer
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x5
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x8 p*generic*indifferent
 a*fixate*dont-interleave-best x3
- 239 O: 01558 (fixate head-noun-cop :assertion)
 ao*fixate [chunk-656](#) p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om p*generic*reject p*probe*rhs-when-sp
 ao*fixate*unpack-fixation-object [chunk-657](#) ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*comprehend*assertion*real po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*generic*indifferent x2
 ao*goal-select*new-goal*fixate/imagine [chunk-658](#)
- 240 O: 01585 (comprehend head-noun-cop :assertion)
 ao*comprehend*applied f:sp f:sps-propose-add-property ao*comprehend*create-dp-on-om
 a*state*applied-newer a*wm*unpack-applied-om x3 **chunk-422 chunk-430 chunk-563
 chunk-639**
chunk-429 chunk-421 chunk-268 f:sp-parts a*dp*unpack-applied-om x4
 a*dp*unpack-applied-om x9 a*wm*unpack-applied-om x2
chunk-430 chunk-563 chunk-422
- 241 ==>S: S44 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*fixate*builds po*probe*with-previous-goal po*probe*with-part x2 po*fixate*current-context

po*fixate*assertion x2 po*probe*with-important-object x4 p*attend*old-regions*reject
 p*generic*indifferent
 a*fixate*newest p*probe*lhs-best a*fixate*newest x3 p*probe*repeated-goal*worst x2
 p*generic*indifferent
 p*fixate*newest*interleave-best x2
 p*fixate*top-down

242 O: 01593 (comprehend lhs :part)
 p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs **chunk-282** ao*substate*count-first
 p*fixate*condition po*fixate*no-problem-space x2 p*probe*repeated-goal*worst p*generic*reject
 a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om*fixated chunk-659
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-660
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x6
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x7 a*wm*unpack-applied-om x2
 p*generic*indifferent
 a*fixate*dont-interleave-best x3 **chunk-284 chunk-285** po*comprehend*actual-context
 po*fixate*state-id po*probe*with-important-object
 a*state*new-important-object*best ao*probe*unpack-probe-om-to-superop-om chunk-661
 ao*probe*unpack-probe-om-to-superop-om chunk-662 ao*goal-select*comprehend*create-token
 a*goal-select*proposed-during a*fixate*newest p*generic*indifferent x3
 a*wm*unpack-applied-om x2
 po*probe*with-important-object
 p*probe*retrieved-by-probe*best p*generic*indifferent

243 O: 01598 (fixate terminate-create-referent :assertion)
 ao*fixate chunk-663 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om p*generic*reject p*probe*rhs-when-sp
 ao*fixate*unpack-fixation-object chunk-664 ao*comprehend*unpack-fixation-object **chunk-647**
 a*wm*unpack-applied-om
 po*comprehend*assertion*real po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*generic*indifferent x2
 ao*goal-select*new-goal*fixate/imagine chunk-665

244 O: 01629 (comprehend terminate-create-referent :assertion)
 ao*comprehend*applied f:sp f:terminating-sps ao*comprehend*create-dp-on-om **chunk-653**
 a*state*applied-newer a*wm*unpack-applied-om x3 **chunk-639 chunk-422 chunk-563**
chunk-430
chunk-638 chunk-562 ao*comprehend*unpack-fixation-object **chunk-654** f:sp-parts
 a*dp*unpack-applied-om x4
 a*dp*unpack-applied-om x6 a*wm*unpack-applied-om x3
chunk-639 chunk-563 a*wm*newest-from-not-newest po*comprehend*assertion*real
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent

245 ==>S: S45 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*fixate*builds po*probe*with-previous-goal po*probe*with-part x2 po*fixate*state-id
 po*fixate*assertion po*probe*with-important-object x7 p*attend*old-regions*reject
 p*generic*indifferent
 a*fixate*newest p*probe*lhs-best a*fixate*newest x2 p*probe*repeated-goal*worst x2
 p*generic*indifferent
 p*fixate*newest*interleave-best

246 O: 01639 (comprehend lhs :part)

p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 po*fixate*condition po*fixate*no-problem-space x2 p*probe*repeated-goal*worst p*generic*reject
 a*state*applied-newer
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x5
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x8 p*generic*indifferent
 a*fixate*dont-interleave-best x3

247 O: 01644 (fixate head-noun-cop :assertion)
 ao*fixate chunk-666 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om p*generic*reject p*probe*rhs-when-sp
 ao*fixate*unpack-fixation-object chunk-667 ao*comprehend*unpack-fixation-object **chunk-657**
 a*wm*unpack-applied-om
 po*comprehend*assertion*real po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*generic*indifferent x2

248 O: 01641 (comprehend rhs :part)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*second f:high-level-goal-cues
chunk-443 chunk-444 chunk-445 chunk-449 chunk-450 chunk-451 chunk-455
chunk-456 chunk-457 chunk-608 chunk-609 chunk-610 chunk-460 chunk-461
chunk-462 chunk-466 chunk-467 chunk-470 chunk-471 chunk-472 chunk-614
chunk-615 chunk-616 chunk-620 chunk-621 chunk-622 chunk-632
 po*fixate*action x4 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 ao*probe*unpack-probe-om-to-superop-om chunk-668 ao*probe*unpack-probe-om-to-superop-om
chunk-669 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-670
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-671
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-672
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-674
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-676
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-677
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-678
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-680
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-682
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-683
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-684
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-686
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-688
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-689
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-690
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-692
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-694
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-695
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-696
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-698
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-700
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-701
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-702
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-704
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-705
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-706
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-708
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-710
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-711
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-712
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-714
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-716
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-717
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-718
 ao*probe*unpack-probe-om-to-superop-om*fixated x3 chunk-721 a*state*new-important-object*best
 ao*probe*unpack-probe-om-to-superop-om chunk-722
 a*fixate*newest x4 a*wm*unpack-applied-om p*probe*retrieved-by-probe*best a*wm*unpack-applied-om
 p*generic*indifferent x4

po*comprehend*high-level-goal po*probe*with-high-level-goal po*imagine*action x2
 po*probe*with-important-object x2 po*imagine*action x2 po*probe*with-important-object x2
chunk-637 po*imagine*action po*probe*with-important-object po*imagine*action x2
 po*probe*with-important-object x2 po*imagine*s-model-constructor x2 po*probe*with-important-object
 x4 po*comprehend*operator-condition po*probe*with-important-object x5
 po*imagine*s-model-constructor x2 po*probe*with-important-object x2
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-high-level-goal*best
 p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest x2 p*imagine*refract
 a*fixate*newest ao*probe*unpack-probe-om-to-superop-om **chunk-723** a*fixate*newest x2
 p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest p*imagine*refract a*fixate*newest
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during a*fixate*newest x2
 p*probe*retrieved-by-probe*best p*generic*indifferent
 a*wm*unpack-applied-om
 249 O: 01653 (comprehend apply-chunks :imp-obj chunk)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:abstract-sp f:sp-parts
 po*imagine*action x7
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer a*fixate*newest
 p*imagine*refract a*fixate*newest x3 p*imagine*refract a*fixate*newest p*imagine*refract
 a*fixate*newest p*imagine*refract a*fixate*newest p*generic*indifferent x7
 250 O: 01729 (imagine condition add-property)
 ao*imagine **chunk-724** p*probe*non-important-after-imagine*worst x4 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x2 p*generic*reject
 po*comprehend*selected-operator ao*fixate*unpack-fixation-object **chunk-725**
 ao*comprehend*unpack-fixation-object ao*fixate*mark-imagined-object x2 **chunk-726**
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during a*wm*unpack-applied-om
 p*generic*indifferent
 ao*goal-select*new-goal*fixate/imagine **chunk-727** po*probe*with-important-object p*imagine*refract
 x2
 p*probe*new-important-object*best p*generic*indifferent
 251 O: 01741 (comprehend add-property :selected-op)
 ao*comprehend*applied f:operator f:add-property-target ao*comprehend*create-dp-on-om
 a*state*applied-newer a*wm*unpack-applied-om x3 **chunk-422 chunk-430 chunk-563**
chunk-639
chunk-429 chunk-421 chunk-268 po*display*run-to-builds a*dp*unpack-applied-om x4
 a*dp*unpack-applied-om x9 p*display*dunk-comprehend a*display*t552 p*generic*indifferent
chunk-430 chunk-563 chunk-422
 252 O: 01745 (display run:to-builds (t552))
 ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
 a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
 a*wm*unpack-applied-om p*generic*reject ao*display*print*t552
 d*print*t552 p*generic*indifferent
 253 O: 01741 (comprehend add-property :selected-op)
 f:operator f:add-property-target po*display*run-to-builds **chunk-422 chunk-430 chunk-563**
chunk-639
 p*display*reject-duplicates a*display*t564 p*generic*indifferent
 254 ==>S: S46 (operator no-change)
 a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
 a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
 po*attend
 p*generic*indifferent
 255 O: 01747 (attend t552 constant96)
 ao*attend **chunk-728** ao*attend*mark-locally p*generic*terminate-and-reject
 a*dp*unpack-applied-om x3 p*generic*reject

a*wm*newest-from-not-newest
 ao*attend*previous-not-newest chunk-729 po*fixate*builds po*probe*with-previous-goal
 po*probe*with-part x2 po*fixate*current-context po*fixate*no-chunk x2
 po*imagine*s-model-constructor x2 po*probe*with-high-level-goal po*fixate*superstate-id
 po*fixate*assertion po*probe*with-important-object
 a*dp*unpack-applied-om p*probe*lhs-best p*imagine*refract p*imagine*worst-after-new-output x2
 p*imagine*refract p*imagine*worst-after-new-output x2 p*probe*repeated-goal*worst x2
 p*generic*indifferent
 a*fixate*newest
 p*fixate*newest*interleave-best
 256 O: 01757 (fixate :no-chunk)
 ao*fixate chunk-730 ao*fixate chunk-731 ao*fixate chunk-732
 p*generic*terminate-and-reject ao*substate*count-first
 a*wm*unpack-applied-om x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-733 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-734 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-735 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x3
 po*probe*with-important-object
 p*probe*new-important-object*best p*generic*indifferent
 257 O: 01751 (comprehend lhs :part)
 ao*probe*goal p*generic*terminate-and-reject ao*substate*count-second ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs **chunk-282**
 po*fixate*condition x3 po*fixate*no-problem-space x2 p*probe*repeated-goal*worst p*generic*reject
 a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om*fixated chunk-736
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-737
 a*fixate*newest x5 a*wm*unpack-applied-om x2 p*generic*indifferent x5
chunk-285 chunk-284 po*comprehend*actual-context po*fixate*state-id
 po*probe*with-important-object
 ao*probe*unpack-probe-om-to-superop-om chunk-738 a*state*new-important-object*best
 ao*probe*unpack-probe-om-to-superop-om chunk-739 ao*goal-select*comprehend*create-token
 a*goal-select*proposed-during a*fixate*newest p*generic*indifferent x3
 a*wm*unpack-applied-om p*probe*retrieved-by-probe*best a*wm*unpack-applied-om
 po*probe*with-important-object
 p*probe*retrieved-by-probe*best p*generic*indifferent
 258 O: 01810 (comprehend apply-chunks :imp-obj sp)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*second f:abstract-sp f:sp-parts
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
 259 O: 01808 (comprehend terminate-create-referent :imp-obj sp)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:sp f:terminating-sps f:sp-parts
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
 260 O: 01806 (comprehend referent-of :imp-obj action)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 261 O: 01804 (comprehend u1 :imp-obj action)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 262 O: 01802 (comprehend type :imp-obj action)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
 263 O: 01800 (comprehend s-model :imp-obj action)

ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 264 O: 01798 (comprehend referent :imp-obj action)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:terminate-create-referent
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
 a*state*new-important-object*best
 p*probe*retrieved-by-probe*best x2
 265 O: 01749 (comprehend terminate-create-referent |(previous-goal)|)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:sp f:terminating-sps f:sp-parts
 p*generic*reject a*state*applied-newer
 266 O: 01796 (comprehend conjunct-symbol :imp-obj action)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
 267 O: 01794 (comprehend replacement :imp-obj action)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 268 O: 01788 (comprehend nil :imp-obj condition)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:nil-chunk-is-abstract
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
 269 O: 01786 (comprehend path-to-referent :imp-obj condition)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 270 O: 01784 (comprehend operator* :imp-obj condition)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:operator-condition*part-of-lhs
chunk-481 chunk-482 chunk-485 chunk-486 chunk-487 chunk-305
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-740
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-741
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-742
 a*wm*unpack-applied-om x3
 po*probe*with-important-object po*comprehend*superstate-target po*imagine*operator-targets x2
 po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*imagine*refract
 p*imagine*worst-after-new-output a*fixate*newest p*fixate*interleave-best
 p*imagine*worst-after-new-output a*fixate*newest p*fixate*interleave-best p*imagine*refract
 p*imagine*worst-after-new-output a*fixate*newest p*fixate*interleave-best
 p*imagine*worst-after-new-output a*fixate*newest p*fixate*interleave-best p*generic*indifferent x7
 271 O: 01833 (imagine target top-context)
 ao*imagine chunk-743 p*probe*non-important-after-imagine*worst x4 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x2 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-744 **chunk-492** ao*comprehend*unpack-fixation-object
 ao*fixate*mark-imagined-object x2 chunk-745 **chunk-493**
 a*wm*unpack-applied-om
 p*imagine*refract x2 po*probe*with-important-object
 p*probe*new-important-object*best p*generic*indifferent
 272 O: 01836 (comprehend top-context :imp-obj target)
 p*fixate*interleave-best x4 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 273 O: 01782 (comprehend s-constructor16 :imp-obj condition)
 p*fixate*interleave-best x4 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 f:operator

- p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
- 274 O: 01778 (comprehend conjunct :imp-obj condition)
 p*fixate*interleave-best x4 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
- 275 O: 01774 (comprehend s-model-constructor :imp-obj condition)
 p*fixate*interleave-best x4 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 f:s-construct-builds-chunk
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer a*state*new-important-object*best
 p*probe*retrieved-by-probe*best x2
- 276 O: 01824 (comprehend apply-chunks :imp-obj chunk)
 p*fixate*interleave-best x4 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 f:abstract-sp f:sp-parts
 p*generic*reject a*state*applied-newer
- 277 O: 01772 (comprehend head-noun-cop :imp-obj assertion)
 p*fixate*interleave-best x4 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:sp
 f:sps-propose-add-property **chunk-663**
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om
chunk-746 a*state*new-important-object*best ao*probe*unpack-probe-om-to-superop-om chunk-747
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-748
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-749
 a*wm*unpack-applied-om p*probe*retrieved-by-probe*best x2 a*wm*unpack-applied-om x3
 f:sp-parts po*probe*with-important-object po*comprehend*selected-operator po*comprehend*assertion*real
 po*probe*with-important-object
 p*probe*repeated-goal*worst ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*repeated-goal*worst
 p*generic*indifferent x4
- 278 O: 01792 (comprehend add-property :imp-obj condition)
 p*fixate*interleave-best x4 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 f:operator f:add-property-target
 p*generic*reject a*state*applied-newer a*state*new-important-object*best
 p*probe*retrieved-by-probe*best
- 279 O: 01768 (comprehend superstate :imp-obj target)
 p*fixate*interleave-best x4 ao*goal-select*new-goal*probe chunk-750 ao*probe*goal
 p*generic*terminate-and-reject ao*comprehend*applied **chunk-504 chunk-505**
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-751
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-752
 a*wm*unpack-applied-om x2
chunk-514 po*fixate*shared-state po*probe*with-important-object
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-753 a*fixate*newest p*generic*indifferent
 x2
 a*wm*unpack-applied-om
 po*probe*with-important-object
 p*generic*indifferent
- 280 O: 01770 (comprehend head-noun-cop-animate :imp-obj assertion)
 p*fixate*interleave-best x4 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:sp
 f:sps-propose-add-property
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om
chunk-754 a*state*new-important-object*best
 a*wm*unpack-applied-om p*probe*retrieved-by-probe*best x2
 f:sp-parts po*probe*with-important-object

p*probe*repeated-goal*worst p*generic*indifferent

281 O: 01764 (comprehend add-property :imp-obj operator*)
p*fixate*interleave-best x4 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
f:operator f:add-property-target
p*generic*reject a*state*applied-newer a*state*new-important-object*best
p*probe*retrieved-by-probe*best

282 O: 01760 (comprehend return-new-pointer |(high-level-goal)|)
p*fixate*interleave-best x4 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer

283 O: 01753 (comprehend rhs :part)
p*fixate*interleave-best x4 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
f:high-level-goal-cues **chunk-443 chunk-444 chunk-445 chunk-449 chunk-450**
chunk-451 chunk-455 chunk-456 chunk-457 chunk-608 chunk-609 chunk-610
chunk-460 chunk-461 chunk-462 chunk-466 chunk-467 chunk-470 chunk-471
chunk-472 chunk-614 chunk-615 chunk-616 chunk-620 chunk-621 chunk-622
po*imagine*action x7 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
p*probe*new-high-level-goal*best
p*imagine*worst-after-new-output a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output
a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output a*fixate*newest p*imagine*refract
a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output a*fixate*newest p*imagine*refract
p*imagine*worst-after-new-output a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output
a*fixate*newest p*generic*indifferent x7

284 O: 01856 (imagine action s-model-constructor)
p*fixate*interleave-best x4 ao*imagine chunk-755 p*probe*non-important-after-imagine*worst
x4 p*generic*terminate-and-reject
a*wm*unpack-applied-om x2 p*generic*reject
ao*fixate*unpack-fixation-object chunk-756 **chunk-630** ao*comprehend*unpack-fixation-object
ao*fixate*mark-imagined-object x2 chunk-757 **chunk-631**
a*wm*unpack-applied-om
po*probe*with-important-object
p*probe*new-important-object*best p*probe*repeated-goal*worst p*generic*indifferent

285 O: 01860 (comprehend s-model-constructor :imp-obj action)
p*fixate*interleave-best x4 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
f:s-construct-builds-chunk po*imagine*action x7
p*generic*reject a*state*applied-newer a*state*new-important-object*best p*imagine*worst-after-new-output
a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output a*fixate*newest
p*imagine*worst-after-new-output a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output
a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output a*fixate*newest p*imagine*refract
p*imagine*worst-after-new-output a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output
a*fixate*newest p*generic*indifferent x7
p*probe*retrieved-by-probe*best x2

286 O: 01864 (imagine action nil)
p*fixate*interleave-best x4 ao*imagine chunk-758 p*probe*non-important-after-imagine*worst
x4 p*generic*terminate-and-reject
a*wm*unpack-applied-om x2 p*generic*reject
ao*fixate*unpack-fixation-object chunk-759 **chunk-634** ao*comprehend*unpack-fixation-object
ao*fixate*mark-imagined-object x2 chunk-760 **chunk-635**
a*wm*unpack-applied-om
po*probe*with-important-object
p*probe*new-important-object*best p*probe*repeated-goal*worst p*generic*indifferent

287 O: 01869 (comprehend nil :imp-obj action)
p*fixate*interleave-best x4 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
f:nil-chunk-is-abstract po*imagine*action x7

- p*generic*reject a*state*applied-newer p*imagine*worst-after-new-output a*fixate*newest
p*imagine*refract p*imagine*worst-after-new-output a*fixate*newest p*imagine*refract
p*imagine*worst-after-new-output a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output
a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output a*fixate*newest p*imagine*refract
p*imagine*worst-after-new-output a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output
a*fixate*newest p*generic*indifferent x7
- 288 O: 01871 (imagine action conjunct)
- p*fixate*interleave-best x4 ao*imagine chunk-761 p*probe*non-important-after-imagine*worst
x4 p*generic*terminate-and-reject
- a*wm*unpack-applied-om x2 p*generic*reject
- ao*fixate*unpack-fixation-object chunk-762 **chunk-627** ao*comprehend*unpack-fixation-object
ao*fixate*mark-imagined-object x2 chunk-763 **chunk-628**
- a*wm*unpack-applied-om
- po*probe*with-important-object
- p*probe*new-important-object*best p*probe*repeated-goal*worst p*generic*indifferent
- 289 O: 01878 (comprehend conjunct :imp-obj action)
- p*fixate*interleave-best x4 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
po*imagine*action x7
- p*generic*reject a*state*applied-newer p*imagine*refract p*imagine*worst-after-new-output
a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output a*fixate*newest p*imagine*refract
p*imagine*worst-after-new-output a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output
a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output a*fixate*newest p*imagine*refract
p*imagine*worst-after-new-output a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output
a*fixate*newest p*generic*indifferent x7
- 290 O: 01748 (fixate builds builds-364 g:builds)
- ao*fixate chunk-764 p*fixate*interleave-best x4 p*generic*terminate-and-reject
- a*wm*unpack-applied-om p*generic*reject
- ao*fixate*unpack-fixation-object chunk-765 **chunk-424** ao*comprehend*unpack-fixation-object
- a*wm*unpack-applied-om
- po*imagine*sp-causes-builds
- p*imagine*worst-after-new-output a*fixate*newest p*generic*indifferent
- 291 O: 01887 (imagine add-property builds)
- p*fixate*interleave-best x4 ao*imagine chunk-766 p*probe*non-important-after-imagine*worst
x4 p*generic*terminate-and-reject p*probe*new-attribute*best x2
- a*wm*unpack-applied-om x2 p*generic*reject
- ao*fixate*unpack-fixation-object chunk-767 ao*comprehend*unpack-fixation-object
ao*fixate*mark-imagined-object x2 chunk-768
- a*wm*unpack-applied-om
- po*display*match-set*after-builds p*imagine*refract
- p*display*dunk-comprehend a*display*t564 p*generic*indifferent
- 292 O: 01888 (display match-set:after-builds (t564))
- ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
- a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
a*wm*unpack-applied-om p*generic*reject ao*display*print*t564
d*print*t564
- 293 O: 01741 (comprehend add-property :selected-op)
- f:operator f:add-property-target ao*comprehend*cleanup-naked-region-pointers x9
po*display*run-to-builds po*display*match-set*after-builds **chunk-563** **chunk-639** **chunk-729**
- p*display*reject-duplicates a*display*t575 p*display*reject-duplicates a*display*t575
p*generic*indifferent x2
- 294 ==>S: S47 (operator no-change)
- a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend x2 a*state*fixate-meta-attributes
a*substate*initialize-goal-set a*state*important-objects a*substate*create-time

po*attend x2
 p*generic*indifferent x2
 295 O: 01893 (attend t564 constant101)
 ao*attend chunk-769 **chunk-769** ao*attend*mark-locally p*generic*terminate-and-reject
 a*dp*unpack-applied-om x4 p*generic*reject
 a*wm*newest-from-not-newest
 ao*attend*previous-not-newest chunk-770 **chunk-770** ao*attend*previous-not-newest
 po*probe*with-previous-goal po*probe*with-part x2 po*fixate*argument po*imagine*sp-causes-builds
 po*imagine*s-model-constructor x2 ao*fixated-recently chunk-772 po*probe*with-high-level-goal
 po*fixate*selected-id po*fixate*assertion po*probe*with-important-object
 a*dp*unpack-applied-om p*probe*lhs-best p*imagine*refract p*imagine*worst-after-new-output x4
 p*imagine*refract p*imagine*worst-after-new-output x4 p*imagine*refract
 p*imagine*worst-after-new-output x4 a*wm*unpack-applied-om p*fixate*nil-assertion-worst
 p*probe*repeated-goal*worst x2 p*generic*indifferent
 po*probe*with-important-object
 a*fixate*newest x9 p*probe*fixated-recently*worst p*generic*indifferent
 p*fixate*newest*interleave-best x4
 296 O: 01900 (fixate add-property :argument t)
 ao*fixate chunk-773 ao*fixate chunk-774 p*generic*terminate-and-reject
 ao*substate*count-first p*probe*new-attribute*best x2
 a*wm*unpack-applied-om x2 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-775 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-776 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x2
 297 O: 01896 (comprehend lhs :part)
 p*fixate*interleave-best x3 ao*probe*goal p*generic*terminate-and-reject ao*substate*count-second
 ao*comprehend*applied*first f:operator-condition*lhs-means-rhs **chunk-772**
 po*fixate*condition x2 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 a*state*new-important-object*best
 a*fixate*newest x2 p*probe*retrieved-by-probe*best p*generic*indifferent x2
 ao*fixate chunk-777 p*generic*terminate-and-reject
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-778 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*imagine*actions-refract po*probe*with-important-object
 p*imagine*worst-after-new-output x2 a*fixate*newest p*probe*new-important-object*best
 p*generic*indifferent x2
 p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*second
 f:nil-chunk-is-abstract **chunk-772**
 p*probe*repeated-goal*worst x4 p*generic*reject a*state*applied-newer
 a*state*new-important-object*best
 p*probe*retrieved-by-probe*best
 300 O: 01906 (fixate o28 :selected-id)
 ao*fixate chunk-779 ao*fixate chunk-780 ao*fixate chunk-781
 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x3 p*generic*reject
 po*comprehend*selected-operator ao*fixate*unpack-fixation-object chunk-782
 ao*comprehend*unpack-fixation-object ao*fixate*unpack-fixation-object chunk-783
 ao*comprehend*unpack-fixation-object ao*fixate*unpack-fixation-object chunk-784
 ao*comprehend*unpack-fixation-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during a*wm*unpack-applied-om x3
 p*generic*indifferent

p*fixate*interleave-best ao*imagine chunk-785 p*probe*non-important-after-imagine*worst x4
 p*generic*terminate-and-reject p*probe*new-attribute*best x2
 a*wm*unpack-applied-om x2 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-786 ao*comprehend*unpack-fixation-object
 ao*fixate*mark-imagined-object x2 chunk-787
 a*wm*unpack-applied-om
 p*imagine*refract

302 O: 01957 (comprehend referent :imp-obj action)
 p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 f:terminate-create-referent **chunk-772**
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
 a*state*new-important-object*best x2
 p*probe*retrieved-by-probe*best x4

303 O: 01977 (comprehend terminate-create-referent :imp-obj sp)
 p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:sp
 f:terminating-sps f:sp-parts **chunk-772**
 p*probe*repeated-goal*worst x3 p*generic*reject a*state*applied-newer
 a*state*new-important-object*best
 p*probe*retrieved-by-probe*best

304 O: 01908 (fixate nil :assertion)
 ao*fixate chunk-788 p*generic*terminate-and-reject
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-789 **chunk-3** ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*comprehend*assertion*real po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*probe*repeated-goal*worst p*generic*indifferent x2
 ao*goal-select*new-goal*fixate/imagine chunk-790

305 O: 01994 (comprehend nil :assertion)
 ao*comprehend*applied f:nil-chunk-is-abstract ao*comprehend*create-dp-on-om **chunk-772**
 a*state*applied-newer a*wm*unpack-applied-om x2 **chunk-729 chunk-639 chunk-770** x2
 po*comprehend*building-agent **chunk-638 chunk-562** a*dp*unpack-applied-om x3
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during a*dp*unpack-applied-om x6
 p*generic*indifferent
chunk-729 chunk-639

306 ==>S: S48 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*probe*with-previous-goal ao*fixated-recently po*fixate*assertion x2 po*probe*with-important-object
 p*attend*old-regions*reject p*generic*indifferent
 a*fixate*newest x2 p*probe*fixated-recently*worst p*probe*repeated-goal*worst x4
 p*generic*indifferent

307 O: 02041 (comprehend head-noun-cop-animate :imp-obj sp)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first f:sp f:sps-propose-add-property
 f:sp-parts **chunk-772 chunk-656** ao*substate*count-first
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer a*state*new-important-object*best
 ao*probe*unpack-probe-om-to-superop-om chunk-792 ao*probe*unpack-probe-om-to-superop-om
chunk-793 a*state*new-important-object*best ao*probe*unpack-probe-om-to-superop-om*fixated
chunk-794
 a*wm*unpack-applied-om x2 p*probe*retrieved-by-probe*best x3 a*wm*unpack-applied-om
 po*comprehend*sp-parts po*probe*with-part po*comprehend*sp-parts po*probe*with-part
 po*comprehend*assertion*real po*probe*with-important-object

ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*lhs-best
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent x6
 308 O: 02005 (comprehend add-property :imp-obj operator*)
 ao*probe*goal p*generic*terminate-and-reject ao*substate*count-second ao*comprehend*applied*second
 f:operator **chunk-772**
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
 a*state*new-important-object*best
 p*probe*retrieved-by-probe*best
 309 O: 02049 (comprehend lhs :part)
 ao*goal-select*new-goal*probe **chunk-795** ao*probe*goal p*generic*terminate-and-reject
 ao*comprehend*applied f:operator-condition*lhs-means-rhs **chunk-772**
 po*fixate*condition x7 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 a*state*new-important-object*best
 a*fixate*newest x7 p*probe*retrieved-by-probe*best p*generic*indifferent x7
 310 O: 02047 (comprehend lhs :sp-parts)
 ao*comprehend*applied f:operator-condition*lhs-means-rhs ao*comprehend*create-dp-on-om **chunk-772**
 a*state*applied-newer a*wm*unpack-applied-om x2 **chunk-770 x2 chunk-639 chunk-729**
chunk-769 x2 chunk-728 a*dp*unpack-applied-om x3
 a*dp*unpack-applied-om x7
chunk-770 x2 chunk-729 a*wm*newest-from-not-newest
 311 ==>S: S49 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*probe*with-previous-goal po*probe*with-part x2 po*fixate*selected-operator po*fixate*selected-id
 po*fixate*assertion x3 po*probe*with-important-object x2 po*fixate*condition x7
 po*fixate*no-problem-space p*attend*old-regions*reject p*generic*indifferent
 p*probe*lhs-best p*probe*repeated-goal*worst a*fixate*newest x5 p*probe*fixated-recently*worst
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x7 p*generic*indifferent
 p*fixate*newest*interleave-best x3 a*fixate*dont-interleave-best x2
 312 O: 02065 (comprehend lhs :part)
 p*fixate*interleave-best x3 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 p*generic*reject a*state*applied-newer
 313 O: 02073 (fixate nil :assertion)
 ao*fixate **chunk-796** p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object **chunk-797** ao*comprehend*unpack-fixation-object **chunk-3**
chunk-789
 a*wm*unpack-applied-om
 po*comprehend*assertion*real po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*generic*indifferent x2
 314 O: 02088 (comprehend nil :imp-obj assertion)
 p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*second
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
 315 O: 02070 (fixate o28 :selected-id)
 ao*fixate **chunk-798** ao*fixate **chunk-799** ao*fixate **chunk-800**
 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x3 p*generic*reject

ao*fixate*unpack-fixation-object chunk-801 ao*comprehend*unpack-fixation-object **chunk-782**
 ao*fixate*unpack-fixation-object chunk-802 ao*comprehend*unpack-fixation-object **chunk-783**
 ao*fixate*unpack-fixation-object chunk-803 ao*comprehend*unpack-fixation-object **chunk-784**
 a*wm*unpack-applied-om x3
 316 O: 02085 (fixate :no-space)
 ao*fixate chunk-804 ao*fixate chunk-805 p*fixate*interleave-best
 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x2 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-806 ao*comprehend*unpack-fixation-object **chunk-73**
chunk-290 chunk-438 chunk-483 chunk-568 ao*fixate*unpack-fixation-object chunk-807
 ao*comprehend*unpack-fixation-object **chunk-439 chunk-484 chunk-569**
 a*wm*unpack-applied-om x2
 po*probe*with-important-object
 p*probe*new-important-object*best p*probe*new-attribute*best p*generic*indifferent
 317 O: 02069 (fixate add-property :selected)
 ao*fixate chunk-808 ao*fixate chunk-809 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x2 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-810 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-811 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x2
 po*probe*with-important-object po*comprehend*selected-operator x3
 p*probe*new-important-object*best ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent x4
 ao*goal-select*new-goal*fixate/imagine chunk-812 ao*goal-select*new-goal*fixate/imagine x2
chunk-814 ao*goal-select*new-goal*fixate/imagine x2 chunk-816
 ao*goal-select*new-goal*fixate/imagine
 318 O: 02098 (comprehend add-property :selected-op)
 ao*comprehend*applied f:operator ao*comprehend*create-dp-on-om **chunk-748 chunk-749**
chunk-746 po*display*match-set*after-selection **chunk-773 chunk-774**
 a*state*applied-newer a*wm*unpack-applied-om x6 **chunk-639 chunk-729 chunk-770** x2
 p*display*dunk-comprehend a*display*t575 p*generic*indifferent
chunk-638 chunk-562 ao*comprehend*unpack-fixation-object **chunk-776**
 ao*comprehend*unpack-fixation-object **chunk-775** po*comprehend*assertion*real a*dp*unpack-applied-om
 x3
 a*dp*unpack-applied-om x6 a*wm*unpack-applied-om x2 ao*goal-select*comprehend*create-token
 a*goal-select*proposed-during p*generic*indifferent
chunk-639 chunk-729
 319 O: 02100 (display match-set:after-selection (t575))
 ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
 a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
 a*wm*unpack-applied-om p*generic*reject ao*display*print*t575
 d*print*t575
 320 O: 02098 (comprehend add-property :selected-op)
 f:operator **chunk-639 chunk-729 chunk-770** x2 po*display*match-set*after-selection
 p*display*reject-duplicates p*generic*indifferent
 321 ==>S: S51 (operator no-change)
 a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
 a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
 po*attend
 p*generic*indifferent
 322 O: 02104 (attend t575 constant115)

ao*attend chunk-818 ao*attend*mark-locally p*generic*terminate-and-reject
 a*dp*unpack-applied-om x3 p*generic*reject
 a*wm*newest-from-not-newest
 ao*attend*previous-not-newest chunk-819 po*probe*with-previous-goal po*probe*with-part
 po*fixate*assertion x4 po*probe*with-important-object x6
 a*dp*unpack-applied-om p*probe*lhs-best p*probe*repeated-goal*worst p*probe*fixated-recently*worst x2
 p*generic*indifferent
 a*fixate*newest x4
 p*fixate*newest*interleave-best x2
 p*fixate*top-down
 323 O: 02105 (comprehend lhs |(previous-goal)|)
 p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 po*fixate*condition x7 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x6 p*generic*indifferent x7
 a*fixate*dont-interleave-best x2
 324 O: 02111 (fixate touch-conjunct-symbol :assertion)
 ao*fixate chunk-820 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-821 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*comprehend*assertion*real po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*generic*indifferent x2
 ao*goal-select*new-goal*fixate/imagine chunk-822
 325 O: 02132 (comprehend touch-conjunct-symbol :assertion)
 ao*comprehend*applied f:sp ao*comprehend*create-dp-on-om
 a*state*applied-newer a*wm*unpack-applied-om **chunk-770 x2 chunk-729 chunk-639
 chunk-819**
chunk-769 x2 chunk-728 f:sp-parts a*dp*unpack-applied-om x4
 a*dp*unpack-applied-om x7 a*wm*unpack-applied-om x2
chunk-770 x2 chunk-819 chunk-729 po*comprehend*sp-parts x2
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent x2
 326 ==>S: S52 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*probe*with-previous-goal po*probe*with-part x2 po*fixate*selected-operator po*fixate*selected-id
 po*fixate*assertion x4 po*probe*with-important-object x5 p*attend*old-regions*reject
 p*generic*indifferent
 p*probe*lhs-best a*fixate*newest x6 p*probe*repeated-goal*worst x2 p*generic*indifferent
 p*fixate*newest*interleave-best
 327 O: 02145 (comprehend lhs :part)
 p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs **chunk-796 chunk-798 chunk-799 chunk-800
 chunk-804 chunk-805 chunk-808 chunk-809** ao*substate*count-first
 po*fixate*condition x7 po*fixate*no-problem-space p*probe*repeated-goal*worst p*generic*reject
 a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om*fixated chunk-823
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-824
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-825
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-826

ao*probe*unpack-probe-om-to-superop-om*fixated chunk-827
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-829
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-831
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-832
 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-833
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-835
 ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-837

 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x7 a*wm*unpack-applied-om x9
 p*generic*indifferent x8

 a*fixate*dont-interleave-best x2 po*comprehend*assertion*real po*probe*with-important-object x2
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent x3
 328 O: 02151 (fixate apply-add-property :assertion)
 ao*fixate chunk-838 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om p*generic*reject p*probe*rhs-when-sp
 ao*fixate*unpack-fixation-object chunk-839 **chunk-511** ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*comprehend*assertion*real po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*generic*indifferent x2
 ao*goal-select*new-goal*fixate/imagine chunk-840
 329 O: 02177 (comprehend apply-add-property :assertion)
 ao*comprehend*applied f:apply-sps f:sp f:apply-add-property-target ao*comprehend*create-dp-on-om
 a*state*applied-newer a*wm*unpack-applied-om x4 **chunk-639 chunk-729 chunk-819**
chunk-770 x2
chunk-818 chunk-638 chunk-562 po*comprehend*building-agent po*comprehend*superstate-target
 po*display*run-to-builds f:sp-parts a*dp*unpack-applied-om x4
 a*dp*unpack-applied-om x9 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*display*dunk-comprehend
 a*display*t582 a*wm*unpack-applied-om x2 p*generic*indifferent x3
chunk-819 chunk-639 chunk-729 a*wm*newest-from-not-newest
 330 O: 02185 (display run:to-builds (t582))
 ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
 a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
 a*wm*unpack-applied-om p*generic*reject ao*display*t582
 d*initial*t582
 331 O: 02177 (comprehend apply-add-property :assertion)
 f:apply-sps f:sp f:sp-parts f:apply-add-property-target po*display*run-to-builds **chunk-639**
chunk-729 chunk-770 x2 **chunk-819**
 p*display*reject-duplicates a*display*t587 p*generic*indifferent
 332 ==>S: S54 (operator no-change)
 a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
 a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
 po*attend
 p*generic*indifferent
 333 O: 02187 (attend t582 constant124)
 ao*attend chunk-841 ao*attend*mark-locally p*generic*terminate-and-reject
 a*dp*unpack-applied-om x3 p*generic*reject
 a*wm*newest-from-not-newest
 ao*attend*previous-not-newest chunk-842 po*fixate*builds po*probe*with-previous-goal
 po*probe*with-part x2 po*fixate*no-chunk po*fixate*assertion x4 po*probe*with-important-object
 x6

a*dp*unpack-applied-om p*probe*rhs-better-when-apply-sp p*probe*lhs-best p*probe*repeated-goal*worst
x2 p*generic*indifferent
a*fixate*newest x6
p*fixate*newest*interleave-best

334 O: 02208 (comprehend touch-conjunct-symbol :imp-obj sp)
p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first f:sp
f:sp-parts ao*substate*count-first
p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer

335 O: 02188 (fixate builds builds-582 g:builds)
ao*fixate chunk-843 p*generic*terminate-and-reject ao*substate*count-second
a*wm*unpack-applied-om p*generic*reject p*probe*rhs-when-sp
ao*fixate*unpack-fixation-object chunk-844 ao*comprehend*unpack-fixation-object
a*wm*unpack-applied-om
po*imagine*sp-causes-builds
p*imagine*worst-after-new-output a*fixate*newest p*generic*indifferent

336 O: 02191 (comprehend rhs :part)
ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*second f:high-level-goal-cues
**chunk-608 chunk-609 chunk-610 chunk-614 chunk-615 chunk-616 chunk-620
chunk-621 chunk-622**
po*fixate*action p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
ao*probe*unpack-probe-om-to-superop-om chunk-845 ao*probe*unpack-probe-om-to-superop-om
chunk-846 ao*probe*unpack-probe-om-to-superop-om*fixated chunk-847
ao*probe*unpack-probe-om-to-superop-om*fixated chunk-848
ao*probe*unpack-probe-om-to-superop-om*fixated chunk-849
ao*probe*unpack-probe-om-to-superop-om*fixated chunk-850
ao*probe*unpack-probe-om-to-superop-om*fixated chunk-851
ao*probe*unpack-probe-om-to-superop-om*fixated chunk-852
ao*probe*unpack-probe-om-to-superop-om*fixated chunk-853
ao*probe*unpack-probe-om-to-superop-om*fixated chunk-854
ao*probe*unpack-probe-om-to-superop-om*fixated chunk-855
a*fixate*newest a*wm*unpack-applied-om p*generic*indifferent
po*comprehend*high-level-goal po*probe*with-high-level-goal po*imagine*action x2
po*probe*with-important-object x6
a*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-high-level-goal*best
p*imagine*worst-after-new-output a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output
a*fixate*newest p*generic*indifferent

337 O: 02193 (comprehend lhs :part)
ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:operator-condition*lhs-means-rhs
po*imagine*action x2
po*fixate*condition x5 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
p*imagine*worst-after-new-output a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output
a*fixate*newest p*generic*indifferent x2
a*fixate*newest x5 p*generic*indifferent x5

338 O: 02212 (imagine apply-add-property builds)
ao*imagine chunk-856 p*probe*non-important-after-imagine*worst x4 p*generic*terminate-and-reject
p*probe*new-attribute*best x2
a*wm*unpack-applied-om x2 p*generic*reject
ao*fixate*unpack-fixation-object chunk-857 ao*comprehend*unpack-fixation-object
ao*fixate*mark-imagined-object x2 chunk-858
a*wm*unpack-applied-om
po*display*match-set*after-builds p*imagine*refract
p*display*dunk-comprehend a*display*t587 p*generic*indifferent

339 O: 02239 (display match-set:after-builds (t587))
ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator ao*display*t582

a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
 a*wm*unpack-applied-om p*generic*reject ao*display*print*t587
 d*print*t587

340 O: 02177 (comprehend apply-add-property :assertion)
 f:apply-sps f:sp f:sp-parts f:apply-add-property-target po*display*run-to-builds
 po*display*match-set*after-builds **chunk-639 chunk-729 chunk-770 x2 chunk-819**
chunk-842 chunk-843
 p*display*reject-duplicates x2 a*wm*unpack-applied-om p*generic*indifferent x2

341 ==>S: S56 (operator no-change)
 a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
 a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
 po*attend
 p*generic*indifferent

342 O: 02242 (attend t587 constant126)
 ao*attend chunk-859 ao*attend*mark-locally p*generic*terminate-and-reject
 a*dp*unpack-applied-om x3 p*generic*reject
 a*wm*newest-from-not-newest
 ao*attend*previous-not-newest chunk-860 po*probe*with-previous-goal po*probe*with-part x2
 po*fixate*no-chunk po*imagine*sp-causes-builds po*probe*with-high-level-goal po*fixate*assertion x5
 po*probe*with-important-object
 a*dp*unpack-applied-om p*probe*rhs-better-when-apply-sp p*probe*lhs-best p*imagine*refract
 p*imagine*worst-after-new-output x4 p*probe*repeated-goal*worst x2 p*generic*indifferent
 a*fixate*newest x7
 p*fixate*newest*interleave-best x2

343 O: 02249 (fixate :no-chunk)
 ao*fixate chunk-861 ao*fixate chunk-862 ao*fixate chunk-863
 p*generic*terminate-and-reject ao*substate*count-first
 a*wm*unpack-applied-om x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-864 **chunk-733** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-865 **chunk-734** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-866 **chunk-735** ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x3
 po*probe*with-important-object
 a*fixate*newest p*probe*new-important-object*best p*generic*indifferent x2

344 O: 02283 (comprehend nil :imp-obj chunk)
 p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*substate*count-second
 ao*comprehend*applied*first f:nil-chunk-is-abstract
 p*probe*repeated-goal*worst x2 p*generic*reject p*probe*rhs-when-sp a*state*applied-newer

345 O: 02257 (fixate terminate-add-property :assertion)
 ao*fixate chunk-867 p*generic*terminate-and-reject
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-868 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*comprehend*assertion*real po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*generic*indifferent x2
 ao*goal-select*new-goal*fixate/imagine chunk-869

346 O: 02285 (comprehend terminate-add-property :assertion)
 ao*comprehend*applied f:sp f:terminating-sps ao*comprehend*create-dp-on-om
 a*state*applied-newer a*wm*unpack-applied-om x2 **chunk-842 chunk-819 chunk-770 x2**

chunk-729 chunk-639 chunk-860

chunk-769 x2 **chunk-728** po*display*run*to-op-after-builds f:sp-parts a*dp*unpack-applied-om x6

a*dp*unpack-applied-om x7 p*display*dunk-comprehend a*display*t593 a*wm*unpack-applied-om x2 p*generic*indifferent

chunk-819 chunk-770 x2 chunk-729

347 O: 02289 (display run:to-op-after-builds (t593))

ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator

a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region a*wm*unpack-applied-om p*generic*reject ao*display*t593

d*run-ms-run*t593

348 O: 02285 (comprehend terminate-add-property :assertion)

f:sp f:terminating-sps f:sp-parts ao*comprehend*cleanup-naked-region-pointers x9 **chunk-770**

x2 **chunk-819 chunk-842 chunk-860** po*display*run*to-op-after-builds

p*display*reject-duplicates p*generic*indifferent

349 ==>S: S57 (operator no-change)

a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend x2 a*state*fixate-meta-attributes a*substate*initialize-goal-set a*state*important-objects a*substate*create-time

po*attend x2

p*generic*indifferent x2

350 O: 02293 (attend t593 constant128)

ao*attend chunk-870 **chunk-870** ao*attend*mark-locally p*generic*terminate-and-reject

a*dp*unpack-applied-om x4 p*generic*reject

a*wm*newest-from-not-newest

ao*attend*previous-not-newest chunk-871 **chunk-871** ao*attend*previous-not-newest

po*probe*with-previous-goal po*probe*with-part x2 po*probe*with-high-level-goal

po*fixate*selected-operator po*fixate*selected-id po*fixate*selected-operator po*fixate*selected-id

po*fixate*assertion x4 po*probe*with-important-object

a*dp*unpack-applied-om p*probe*lhs-best p*fixate*nil-assertion-worst x2 p*probe*repeated-goal*worst x2 p*generic*indifferent

a*fixate*newest

p*fixate*newest*interleave-best x4

351 O: 02298 (comprehend lhs :part)

p*fixate*interleave-best x4 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first

f:operator-condition*lhs-means-rhs **chunk-808 chunk-809 chunk-798 chunk-799**

chunk-800 chunk-796 ao*substate*count-first

p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer

ao*probe*unpack-probe-om-to-superop-om*fixated chunk-873

ao*probe*unpack-probe-om-to-superop-om*fixated chunk-874

ao*probe*unpack-probe-om-to-superop-om*fixated chunk-875

ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-877

ao*probe*unpack-probe-om-to-superop-om*fixated chunk-878

ao*probe*unpack-probe-om-to-superop-om*fixated chunk-879

ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-881

ao*probe*unpack-probe-om-to-superop-om*fixated x2 chunk-883

ao*probe*unpack-probe-om-to-superop-om*fixated chunk-884

a*wm*unpack-applied-om x7

po*imagine*s-model-constructor x2 po*probe*with-important-object po*comprehend*assertion*real

po*probe*with-important-object

a*fixate*newest p*imagine*worst-after-new-output a*fixate*newest ao*goal-select*comprehend*create-token

a*goal-select*proposed-during p*generic*indifferent x5

ao*fixate chunk-885 p*generic*terminate-and-reject ao*substate*count-second

a*wm*unpack-applied-om p*generic*reject ao*goal-select*new-goal*fixate/imagine chunk-886

p*probe*rhs-when-sp

ao*fixate*unpack-fixation-object chunk-887 **chunk-778** ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*probe*with-important-object
 p*probe*new-important-object*best p*generic*indifferent
 353 O: 02338 (comprehend nil :assertion)
 ao*comprehend*applied f:nil-chunk-is-abstract ao*comprehend*create-dp-on-om
 a*state*applied-newer a*wm*unpack-applied-om **chunk-860 chunk-842 chunk-819 chunk-871**
 x2
chunk-859 chunk-841 chunk-818 a*dp*unpack-applied-om x4
 a*dp*unpack-applied-om x9
chunk-860 chunk-871 x2 chunk-842 chunk-819
 354 ==>S: S58 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*fixate*builds po*probe*with-previous-goal po*probe*with-part x2 po*fixate*selected-id
 po*fixate*assertion x3 po*probe*with-important-object x4 p*attend*old-regions*reject
 p*generic*indifferent
 a*fixate*newest p*probe*lhs-best a*fixate*newest x4 p*probe*repeated-goal*worst x2
 p*generic*indifferent
 p*fixate*newest*interleave-best
 355 O: 02350 (comprehend lhs :part)
 p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 356 O: 02352 (fixate o27 :selected-id)
 ao*fixate chunk-888 ao*fixate chunk-889 ao*fixate chunk-890
 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om x3 p*generic*reject
 po*comprehend*selected-operator ao*fixate*unpack-fixation-object chunk-891
 ao*comprehend*unpack-fixation-object ao*fixate*unpack-fixation-object chunk-892
 ao*comprehend*unpack-fixation-object ao*fixate*unpack-fixation-object chunk-893
 ao*comprehend*unpack-fixation-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during a*wm*unpack-applied-om x3
 p*generic*indifferent
 ao*goal-select*new-goal*fixate/imagine chunk-894
 357 O: 02364 (comprehend add-property :selected-op)
 ao*comprehend*applied f:operator ao*comprehend*create-dp-on-om **chunk-820 chunk-773**
chunk-774
 a*state*applied-newer a*wm*unpack-applied-om x4 **chunk-819 chunk-842 chunk-871 x2**
chunk-860
chunk-870 x2 chunk-769 x2 chunk-776 ao*comprehend*unpack-fixation-object
chunk-775 ao*comprehend*unpack-fixation-object **chunk-821** ao*comprehend*unpack-fixation-object
 a*dp*unpack-applied-om x4
 a*dp*unpack-applied-om x8 a*wm*unpack-applied-om x3
chunk-871 x2 chunk-819 chunk-779 chunk-780 chunk-781 chunk-788
 a*wm*newest-from-not-newest po*comprehend*assertion*real
 a*wm*unpack-applied-om x4 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 p*generic*indifferent
 ao*comprehend*unpack-fixation-object **chunk-782 chunk-801** ao*comprehend*unpack-fixation-object
chunk-783 chunk-802 ao*comprehend*unpack-fixation-object **chunk-784 chunk-803**
 ao*comprehend*unpack-fixation-object **chunk-3 chunk-789 chunk-797**
 a*wm*unpack-applied-om x4
 po*comprehend*assertion*real

a*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent

358 ==>S: S59 (operator no-change)

a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
a*state*important-objects a*substate*create-time

po*fixate*builds po*probe*with-previous-goal po*fixate*selected-operator x2
po*fixate*previous-argument po*fixate*assertion x2 po*probe*with-important-object x3
p*attend*old-regions*reject p*generic*indifferent

a*fixate*newest x6 p*probe*repeated-goal*worst p*generic*indifferent
p*fixate*newest*interleave-best x2

359 O: 02376 (fixate policeman :previous-arg)

ao*fixate **chunk-895** p*generic*terminate-and-reject ao*substate*count-first
a*wm*unpack-applied-om p*generic*reject

ao*fixate*unpack-fixation-object **chunk-896** ao*comprehend*unpack-fixation-object
a*wm*unpack-applied-om

360 O: 02383 (comprehend nil :imp-obj assertion)

p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*substate*count-second
ao*comprehend*applied*first

p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer

361 O: 02375 (fixate add-property :selected)

ao*fixate **chunk-897** ao*fixate **chunk-898** p*generic*terminate-and-reject
p*probe*new-important-object*best

a*wm*unpack-applied-om x2 p*generic*reject

ao*fixate*unpack-fixation-object **chunk-899** ao*comprehend*unpack-fixation-object **chunk-811**
a*wm*unpack-applied-om

po*comprehend*selected-operator x4 po*display*match-set*after-selection

ao*goal-select*comprehend*create-token a*goal-select*proposed-during
ao*goal-select*comprehend*create-token a*goal-select*proposed-during
ao*goal-select*comprehend*create-token a*goal-select*proposed-during
ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*display*dunk-comprehend
a*display*t618 p*generic*indifferent x5

362 O: 02393 (display match-set:after-selection (t618))

ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator ao*display*t593

a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
a*wm*unpack-applied-om p*generic*reject ao*display*t618

d*initial*t618 p*generic*indifferent

363 O: 02364 (comprehend add-property :selected-op)

f:operator **chunk-819 chunk-842 chunk-860 chunk-871** x2
po*display*match-set*after-selection

p*display*reject-duplicates p*generic*indifferent

364 ==>S: S62 (operator no-change)

a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend x2 a*state*fixate-meta-attributes
a*substate*initialize-goal-set a*state*important-objects a*substate*create-time

po*attend x2
p*generic*indifferent x2

365 O: 02396 (attend t618 constant139)

ao*attend **chunk-900 chunk-900** ao*attend*mark-locally p*generic*terminate-and-reject
a*dp*unpack-applied-om x4 p*generic*reject

a*wm*newest-from-not-newest

ao*attend*previous-not-newest **chunk-901 chunk-901** ao*attend*previous-not-newest
po*fixate*builds x2 po*probe*with-previous-goal po*fixate*assertion x3
po*probe*with-important-object x3

a*dp*unpack-applied-om p*probe*repeated-goal*worst p*generic*indifferent x9
 a*fixate*newest x5
 p*fixate*newest*interleave-best x2
 366 O: 02408 (comprehend nil :imp-obj assertion)
 p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 ao*substate*count-first
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
 367 O: 02403 (fixate terminate-add-property :assertion)
 ao*fixate chunk-903 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-904 **chunk-868** ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*comprehend*assertion*real po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*generic*indifferent x2
 ao*goal-select*new-goal*fixate/imagine chunk-905
 368 O: 02410 (comprehend terminate-add-property :assertion)
 ao*comprehend*applied f:sp f:terminating-sps ao*comprehend*create-dp-on-om
 a*state*applied-newer a*wm*unpack-applied-om x2 **chunk-871** x2 **chunk-860** **chunk-842**
chunk-819 **chunk-901** x2
chunk-859 **chunk-841** **chunk-818** po*display*run*to-end-of-space x2 f:sp-parts
 a*dp*unpack-applied-om x5
 a*dp*unpack-applied-om x9 p*display*dunk-comprehend a*display*t621 p*display*dunk-comprehend
 a*display*t621 a*wm*unpack-applied-om x2 p*generic*indifferent x2
chunk-871 x2 **chunk-860** **chunk-842** **chunk-819** po*comprehend*sp-parts x2
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent x2
 369 O: 02415 (display run:to-end-of-space (t621))
 ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator ao*display*t618
 a*topstate*clean-up-old-comprehends-and-displays a*wm*unpack-applied-om p*generic*reject ao*display*t621
 d*run-ms-run*t621
 370 O: 02366 (comprehend touch-conjunct-symbol :assertion)
 ao*comprehend*applied f:sp ao*comprehend*create-dp-on-om **chunk-838**
 a*state*applied-newer a*wm*unpack-applied-om x2 **chunk-901** x2 **chunk-871** x2
chunk-860 **chunk-842** **chunk-819**
chunk-900 x2 **chunk-870** x2 **chunk-769** x2 **chunk-511** **chunk-839**
 ao*comprehend*unpack-fixation-object f:sp-parts a*dp*unpack-applied-om x5
 a*dp*unpack-applied-om a*wm*unpack-applied-om x3
chunk-901 x2 **chunk-871** x2 **chunk-819** **chunk-826** **chunk-829** **chunk-827**
chunk-825 **chunk-835** **chunk-837** **chunk-824** **chunk-823** a*wm*newest-from-not-newest
 po*comprehend*assertion*real
 a*wm*unpack-applied-om x6 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 p*generic*indifferent
 po*comprehend*assertion*real
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent
 371 ==>S: S63 (operator no-change)
 a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend x2 a*state*fixate-meta-attributes
 a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
 po*attend x2
 p*generic*indifferent x2

- 372 O: 02426 (attend t621 constant145)
 ao*attend chunk-906 **chunk-906** ao*attend*mark-locally p*generic*terminate-and-reject
 a*dp*unpack-applied-om x4 p*generic*reject
 a*wm*newest-from-not-newest
 ao*attend*previous-not-newest chunk-907 **chunk-907** ao*attend*previous-not-newest
 po*fixate*builds x2 po*probe*with-previous-goal po*probe*with-part x2 po*fixate*selected-id
 po*fixate*selected-operator po*fixate*selected-id po*fixate*assertion x3
 po*probe*with-important-object x4
 a*dp*unpack-applied-om p*probe*lhs-best p*probe*repeated-goal*worst p*generic*indifferent
 a*fixate*newest x8
 p*fixate*newest*interleave-best x2
- 373 O: 02433 (comprehend lhs :part)
 p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
- 374 O: 02437 (fixate o26 :selected-id)
 ao*fixate chunk-909 ao*fixate chunk-910 ao*fixate chunk-911
 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om x3 p*generic*reject p*probe*rhs-when-sp
 ao*fixate*unpack-fixation-object chunk-912 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-913 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-914 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x3
- 375 O: 02431 (comprehend rhs :part)
 p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*second
 f:high-level-goal-cues
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om
chunk-915 ao*probe*unpack-probe-om-to-superop-om chunk-916
 a*wm*unpack-applied-om x2
 po*comprehend*high-level-goal po*probe*with-high-level-goal
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-high-level-goal*best
 p*generic*indifferent x2
- 376 O: 02436 (fixate exhausted :selected)
 ao*fixate chunk-917 ao*fixate chunk-918 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x2 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-919 **chunk-266** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-920 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x2
 po*probe*with-important-object po*comprehend*current-context-when-exhausted
 p*probe*new-important-object*best ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 p*generic*indifferent x2
 ao*goal-select*select-now
- 377 O: 02455 (comprehend current-context :sno)
 ao*comprehend*applied ao*comprehend*create-dp-on-om po*display*print-stack **chunk-272** **chunk-271**
 a*state*applied-newer a*wm*unpack-applied-om x2 **chunk-860** **chunk-842** **chunk-819**
chunk-871 x2 **chunk-901** x2 **chunk-907** x2 p*display*dunk-comprehend a*display*t639
 p*generic*indifferent
chunk-859 **chunk-841** **chunk-818** a*dp*unpack-applied-om x6
 a*dp*unpack-applied-om x9
chunk-860 **chunk-871** x2 **chunk-842** **chunk-819**
- 378 O: 02457 (display print-stack (t639))

ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator ao*display*t621
 a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
 a*wm*unpack-applied-om p*generic*reject ao*display*print*t639
 d*pgs*t639

379 O: O2455 (comprehend current-context :sno)
 ao*comprehend*cleanup-naked-region-pointers po*display*print-stack **chunk-907** x2
 p*display*reject-duplicates p*generic*indifferent

380 ==>S: S65 (operator no-change)
 a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
 a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
 po*attend
 p*generic*indifferent

381 O: O2459 (attend t639 constant148)
 ao*attend chunk-921 ao*attend*mark-locally p*generic*terminate-and-reject
 a*dp*unpack-applied-om x3 p*generic*reject
 a*wm*newest-from-not-newest
 ao*attend*previous-not-newest chunk-922 po*probe*with-previous-goal po*probe*with-part x2
 po*fixate*current-context ao*fixated-recently chunk-923 po*probe*with-high-level-goal x2
 po*probe*with-important-object x4
 a*dp*unpack-applied-om p*probe*lhs-best a*wm*unpack-applied-om p*generic*indifferent
 p*probe*fixated-recently*worst x2 po*probe*with-important-object
 a*fixate*newest p*probe*fixated-recently*worst p*generic*indifferent
 p*fixate*newest*interleave-best

382 O: O2464 (comprehend lhs :part)
 p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first **chunk-923**
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer a*state*new-important-object*best
 p*probe*retrieved-by-probe*best x2

383 O: O2466 (fixate :current-context s-construct)
 ao*fixate chunk-924 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-925 **chunk-274 chunk-283 chunk-542**
 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*comprehend*actual-context po*fixate*state-id po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during a*fixate*newest
 p*probe*new-important-object*best p*generic*indifferent x3
 ao*goal-select*new-goal*fixate/imagine chunk-926 p*fixate*newest*interleave-best

384 O: O2481 (comprehend s-construct :actual-context)
 ao*comprehend*applied f:s-construct-builds-chunk f:recall-u-model ao*comprehend*create-dp-on-om
chunk-923
 a*state*applied-newer a*wm*unpack-applied-om x4 **chunk-922**
chunk-906 x2 **chunk-276** a*dp*unpack-applied-om
 a*dp*unpack-applied-om x4 a*wm*unpack-applied-om
chunk-922 chunk-278 chunk-277
 a*wm*unpack-applied-om x2
 po*comprehend*sp-parts x2
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent x2

385 ==>S: S66 (operator no-change)

a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*probe*with-previous-goal po*probe*with-part x2 po*fixate*state-id po*fixate*selected-operator
 po*fixate*selected-id po*fixate*assertion po*probe*with-important-object x5
 p*attend*old-regions*reject p*generic*indifferent
 p*probe*lhs-best a*fixate*newest x4 p*probe*fixated-recently*worst p*probe*repeated-goal*worst
 p*generic*indifferent
 p*fixate*newest*interleave-best
 386 O: 02495 (comprehend lhs :part)
 p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 387 O: 02497 (fixate state s15 :state-id)
 ao*fixate chunk-927 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-928 **chunk-280** **chunk-306**
 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*comprehend*u-model po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*generic*indifferent x2
 388 O: 02513 (comprehend s15 :imp-obj state)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*second
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 389 O: 02509 (comprehend apply-chunks :imp-obj sp)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:abstract-sp f:sp-parts
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
 390 O: 02505 (comprehend u-model :imp-obj object)
 ao*goal-select*new-goal*probe chunk-929 ao*probe*goal p*generic*terminate-and-reject
 ao*comprehend*applied f:high-level-goal-cues f:u-model
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om
chunk-930 ao*probe*unpack-probe-om-to-superop-om chunk-931
 ao*probe*unpack-probe-om-to-superop-om chunk-932
 a*wm*unpack-applied-om x3
 po*comprehend*high-level-goal po*probe*with-high-level-goal x2
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-high-level-goal*best
 x2 p*generic*indifferent x3
 391 O: 02511 (comprehend u-model)
 ao*comprehend*applied f:high-level-goal-cues f:u-model ao*comprehend*create-dp-on-om
 a*state*applied-newer a*wm*unpack-applied-om x3 **chunk-922**
chunk-921 a*dp*unpack-applied-om
 a*dp*unpack-applied-om x3
chunk-922 a*wm*newest-from-not-newest
 392 ==>S: S67 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*probe*with-previous-goal po*probe*with-part x2 po*fixate*current-context po*imagine*referent
 po*probe*with-high-level-goal po*fixate*selected-operator po*fixate*selected-id po*fixate*assertion
 po*probe*with-important-object x4 p*attend*old-regions*reject p*generic*indifferent
 p*probe*lhs-best a*fixate*newest x5 p*probe*fixated-recently*worst p*probe*repeated-goal*worst
 p*generic*indifferent

p*fixate*newest*interleave-best

393 O: 02529 (imagine :referent)

p*fixate*interleave-best ao*imagine chunk-933 ao*imagine chunk-934
p*probe*non-important-after-imagine*worst x4 p*generic*terminate-and-reject ao*substate*count-first
p*probe*new-attribute*best

a*wm*unpack-applied-om x4 p*generic*reject

ao*fixate*unpack-fixation-object chunk-935 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-936 **chunk-209** ao*comprehend*unpack-fixation-object
ao*fixate*mark-imagined-object x2 chunk-937 ao*fixate*mark-imagined-object x2 chunk-938

a*wm*unpack-applied-om ao*imagine*imagined-but-seen chunk-939 a*wm*unpack-applied-om
p*imagine*refract a*wm*unpack-applied-om p*imagine*refract

po*probe*with-important-object po*display*scroll*to-object
p*display*dunk-comprehend a*display*t646 p*generic*indifferent x2

394 O: 02545 (display scroll:to-object u-model (t646))

ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
a*wm*unpack-applied-om p*generic*reject ao*display*print*t646

d*print*t373 d*print*t364 d*print*t323 d*print-id*t252 d*print-id*t245 p*generic*indifferent

395 O: 02511 (comprehend u-model)

chunk-101 chunk-117 chunk-268 chunk-421 chunk-429 f:high-level-goal-cues f:u-model
ao*comprehend*cleanup-naked-region-pointers x2 po*display*scroll*to-object

a*dp*unpack-applied-om a*wm*unpack-applied-om p*display*reject-duplicates a*display*t649
p*generic*indifferent

chunk-118 chunk-269 chunk-422 chunk-430 a*wm*newest-from-not-newest x5

a*dp*unpack-applied-om x4 **chunk-551 chunk-535**

a*dp*unpack-applied-om

396 ==>S: S71 (operator no-change)

a*subgoal*wm-pointer a*state*fixate-meta-attributes a*substate*initialize-goal-set
a*state*important-objects a*substate*create-time

po*probe*where-was-i po*fixate*builds po*probe*with-previous-goal po*probe*with-part x2
po*fixate*current-context po*imagine*referent ao*fixated-recently chunk-940
po*probe*with-high-level-goal po*fixate*selected-operator po*fixate*selected-id po*fixate*assertion
po*fixate*augmentation x8 po*probe*with-important-object x5 po*fixate*id-of-imagined-object x2

p*probe*where-was-i*best p*probe*lhs-best p*imagine*refract x2 a*wm*unpack-applied-om
p*fixate*u-something p*probe*fixated-recently*best p*probe*repeated-goal*worst p*fixate*u-something
p*generic*indifferent

p*probe*fixated-recently*best po*probe*with-important-object
p*probe*fixated-recently*best p*generic*indifferent

397 O: 02583 (fixate u20 :id-of-imagined-att)

ao*fixate chunk-941 ao*fixate chunk-942 p*generic*terminate-and-reject
ao*substate*count-first p*probe*new-attribute*best

a*wm*unpack-applied-om x2 p*generic*reject

ao*fixate*unpack-fixation-object chunk-943 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-944 **chunk-111** ao*comprehend*unpack-fixation-object

a*wm*unpack-applied-om x2

po*display*print-object*fresh

p*display*dunk-comprehend a*display*t649 p*generic*indifferent

398 O: 02587 (display print-object-fresh u20 (t649))

ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
a*wm*unpack-applied-om p*generic*reject ao*display*print*t649

d*print*t649 d*pgs*t639

399 O: 02511 (comprehend u-model)
chunk-921 f:high-level-goal-cues f:u-model ao*comprehend*cleanup-naked-region-pointers
po*display*scroll*to-object **chunk-940** po*display*print-object*fresh
a*dp*unpack-applied-om x3 p*display*reject-duplicates x2 p*generic*indifferent x2
chunk-922 a*wm*newest-from-not-newest

400 ==>S: S73 (operator no-change)
a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
po*attend
p*generic*indifferent

401 O: 02591 (attend t649 constant157)
ao*attend chunk-945 ao*attend*mark-locally p*generic*terminate-and-reject
a*dp*unpack-applied-om x3 p*generic*reject
a*wm*newest-from-not-newest
ao*attend*previous-not-newest chunk-946 po*probe*with-previous-goal po*probe*with-part x2
po*fixate*current-context po*imagine*referent ao*fixated-recently chunk-947 ao*fixated-recently
chunk-948 po*probe*with-high-level-goal po*fixate*selected-operator po*fixate*selected-id
po*fixate*assertion po*fixate*augmentation x8 po*probe*with-important-object x6
a*dp*unpack-applied-om p*probe*lhs-best p*imagine*refract x2 p*imagine*worst-after-new-output x4
a*wm*unpack-applied-om p*probe*fixated-recently*worst x6 p*probe*repeated-goal*worst
p*generic*indifferent
po*display*scroll*to-object po*probe*with-important-object
a*fixate*newest p*display*reject-duplicates p*probe*fixated-recently*worst p*generic*indifferent x2
p*fixate*newest*interleave-best x8

402 O: 02596 (comprehend lhs :part)
p*fixate*interleave-best x8 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
f:operator-condition*lhs-means-rhs ao*substate*count-first **chunk-947 chunk-948**
p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer p*fixate*fixated-recently-in-view*best
x8 a*state*new-important-object*best
p*probe*retrieved-by-probe*best

403 O: 02612 (fixate referent r9 :augmentation)
ao*fixate chunk-949 ao*fixate chunk-950 ao*fixate chunk-951 ao*fixate chunk-952
ao*fixate chunk-953 ao*fixate chunk-954 p*generic*terminate-and-reject
ao*substate*count-second p*probe*new-attribute*best x2
a*wm*unpack-applied-om x6 p*generic*reject
ao*fixate*unpack-fixation-object chunk-955 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-956 **chunk-126** ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-957 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-958 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-959 ao*comprehend*unpack-fixation-object
a*wm*unpack-applied-om x5
po*probe*with-attribute po*comprehend*objects-attribute
p*probe*new-attribute*best ao*goal-select*comprehend*create-token a*goal-select*proposed-during
p*generic*indifferent x2
ao*goal-select*new-goal*fixate/imagine chunk-960

404 O: 02630 (comprehend referent :objects-att)
ao*comprehend*applied ao*comprehend*create-dp-on-om **chunk-947 chunk-948** po*display*print-object
a*state*applied-newer a*wm*unpack-applied-om **chunk-922 chunk-946** p*display*dunk-comprehend
a*display*t654 p*generic*indifferent
chunk-906 x2 a*dp*unpack-applied-om x2
a*dp*unpack-applied-om x4
chunk-922

405 O: 02632 (display print-object r9 (t654))
 ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
 a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
 a*wm*unpack-applied-om p*generic*reject ao*display*print*t654
 d*print*t654

406 O: 02630 (comprehend referent :objects-att)
chunk-922 chunk-946 chunk-947 chunk-948 po*display*print-object
 p*display*reject-duplicates p*generic*indifferent

407 ==>S: S74 (operator no-change)
 a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
 a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
 po*attend
 p*generic*indifferent

408 O: 02635 (attend t654 constant159)
 ao*attend chunk-961 ao*attend*mark-locally p*generic*terminate-and-reject
 a*dp*unpack-applied-om x3 p*generic*reject
 a*wm*newest-from-not-newest
 ao*attend*previous-not-newest chunk-962 po*probe*with-previous-goal po*fixate*current-context
 ao*fixated-recently x2 po*probe*with-high-level-goal po*probe*with-attribute
 po*fixate*selected-operator po*fixate*selected-id po*fixate*augmentation x7 po*fixate*assertion
 po*fixate*augmentation x3 po*probe*with-important-object x3
 a*dp*unpack-applied-om p*probe*repeated-goal*worst p*probe*fixated-recently*worst x4
 p*probe*repeated-goal*worst p*generic*indifferent
 a*fixate*newest
 p*fixate*newest*interleave-best x3

409 O: 02655 (fixate properties p88 :augmentation)
 ao*fixate chunk-965 ao*fixate chunk-966 ao*fixate chunk-967 ao*fixate chunk-968
 ao*fixate chunk-969 ao*fixate chunk-970 p*generic*terminate-and-reject
 ao*substate*count-first
 a*wm*unpack-applied-om x6 p*generic*reject p*fixate*fixated-recently-in-view*best
 ao*fixate*unpack-fixation-object chunk-971 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-972 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-973 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-974 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-975 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-976 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x6
 po*probe*with-attribute po*fixate*two-valued-attribute
 p*probe*new-attribute*best a*fixate*newest p*generic*indifferent x2
 p*fixate*newest*interleave-best

410 O: 02653 (fixate referent-of u20 :augmentation)
 ao*fixate chunk-977 ao*fixate chunk-978 ao*fixate chunk-979 ao*fixate chunk-980
 ao*fixate chunk-981 ao*fixate chunk-982 p*generic*terminate-and-reject
 ao*substate*count-second
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-983 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-984 **chunk-112** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-985 **chunk-41 chunk-92**
 ao*comprehend*unpack-fixation-object ao*fixate*unpack-fixation-object chunk-986
 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x4
 po*probe*with-attribute

p*probe*new-attribute*best p*generic*indifferent

411 O: 02651 (fixate head u17 :augmentation)

ao*fixate chunk-987 ao*fixate chunk-988 ao*fixate chunk-989 ao*fixate chunk-990
ao*fixate chunk-991 ao*fixate chunk-992 p*fixate*interleave-best x2
p*generic*terminate-and-reject p*probe*new-attribute*best
a*wm*unpack-applied-om x6 p*generic*reject
ao*fixate*unpack-fixation-object chunk-993 **chunk-125** ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-994 **chunk-127** ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-995 **chunk-135** ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-996 **chunk-136** ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-997 **chunk-137** ao*comprehend*unpack-fixation-object
a*wm*unpack-applied-om x5
po*probe*with-attribute
p*probe*new-attribute*best p*generic*indifferent

412 O: 02654 (fixate properties p87 :augmentation)

ao*fixate chunk-998 ao*fixate chunk-999 ao*fixate chunk-1000 ao*fixate chunk-1001
ao*fixate chunk-1002 ao*fixate chunk-1003 p*generic*terminate-and-reject
p*probe*new-attribute*best
a*wm*unpack-applied-om x6 p*generic*reject
ao*fixate*unpack-fixation-object chunk-1004 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-1005 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-1006 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-1007 ao*comprehend*unpack-fixation-object
a*wm*unpack-applied-om x4
po*probe*with-attribute po*comprehend*objects-attribute po*fixate*two-valued-attribute
po*comprehend*objects-attribute
p*probe*new-attribute*best ao*goal-select*comprehend*create-token a*goal-select*proposed-during
a*fixate*newest ao*goal-select*comprehend*create-token a*goal-select*proposed-during
p*generic*indifferent x4
ao*goal-select*new-goal*fixate/imagine chunk-1008 p*fixate*newest*interleave-best
ao*goal-select*new-goal*fixate/imagine chunk-1009

413 O: 02674 (comprehend properties :objects-att)

ao*comprehend*applied ao*comprehend*create-dp-on-om **chunk-947 chunk-948**
a*state*applied-newer a*wm*unpack-applied-om **chunk-946 chunk-922 chunk-962**
chunk-126 chunk-956 ao*comprehend*unpack-fixation-object **chunk-111 chunk-944**
ao*comprehend*unpack-fixation-object **chunk-945 chunk-921** a*dp*unpack-applied-om x3
a*wm*unpack-applied-om x2 a*dp*unpack-applied-om x6
chunk-946 chunk-962 chunk-922

414 ==>S: S75 (operator no-change)

a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
a*state*important-objects a*substate*create-time
po*probe*with-previous-goal po*fixate*current-context po*probe*with-attribute x4
po*fixate*selected-operator po*fixate*selected-id po*fixate*two-valued-attribute x2
po*fixate*assertion po*fixate*augmentation x7 po*probe*with-important-object
p*attend*old-regions*reject p*generic*indifferent
a*fixate*newest p*probe*repeated-goal*worst x2 a*fixate*newest p*probe*fixated-recently*worst
p*generic*indifferent
p*fixate*newest*interleave-best x2

415 O: 02699 (fixate referent r9 :augmentation)

ao*fixate chunk-1010 ao*fixate chunk-1011 ao*fixate chunk-1012 ao*fixate
chunk-1013 ao*fixate chunk-1014 ao*fixate chunk-1015 p*fixate*interleave-best x2
p*generic*terminate-and-reject ao*substate*count-first p*probe*new-attribute*best x2
a*wm*unpack-applied-om x6 p*generic*reject p*fixate*fixated-recently-in-view*best x7
ao*fixate*unpack-fixation-object chunk-1016 ao*comprehend*unpack-fixation-object **chunk-955**

ao*fixate*unpack-fixation-object chunk-1017 ao*comprehend*unpack-fixation-object **chunk-957**
 ao*fixate*unpack-fixation-object chunk-1018 ao*comprehend*unpack-fixation-object **chunk-958**
 ao*fixate*unpack-fixation-object chunk-1019 ao*comprehend*unpack-fixation-object **chunk-209**
chunk-936 ao*fixate*unpack-fixation-object chunk-1020 ao*comprehend*unpack-fixation-object
chunk-959
 a*wm*unpack-applied-om x5
 po*probe*with-attribute
 p*probe*new-attribute*best p*generic*indifferent
 416 O: 02698 (fixate zero-head u15 :augmentation)
 ao*fixate chunk-1021 ao*fixate chunk-1022 ao*fixate chunk-1023 ao*fixate
chunk-1024 ao*fixate chunk-1025 ao*fixate chunk-1026 p*fixate*interleave-best x2
 p*generic*terminate-and-reject ao*substate*count-second p*probe*new-attribute*best
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1027 **chunk-146** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1028 **chunk-147** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1029 **chunk-148** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1030 **chunk-149** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1031 **chunk-150** ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x5
 po*probe*with-attribute
 p*probe*new-attribute*best p*generic*indifferent
 417 O: 02697 (fixate spec u14 :augmentation)
 ao*fixate chunk-1032 ao*fixate chunk-1033 ao*fixate chunk-1034 ao*fixate
chunk-1035 ao*fixate chunk-1036 ao*fixate chunk-1037 p*fixate*interleave-best x2
 p*generic*terminate-and-reject p*probe*new-attribute*best
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1038 **chunk-158** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1039 **chunk-159** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1040 **chunk-160** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1041 **chunk-161** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1042 **chunk-162** ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x5
 po*probe*with-attribute
 p*probe*new-attribute*best p*generic*indifferent
 418 O: 02696 (fixate empty-node e15 :augmentation)
 ao*fixate chunk-1043 ao*fixate chunk-1044 ao*fixate chunk-1045 ao*fixate
chunk-1046 ao*fixate chunk-1047 ao*fixate chunk-1048 p*fixate*interleave-best x2
 p*generic*terminate-and-reject p*probe*new-attribute*best
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1049 **chunk-170** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1050 **chunk-171** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1051 **chunk-172** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1052 **chunk-173** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1053 **chunk-174** ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x5
 po*probe*with-attribute
 p*probe*new-attribute*best p*generic*indifferent
 419 O: 02695 (fixate right-edge w13 :augmentation)
 ao*fixate chunk-1054 ao*fixate chunk-1055 ao*fixate chunk-1056 ao*fixate
chunk-1057 ao*fixate chunk-1058 ao*fixate chunk-1059 p*fixate*interleave-best x2
 p*generic*terminate-and-reject p*probe*new-attribute*best
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1060 **chunk-182** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1061 **chunk-183** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1062 **chunk-184** ao*comprehend*unpack-fixation-object

ao*fixate*unpack-fixation-object chunk-1063 **chunk-185** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1064 **chunk-186** ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x5
 po*probe*with-attribute
 p*probe*new-attribute*best p*generic*indifferent
 420 O: 02693 (fixate left-edge w8 :augmentation)
 ao*fixate chunk-1065 ao*fixate chunk-1066 ao*fixate chunk-1067 ao*fixate
chunk-1068 ao*fixate chunk-1069 ao*fixate chunk-1070 p*fixate*interleave-best x2
 p*generic*terminate-and-reject p*probe*new-attribute*best
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1071 **chunk-194** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1072 **chunk-195** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1073 **chunk-196** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1074 **chunk-197** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1075 **chunk-198** ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x5
 po*probe*with-attribute
 p*probe*new-attribute*best p*generic*indifferent
 421 O: 02691 (fixate :two properties)
 ao*fixate chunk-1076 p*generic*terminate-and-reject
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1077 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 422 O: 02692 (fixate nil :assertion)
 ao*fixate chunk-1078 p*generic*terminate-and-reject
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1079 **chunk-3** **chunk-789** **chunk-797**
 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*comprehend*assertion*real po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*generic*indifferent x2
 ao*goal-select*new-goal*fixate/imagine chunk-1080
 423 O: 02714 (comprehend nil :assertion)
 ao*comprehend*applied ao*comprehend*create-dp-on-om
 a*state*applied-newer **chunk-922** **chunk-962** **chunk-946**
chunk-961 **chunk-906** x2 a*dp*unpack-applied-om x3
 a*dp*unpack-applied-om x7
chunk-962 **chunk-922** a*wm*newest-from-not-newest
 424 ==>S: S76 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*probe*with-previous-goal po*fixate*current-context po*probe*with-attribute x6
 po*fixate*selected-operator po*fixate*selected-id po*fixate*augmentation x4
 po*probe*with-important-object x2 p*attend*old-regions*reject p*generic*indifferent
 a*fixate*newest x7 p*probe*fixated-recently*worst p*probe*repeated-goal*worst p*generic*indifferent
 p*fixate*newest*interleave-best x3
 425 O: 02739 (fixate properties p88 :augmentation)
 ao*fixate chunk-1081 ao*fixate chunk-1082 ao*fixate chunk-1083 ao*fixate
chunk-1084 ao*fixate chunk-1085 ao*fixate chunk-1086 p*generic*terminate-and-reject
 ao*substate*count-first p*probe*new-attribute*best

a*wm*unpack-applied-om x6 p*generic*reject p*fixate*fixated-recently-in-view*best x5
 ao*fixate*unpack-fixation-object chunk-1087 ao*comprehend*unpack-fixation-object **chunk-971**
 ao*fixate*unpack-fixation-object chunk-1088 ao*comprehend*unpack-fixation-object **chunk-972**
 ao*fixate*unpack-fixation-object chunk-1089 ao*comprehend*unpack-fixation-object **chunk-973**
 ao*fixate*unpack-fixation-object chunk-1090 ao*comprehend*unpack-fixation-object **chunk-974**
 ao*fixate*unpack-fixation-object chunk-1091 ao*comprehend*unpack-fixation-object **chunk-975**
 ao*fixate*unpack-fixation-object chunk-1092 ao*comprehend*unpack-fixation-object **chunk-976**
 a*wm*unpack-applied-om x6
 po*probe*with-attribute
 p*probe*new-attribute*best p*generic*indifferent
 426 O: 02737 (fixate referent-of u20 :augmentation)
 ao*fixate chunk-1093 ao*fixate chunk-1094 ao*fixate chunk-1095 ao*fixate
chunk-1096 ao*fixate chunk-1097 ao*fixate chunk-1098 p*generic*terminate-and-reject
 ao*substate*count-second
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1099 ao*comprehend*unpack-fixation-object **chunk-983**
 ao*fixate*unpack-fixation-object chunk-1100 ao*comprehend*unpack-fixation-object **chunk-112**
chunk-984 ao*fixate*unpack-fixation-object chunk-1101 ao*comprehend*unpack-fixation-object
chunk-41 chunk-92 chunk-985 ao*fixate*unpack-fixation-object chunk-1102
 ao*comprehend*unpack-fixation-object **chunk-986**
 a*wm*unpack-applied-om x4
 po*probe*with-attribute
 p*probe*new-attribute*best p*generic*indifferent
 427 O: 02736 (fixate head u17 :augmentation)
 ao*fixate chunk-1103 ao*fixate chunk-1104 ao*fixate chunk-1105 ao*fixate
chunk-1106 ao*fixate chunk-1107 ao*fixate chunk-1108 p*fixate*interleave-best
 p*generic*terminate-and-reject p*probe*new-attribute*best
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1109 ao*comprehend*unpack-fixation-object **chunk-125**
chunk-993 ao*fixate*unpack-fixation-object chunk-1110 ao*comprehend*unpack-fixation-object
chunk-127 chunk-994 ao*fixate*unpack-fixation-object chunk-1111
 ao*comprehend*unpack-fixation-object **chunk-135 chunk-995** ao*fixate*unpack-fixation-object
chunk-1112 ao*comprehend*unpack-fixation-object **chunk-136 chunk-996**
 ao*fixate*unpack-fixation-object chunk-1113 ao*comprehend*unpack-fixation-object **chunk-137**
chunk-997
 a*wm*unpack-applied-om x5
 po*probe*with-attribute
 p*probe*new-attribute*best p*generic*indifferent
 428 O: 02738 (fixate properties p87 :augmentation)
 ao*fixate chunk-1114 ao*fixate chunk-1115 ao*fixate chunk-1116 ao*fixate
chunk-1117 ao*fixate chunk-1118 ao*fixate chunk-1119 p*generic*terminate-and-reject
 p*probe*new-attribute*best x2
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1120 ao*comprehend*unpack-fixation-object **chunk-1004**
 ao*fixate*unpack-fixation-object chunk-1121 ao*comprehend*unpack-fixation-object **chunk-1005**
 ao*fixate*unpack-fixation-object chunk-1122 ao*comprehend*unpack-fixation-object **chunk-1006**
 ao*fixate*unpack-fixation-object chunk-1123 ao*comprehend*unpack-fixation-object **chunk-1007**
 a*wm*unpack-applied-om x4
 po*probe*with-attribute po*comprehend*objects-attribute x2
 p*probe*new-attribute*best ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent x3
 429 O: 02719 (comprehend properties |(previous-goal)|)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 p*probe*repeated-goal*worst x3 p*generic*reject a*state*applied-newer
 430 O: 02735 (fixate o26 :selected-id)

ao*fixate chunk-1124 ao*fixate chunk-1125 ao*fixate chunk-1126
 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1127 **chunk-912** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1128 **chunk-913** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1129 **chunk-914** ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x3
 431 O: 02734 (fixate exhausted :selected)
 ao*fixate chunk-1130 ao*fixate chunk-1131 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x2 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1132 **chunk-266 chunk-919**
 ao*comprehend*unpack-fixation-object ao*fixate*unpack-fixation-object chunk-1133 **chunk-920**
 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x2
 po*probe*with-important-object po*comprehend*current-context-when-exhausted
 po*comprehend*selected-operator x3
 p*probe*new-important-object*best ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent x5
 ao*goal-select*select-now ao*goal-select*new-goal*fixate/imagine chunk-1134
 ao*goal-select*new-goal*fixate/imagine x2 chunk-1136 ao*goal-select*new-goal*fixate/imagine
 x2 chunk-1138 ao*goal-select*new-goal*fixate/imagine
 432 O: 02764 (comprehend exhausted :selected-op)
 ao*comprehend*applied f:high-level-goal-cues f:operator f:exhausted-builds-proposal
 ao*comprehend*create-dp-on-om po*display*match-set*after-selection
 a*state*applied-newer a*wm*unpack-applied-om x4 **chunk-946 chunk-922 chunk-962**
 p*display*dunk-comprehend a*display*t685 p*generic*indifferent
chunk-126 chunk-956 ao*comprehend*unpack-fixation-object **chunk-111 chunk-944**
 ao*comprehend*unpack-fixation-object **chunk-945 chunk-921** po*comprehend*high-level-goal
 a*dp*unpack-applied-om x3
 a*wm*unpack-applied-om x2 a*dp*unpack-applied-om x6 ao*goal-select*comprehend*create-token
 a*goal-select*proposed-during p*generic*indifferent
chunk-946 chunk-962 chunk-922
 433 O: 02766 (display match-set:after-selection (t685))
 ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
 a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
 a*wm*unpack-applied-om p*generic*reject ao*display*print*t685
 d*print*t685
 434 O: 02764 (comprehend exhausted :selected-op)
 f:high-level-goal-cues f:operator f:exhausted-builds-proposal **chunk-922 chunk-946 chunk-962**
 po*display*match-set*after-selection
 p*display*reject-duplicates p*generic*indifferent
 435 ==>S: S78 (operator no-change)
 a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
 a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
 po*attend
 p*generic*indifferent
 436 O: 02770 (attend t685 constant172)
 ao*attend chunk-1140 ao*attend*mark-locally p*generic*terminate-and-reject
 a*dp*unpack-applied-om x3 p*generic*reject
 a*wm*newest-from-not-newest

ao*attend*previous-not-newest chunk-1141 po*probe*with-previous-goal po*fixate*current-context
 po*probe*with-high-level-goal po*probe*with-attribute *x4* po*fixate*two-valued-attribute *x2*
 po*fixate*assertion po*fixate*augmentation *x7* po*fixate*assertion po*probe*with-important-object
 a*dp*unpack-applied-om p*probe*repeated-goal*worst p*generic*indifferent
 a*fixate*newest
 p*fixate*newest*interleave-best

437 O: 02794 (fixate implement-exhausted :assertion)
 ao*fixate chunk-1142 p*generic*terminate-and-reject ao*substate*count-first
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1143 **chunk-519** ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*comprehend*assertion*real po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*generic*indifferent *x2*

438 O: 02799 (comprehend implement-exhausted :imp-obj assertion)
 ao*probe*goal p*generic*terminate-and-reject ao*substate*count-second ao*comprehend*applied*first
 f:apply-sps f:sp
 p*probe*repeated-goal*worst p*generic*reject ao*goal-select*new-goal*probe chunk-1144
 a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om chunk-1145
 ao*probe*unpack-probe-om-to-superop-om chunk-1146
 a*wm*unpack-applied-om *x2*
 f:sp-parts po*probe*with-important-object
 ao*probe*unpack-probe-om-to-superop-om chunk-1147 ao*probe*unpack-probe-om-to-superop-om
chunk-1148 p*probe*repeated-goal*worst p*generic*indifferent
 a*wm*unpack-applied-om *x2*
 po*comprehend*sp-parts po*probe*with-part po*comprehend*sp-parts po*probe*with-part
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*rhs-better-when-apply-sp
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*lhs-best
 p*generic*indifferent *x4*

439 O: 02797 (comprehend implement-exhausted :assertion)
 ao*comprehend*applied f:apply-sps f:sp f:sp-parts ao*comprehend*create-dp-on-om
 po*display*print-sp*applies-operator
 a*state*applied-newer a*wm*unpack-applied-om *x4* **chunk-962 chunk-946 chunk-922**
chunk-1141 p*display*dunk-comprehend a*display*t691 p*generic*indifferent
chunk-961 chunk-906 *x2* a*dp*unpack-applied-om *x4*
 a*dp*unpack-applied-om *x7*
chunk-962 chunk-1141 chunk-922

440 O: 02811 (display print-sp:applies-operator implement-exhausted (t691))
 ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
 a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
 a*wm*unpack-applied-om p*generic*reject ao*display*print*t691
 d*print*t691

441 O: 02797 (comprehend implement-exhausted :assertion)
 f:apply-sps f:sp f:sp-parts **chunk-922 chunk-946 chunk-962 chunk-1141**
 po*display*print-sp*applies-operator
 p*display*reject-duplicates p*generic*indifferent

442 ==>S: S81 (operator no-change)
 a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
 a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
 po*attend
 p*generic*indifferent

443 O: 02813 (attend t691 constant176)
 ao*attend chunk-1149 ao*attend*mark-locally p*generic*terminate-and-reject
 a*dp*unpack-applied-om x3 p*generic*reject
 a*wm*newest-from-not-newest
 ao*attend*previous-not-newest chunk-1150 po*probe*with-previous-goal po*probe*with-part x2
 po*fixate*current-context po*probe*with-high-level-goal po*fixate*binding-attribute*target
 po*fixate*selected-operator po*fixate*selected-id po*fixate*augmentation x8 po*fixate*assertion
 po*fixate*augmentation x3 po*probe*with-important-object x3
 a*dp*unpack-applied-om p*probe*lhs-best p*probe*rhs-better-when-apply-sp
 p*fixate*invariant-feature*dont-interleave-best p*probe*repeated-goal*worst x2 p*generic*indifferent
 a*fixate*dont-interleave-best
 a*fixate*newest
 p*fixate*newest*interleave-best

444 O: 02838 (comprehend exhausted :imp-obj operator*)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first f:high-level-goal-cues
 f:operator ao*substate*count-first
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer
 p*probe*new-high-level-goal*best

445 O: 02821 (comprehend return-new-pointer |(high-level-goal)|)
 ao*probe*goal p*generic*terminate-and-reject ao*substate*count-second ao*comprehend*applied*second
 p*probe*repeated-goal*worst p*generic*reject p*probe*rhs-when-sp a*state*applied-newer

446 O: 02818 (comprehend rhs :part)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:high-level-goal-cues
 po*fixate*action x2 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 p*probe*new-high-level-goal*best ao*probe*unpack-probe-om-to-superop-om chunk-1151
 a*fixate*newest x2 a*wm*unpack-applied-om p*generic*indifferent x2
 p*fixate*newest*interleave-best x2 po*probe*with-high-level-goal
 p*fixate*interleave-best p*fixate*bottom-up p*fixate*interleave-best p*probe*repeated-goal*worst
 p*probe*new-high-level-goal*best p*generic*indifferent

447 O: 02816 (comprehend lhs :part)
 p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 f:operator-condition*lhs-means-rhs
 po*fixate*condition x7 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x3
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x3 p*generic*indifferent x7
 a*fixate*dont-interleave-best p*fixate*newest*interleave-best a*fixate*dont-interleave-best
 p*fixate*newest*interleave-best x3 a*fixate*dont-interleave-best p*fixate*newest*interleave-best
 x3
 p*fixate*interleave-best p*fixate*bottom-up p*fixate*interleave-best x2 p*fixate*bottom-up
 p*fixate*interleave-best p*fixate*bottom-up

448 O: 02845 (fixate :action superstate construction-done)
 ao*fixate chunk-1152 ao*fixate chunk-1153 ao*fixate chunk-1154
 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1155 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1156 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1157 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x3
 po*probe*with-important-object po*comprehend*superstate po*fixate*superstate-id
 po*probe*with-important-object x2
 p*probe*new-important-object*best ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 a*fixate*newest p*probe*new-important-object*best p*probe*new-attribute*best

p*probe*new-important-object*best p*generic*indifferent x5
 ao*goal-select*new-goal*fixate/imagine chunk-1158

449 O: 02857 (comprehend superstate :superstate)
 ao*comprehend*applied ao*comprehend*create-dp-on-om
 a*state*applied-newer **chunk-1141 chunk-962 chunk-946 chunk-922 chunk-1150**
chunk-1140 chunk-945 chunk-921 a*dp*unpack-applied-om x5
 a*dp*unpack-applied-om x9
chunk-1141 chunk-1150 chunk-962 chunk-946 chunk-922

450 ==>S: S82 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects po*imagine*assertions a*substate*create-time
 a*fixate*newest x4 po*probe*with-previous-goal po*probe*with-part x2 po*fixate*current-context
 po*fixate*binding-attribute*target po*fixate*selected-operator po*fixate*selected-id
 po*fixate*superstate-id po*fixate*assertion po*fixate*augmentation po*fixate*assertion p*imagine*refract
 po*probe*with-important-object x4 p*attend*old-regions*reject p*generic*indifferent x5
 p*probe*lhs-best a*fixate*newest p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x3
 p*fixate*superstate a*fixate*newest p*probe*repeated-goal*worst p*generic*indifferent
 a*fixate*dont-interleave-best p*fixate*newest*interleave-best

451 O: 02879 (fixate superstate s15 :superstate-id)
 ao*fixate chunk-1159 ao*fixate chunk-1160 p*fixate*interleave-best x4
 p*generic*terminate-and-reject ao*substate*count-first p*probe*new-attribute*best
 a*wm*unpack-applied-om x2 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1161 **chunk-506** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1162 **chunk-507** ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x2
 po*display*run*action-and-print po*fixate*shared-state po*probe*with-important-object
 a*display*t720 p*display*dunk-comprehend a*fixate*newest p*probe*new-important-object*best
 p*generic*indifferent x3

452 O: 02901 (display run:action-and-print (t720))
 ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
 a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
 a*wm*unpack-applied-om p*generic*reject ao*display*print*t720
 d*print*t720

453 O: 02857 (comprehend superstate :superstate)
 ao*comprehend*cleanup-naked-region-pointers x9 **chunk-946 chunk-962 chunk-1141**
chunk-1150 po*display*run*action-and-print
 p*display*reject-duplicates p*generic*indifferent

454 ==>S: S83 (operator no-change)
 a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend x2 a*state*fixate-meta-attributes
 a*substate*initialize-goal-set a*state*important-objects po*imagine*assertions a*substate*create-time
 po*attend x2 p*generic*indifferent x4
 p*generic*indifferent x2

455 O: 02912 (attend t720 constant179)
 ao*attend chunk-1163 **chunk-1163** ao*attend*mark-locally p*generic*terminate-and-reject
 a*dp*unpack-applied-om x4 p*generic*reject
 a*wm*newest-from-not-newest
 ao*attend*previous-not-newest chunk-1164 **chunk-1164** ao*attend*previous-not-newest
 po*fixate*builds po*probe*with-previous-goal po*probe*with-part x2 po*fixate*annotations
 ao*fixated-recently chunk-1166 ao*fixated-recently chunk-1167
 po*fixate*binding-attribute*target po*fixate*augmentation po*fixate*assertion po*fixate*augmentation
 x7 p*imagine*worst-after-new-output x4 p*imagine*refract p*imagine*worst-after-new-output x4
 po*probe*with-important-object x5

a*dp*unpack-applied-om p*probe*lhs-best a*wm*unpack-applied-om
 p*fixate*invariant-feature*dont-interleave-best p*probe*repeated-goal*worst p*generic*indifferent
 p*probe*fixated-recently*worst x2 po*probe*with-important-object a*fixate*dont-interleave-best
 a*fixate*newest p*probe*fixated-recently*worst p*generic*indifferent
 p*fixate*newest*interleave-best x9
 456 O: 02920 (fixate :annotations)
 ao*fixate chunk-1168 p*generic*terminate-and-reject ao*substate*count-first
 a*wm*unpack-applied-om p*generic*reject p*fixate*fixated-recently-in-view*best x7
 ao*fixate*unpack-fixation-object chunk-1169 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 f:propose-return-operator
 a*wm*unpack-applied-om x2
 po*display*match-set*asserted-sp po*probe*with-important-object
 p*display*dunk-comprehend a*display*t730 p*generic*indifferent x2
 457 O: 02953 (display match-set:asserted-sp (t730))
 ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
 p*display*reject-duplicates a*wm*unpack-applied-om p*generic*reject ao*display*print*t730
 d*print*t730
 458 O: 02857 (comprehend superstate :superstate)
 f:propose-return-operator **chunk-962 chunk-1141 chunk-1150 chunk-1164** x2
 po*display*run*action-and-print **chunk-1166 chunk-1167**
 p*display*reject-duplicates p*generic*indifferent
 459 ==>S: S85 (operator no-change)
 a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
 a*substate*initialize-goal-set a*state*important-objects po*imagine*assertions a*substate*create-time
 po*attend p*generic*indifferent x4
 p*generic*indifferent
 460 O: 02961 (attend t730 constant180)
 ao*attend chunk-1170 ao*attend*mark-locally p*generic*terminate-and-reject
 a*dp*unpack-applied-om x3 p*generic*reject
 a*wm*newest-from-not-newest
 ao*attend*previous-not-newest chunk-1171 po*fixate*builds po*probe*with-previous-goal
 po*probe*with-part x2 ao*fixated-recently x2 po*fixate*binding-attribute*target
 po*fixate*augmentation po*fixate*assertion po*fixate*augmentation x7 po*fixate*assertion x2
 p*imagine*worst-after-new-output x8 p*imagine*refract p*imagine*worst-after-new-output x8
 po*probe*with-important-object x7
 a*dp*unpack-applied-om p*probe*lhs-best p*fixate*invariant-feature*dont-interleave-best
 p*probe*fixated-recently*worst x4 p*probe*repeated-goal*worst p*generic*indifferent
 a*fixate*dont-interleave-best
 a*fixate*newest
 p*fixate*newest*interleave-best x2
 p*fixate*top-down
 461 O: 02965 (comprehend lhs :part)
 p*fixate*interleave-best x6 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs **chunk-1166 chunk-1167** ao*substate*count-first
 po*fixate*condition x7 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 a*state*new-important-object*best p*fixate*fixated-recently-in-view*best x7
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x3
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x3 p*probe*retrieved-by-probe*best

x2 p*generic*indifferent x7
a*fixate*dont-interleave-best x3

462 O: 02989 (fixate propose-return-operator :assertion)
a*fixate chunk-1174 p*generic*terminate-and-reject ao*substate*count-second
a*wm*unpack-applied-om p*generic*reject
a*fixate*unpack-fixation-object chunk-1175 ao*comprehend*unpack-fixation-object
a*wm*unpack-applied-om
po*comprehend*assertion*real po*probe*with-important-object
a*topstate*clean-up-old-comprehends-and-displays ao*goal-select*comprehend*create-token
a*goal-select*proposed-during p*probe*new-important-object*best p*generic*indifferent x2
ao*goal-select*new-goal*fixate/imagine chunk-1176

463 O: 03012 (comprehend propose-return-operator :assertion)
a*comprehend*applied f:high-level-goal-cues f:sp f:sp-parts ao*comprehend*create-dp-on-om
po*comprehend*proposal-context **chunk-1166 chunk-1167**
a*state*applied-newer a*wm*unpack-applied-om x6 **chunk-1164 x2 chunk-1150 chunk-1141**
chunk-962 chunk-1171 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
p*generic*indifferent
chunk-1149 chunk-961 po*comprehend*high-level-goal a*dp*unpack-applied-om x5
ao*goal-select*select-now
a*dp*unpack-applied-om x6 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
p*generic*indifferent
chunk-1164 x2 chunk-1150 chunk-1141 chunk-962

464 O: 03016 (comprehend proposal-context :sno)
a*comprehend*applied ao*comprehend*create-dp-on-om po*display*print-stack **chunk-1166 chunk-1167**
a*state*applied-newer a*wm*unpack-applied-om **chunk-1171 chunk-962 chunk-1141 chunk-1150**
chunk-1164 x2 p*display*dunk-comprehend a*display*t738 p*generic*indifferent
chunk-1170 chunk-1163 x2 chunk-1140 chunk-945 a*dp*unpack-applied-om x5
a*dp*unpack-applied-om
chunk-1171 chunk-1164 x2 chunk-1141 chunk-1150 chunk-962
a*wm*newest-from-not-newest

465 O: 03020 (display print-stack (t738))
a*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
a*wm*unpack-applied-om p*generic*reject ao*display*print*t738
d*print*t738 p*generic*indifferent

466 O: 03016 (comprehend proposal-context :sno)
po*display*print-stack **chunk-962 chunk-1141 chunk-1150 chunk-1164 x2 chunk-1171**
p*display*reject-duplicates p*generic*indifferent

467 ==>S: S87 (operator no-change)
a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
po*attend
p*generic*indifferent

468 O: 03022 (attend t738 constant184)
ao*attend chunk-1177 ao*attend*mark-locally p*generic*terminate-and-reject
a*dp*unpack-applied-om x3 p*generic*reject
a*wm*newest-from-not-newest
ao*attend*previous-not-newest chunk-1178 po*fixate*builds po*probe*with-previous-goal
po*probe*with-part x2 po*fixate*current-context po*probe*with-high-level-goal
po*fixate*binding-attribute*target po*fixate*augmentation po*fixate*assertion po*fixate*augmentation
x7 po*fixate*assertion x2 po*probe*with-important-object x2 po*fixate*proposal-context

a*dp*unpack-applied-om p*probe*lhs-best p*fixate*invariant-feature*dont-interleave-best
 p*probe*fixated-recently*worst x2 p*generic*indifferent
 a*fixate*dont-interleave-best
 a*fixate*newest
 p*fixate*newest*interleave-best x2

469 O: 03059 (fixate :proposal-context create-operator)
 ao*fixate chunk-1179 p*generic*terminate-and-reject ao*substate*count-first
 a*wm*unpack-applied-om p*generic*reject p*fixate*fixated-recently-in-view*best x7
 ao*fixate*unpack-fixation-object chunk-1180 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om

470 O: 03052 (fixate receivers2 r8 :augmentation)
 ao*fixate chunk-1181 ao*fixate chunk-1182 ao*fixate chunk-1183 ao*fixate
chunk-1184 ao*fixate chunk-1185 ao*fixate chunk-1186 p*fixate*interleave-best
 p*generic*terminate-and-reject ao*substate*count-second p*probe*new-attribute*best
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1187 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1188 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1189 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1190 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1191 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1192 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x6
 po*probe*with-attribute
 p*probe*new-attribute*best p*generic*indifferent

471 O: 03051 (fixate assigners2 a65 :augmentation)
 ao*fixate chunk-1193 ao*fixate chunk-1194 ao*fixate chunk-1195 ao*fixate
chunk-1196 ao*fixate chunk-1197 ao*fixate chunk-1198 p*fixate*interleave-best
 p*generic*terminate-and-reject p*probe*new-attribute*best
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1199 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1200 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1201 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1202 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1203 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x5
 po*probe*with-attribute
 p*probe*new-attribute*best p*generic*indifferent

472 O: 03050 (fixate receivers r7 :augmentation)
 ao*fixate chunk-1204 ao*fixate chunk-1205 ao*fixate chunk-1206 ao*fixate
chunk-1207 ao*fixate chunk-1208 ao*fixate chunk-1209 p*fixate*interleave-best
 p*generic*terminate-and-reject p*probe*new-attribute*best
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1210 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1211 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1212 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1213 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1214 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x5
 po*probe*with-attribute
 p*probe*new-attribute*best p*generic*indifferent

473 O: 03049 (fixate assigners a64 :augmentation)
 ao*fixate chunk-1215 ao*fixate chunk-1216 ao*fixate chunk-1217 ao*fixate
chunk-1218 ao*fixate chunk-1219 ao*fixate chunk-1220 p*fixate*interleave-best

p*generic*terminate-and-reject p*probe*new-attribute*best
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1221 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1222 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1223 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1224 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1225 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x5
 po*probe*with-attribute
 p*probe*new-attribute*best p*generic*indifferent
 474 O: 03048 (fixate for-formatting f4 :augmentation)
 ao*fixate chunk-1226 ao*fixate chunk-1227 ao*fixate chunk-1228 ao*fixate
chunk-1229 ao*fixate chunk-1230 ao*fixate chunk-1231 p*fixate*interleave-best
 p*generic*terminate-and-reject p*probe*new-attribute*best
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1232 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1233 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1234 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1235 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1236 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x5
 po*probe*with-attribute
 p*probe*new-attribute*best p*generic*indifferent
 475 O: 03047 (fixate adjacency-info a3 :augmentation)
 ao*fixate chunk-1237 ao*fixate chunk-1238 ao*fixate chunk-1239 ao*fixate
chunk-1240 ao*fixate chunk-1241 ao*fixate chunk-1242 p*fixate*interleave-best
 p*generic*terminate-and-reject p*probe*new-attribute*best
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1243 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1244 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1245 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1246 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1247 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x5
 po*probe*with-attribute
 p*probe*new-attribute*best p*generic*indifferent
 476 O: 03046 (fixate ordering-info o3 :augmentation)
 ao*fixate chunk-1248 ao*fixate chunk-1249 ao*fixate chunk-1250 ao*fixate
chunk-1251 ao*fixate chunk-1252 ao*fixate chunk-1253 p*fixate*interleave-best
 p*generic*terminate-and-reject p*probe*new-attribute*best
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1254 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1255 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1256 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1257 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1258 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x5
 po*probe*with-attribute
 p*probe*new-attribute*best p*generic*indifferent
 477 O: 03030 (fixate :current-context s-construct)
 ao*fixate chunk-1259 p*generic*terminate-and-reject
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1260 **chunk-274** **chunk-283** **chunk-542**
chunk-925 ao*comprehend*unpack-fixation-object

a*wm*unpack-applied-om
 po*comprehend*actual-context po*fixate*state-id po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 p*fixate*fixated-recently-in-view*best a*fixate*newest p*probe*new-important-object*best
 p*generic*indifferent x3
 ao*goal-select*new-goal*fixate/imagine chunk-1261 p*fixate*newest*interleave-best

478 O: 03074 (comprehend s-construct :actual-context)
 ao*comprehend*applied f:s-construct-builds-chunk f:proposal-build-action ao*comprehend*create-dp-on-om
 a*state*applied-newer a*wm*unpack-applied-om x3 **chunk-1171 chunk-1164 x2 chunk-1150**
chunk-1141 chunk-962 chunk-1178
chunk-1149 chunk-961 chunk-276 a*dp*unpack-applied-om x6
 a*dp*unpack-applied-om x6 a*wm*unpack-applied-om
chunk-1164 x2 chunk-1150 chunk-1141 chunk-962 chunk-277 chunk-278
 a*wm*unpack-applied-om x2
 po*comprehend*sp-parts x2
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent x2

479 ==>S: S88 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*fixate*builds po*probe*with-previous-goal po*probe*with-part x2 po*fixate*state-id
 po*probe*with-attribute x7 po*fixate*binding-attribute*target po*fixate*augmentation x8
 po*fixate*assertion x3 po*fixate*augmentation x3 po*probe*with-important-object x5
 p*attend*old-regions*reject p*generic*indifferent
 a*fixate*newest p*probe*lhs-best a*fixate*newest p*fixate*invariant-feature*dont-interleave-best
 a*fixate*newest p*probe*fixated-recently*worst p*probe*repeated-goal*worst p*generic*indifferent
 p*fixate*newest*interleave-best a*fixate*dont-interleave-best

480 O: 03087 (comprehend lhs :part)
 p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 po*fixate*condition x7 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 p*fixate*fixated-recently-in-view*best
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x3
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x3 p*generic*indifferent x7
 a*fixate*dont-interleave-best x3

481 O: 03091 (fixate state s15 :state-id)
 ao*fixate chunk-1262 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1263 **chunk-280 chunk-306 chunk-928**
 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*probe*with-important-object
 p*probe*new-important-object*best p*probe*fixated-recently*worst p*generic*indifferent

482 O: 03138 (comprehend s15 :imp-obj state)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*second
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer

483 O: 03129 (comprehend apply-chunks :imp-obj sp)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:abstract-sp f:sp-parts
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer

484 O: 03125 (comprehend proposal-build :imp-obj action)

ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:high-level-goal-cues
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om
chunk-1264 ao*probe*unpack-probe-om-to-superop-om chunk-1265
 a*wm*unpack-applied-om x2
 po*comprehend*high-level-goal po*probe*with-high-level-goal
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-high-level-goal*best
 p*generic*indifferent x2
 485 O: 03142 (comprehend return-new-pointer |(high-level-goal)|)
 ao*goal-select*new-goal*probe chunk-1266 ao*probe*goal p*generic*terminate-and-reject
 ao*comprehend*applied f:pay-attention-when-building-proposal
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om
chunk-1267 ao*probe*unpack-probe-om-to-superop-om chunk-1268
 a*wm*unpack-applied-om x2
 p*fixate*assertion*pay-attention x3 po*comprehend*high-level-goal po*probe*with-high-level-goal
 p*fixate*interleave-best p*fixate*top-down p*fixate*interleave-best x2
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-high-level-goal*best
 p*generic*indifferent x2
 486 O: 03140 (comprehend return-new-pointer :high-lev-goal)
 ao*comprehend*applied f:pay-attention-when-building-proposal ao*comprehend*create-dp-om-on
 a*state*applied-newer a*wm*unpack-applied-om x2 **chunk-1171 chunk-1178 chunk-962**
chunk-1141 chunk-1150 chunk-1164 x2
chunk-1177 chunk-1170 chunk-1163 x2 **chunk-1140 chunk-945** a*dp*unpack-applied-om
 x6
 a*dp*unpack-applied-om
chunk-1178 chunk-1171 chunk-1164 x2 **chunk-1141 chunk-1150 chunk-962**
 a*wm*newest-from-not-newest
 487 ==>S: S89 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*fixate*builds po*probe*with-previous-goal po*probe*with-part x2 po*fixate*current-context
 po*probe*with-high-level-goal x2 po*fixate*binding-attribute*target po*fixate*augmentation
 po*fixate*assertion po*fixate*augmentation x7 po*fixate*assertion x2
 po*probe*with-important-object x3 p*attend*old-regions*reject p*generic*indifferent
 a*fixate*newest p*probe*lhs-best a*fixate*newest p*probe*repeated-goal*worst
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest p*fixate*assertion*pay-attention
 a*fixate*newest x8 p*fixate*assertion*pay-attention a*fixate*newest p*fixate*assertion*pay-attention
 a*fixate*newest p*generic*indifferent
 p*fixate*newest*interleave-best a*fixate*dont-interleave-best p*fixate*top-down
 488 O: 03152 (comprehend lhs :part)
 p*fixate*interleave-best x4 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 po*fixate*condition x7 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x3
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x3 p*generic*indifferent x7
 a*fixate*dont-interleave-best x3
 489 O: 03181 (fixate propose-return-operator :assertion)
 ao*fixate chunk-1269 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1270 **chunk-1175** ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*probe*with-important-object
 p*probe*new-important-object*best p*generic*indifferent

- 490 O: 03196 (comprehend propose-return-operator :imp-obj assertion)
 p*fixate*interleave-best x3 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*second
 f:high-level-goal-cues f:sp
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer p*probe*new-high-level-goal*best
 ao*probe*unpack-probe-om-to-superop-om chunk-1271 ao*probe*unpack-probe-om-to-superop-om
chunk-1272
 a*wm*unpack-applied-om x2
 po*probe*with-high-level-goal f:sp-parts po*probe*with-important-object
 p*probe*repeated-goal*worst p*probe*new-high-level-goal*best p*probe*repeated-goal*worst
 p*generic*indifferent x2
- 491 O: 03182 (fixate terminate-s-model-constructor :assertion)
 ao*fixate chunk-1273 p*generic*terminate-and-reject
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1274 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*comprehend*assertion*pay-attention po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*generic*indifferent x2
 ao*goal-select*new-goal*fixate/imagine chunk-1275
- 492 O: 03202 (comprehend terminate-s-model-constructor :assertion-pay-attn)
 ao*comprehend*applied f:sp f:terminating-sps ao*comprehend*create-dp-on-om
 po*display*print-sp*when-paying-attention
 a*state*applied-newer a*wm*unpack-applied-om x2 **chunk-962 chunk-1150 chunk-1141**
chunk-1164 x2 **chunk-1171 chunk-1178** p*display*dunk-comprehend a*display*t754
 p*generic*indifferent
chunk-1149 chunk-961 f:sp-parts a*dp*unpack-applied-om x6
 a*dp*unpack-applied-om x6 a*wm*unpack-applied-om x2
chunk-1150 chunk-1164 x2 **chunk-962 chunk-1141**
- 493 O: 03206 (display print-sp:paying-attention terminate-s-model-constructor (t754))
 ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
 a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
 a*wm*unpack-applied-om p*generic*reject ao*display*print*t754
 d*print*t754
- 494 O: 03202 (comprehend terminate-s-model-constructor :assertion-pay-attn)
 f:sp f:terminating-sps f:sp-parts ao*comprehend*cleanup-naked-region-pointers **chunk-1178**
 po*display*print-sp*when-paying-attention
 p*display*reject-duplicates p*generic*indifferent
- 495 ==>S: S92 (operator no-change)
 a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend x2 a*state*fixate-meta-attributes
 a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
 po*attend
 p*generic*indifferent
- 496 O: 03208 (attend t754 constant193)
 ao*attend chunk-1276 ao*attend*mark-locally p*generic*terminate-and-reject
 a*dp*unpack-applied-om x3 p*generic*reject
 a*wm*newest-from-not-newest
 ao*attend*previous-not-newest chunk-1277 po*probe*with-previous-goal po*probe*with-part x2
 po*fixate*current-context ao*fixated-recently chunk-1278 ao*fixated-recently chunk-1279
 po*probe*with-high-level-goal po*fixate*binding-attribute*target po*probe*with-important-object x4
 a*dp*unpack-applied-om p*probe*lhs-best a*wm*unpack-applied-om x2 p*probe*repeated-goal*worst x2
 p*generic*indifferent

po*comprehend*assertion*pay-attention*fixated-recently p*probe*fixated-recently*worst x4
 po*probe*with-important-object po*comprehend*assertion*pay-attention*fixated-recently
 p*probe*fixated-recently*worst x4 po*probe*with-important-object
 a*fixate*newest x2 ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 p*probe*fixated-recently*worst ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 p*probe*repeated-goal*worst p*probe*fixated-recently*worst p*generic*indifferent x4
 p*fixate*newest*interleave-best
 497 O: 03213 (comprehend lhs :part)
 p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first **chunk-1278 chunk-1279**
 po*fixate*condition x4 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 a*state*new-important-object*best x2
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x4 p*probe*retrieved-by-probe*best
 x6 p*generic*indifferent x4
 a*fixate*dont-interleave-best p*fixate*newest*interleave-best x4
 p*fixate*interleave-best p*fixate*top-down p*fixate*interleave-best p*fixate*top-down
 p*fixate*interleave-best
 498 O: 03218 (fixate bound-by type :bind-target)
 ao*fixate chunk-1280 ao*fixate chunk-1281 ao*fixate chunk-1282
 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om x3 p*generic*reject p*probe*rhs-when-sp
 ao*fixate*unpack-fixation-object chunk-1283 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1284 **chunk-489 chunk-624**
 ao*comprehend*unpack-fixation-object ao*fixate*unpack-fixation-object chunk-1285
 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x3
 po*probe*with-important-object
 p*probe*new-important-object*best p*generic*indifferent
 499 O: 03211 (comprehend rhs :part)
 p*fixate*interleave-best x3 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*second
 f:high-level-goal-cues **chunk-1278 chunk-1279**
 p*fixate*bottom-up x2 po*fixate*action p*probe*repeated-goal*worst p*generic*reject
 a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om chunk-1286
 ao*probe*unpack-probe-om-to-superop-om chunk-1287 a*state*new-important-object*best x2
 a*fixate*newest a*wm*unpack-applied-om x2 p*probe*retrieved-by-probe*best x6
 p*generic*indifferent
 p*fixate*newest*interleave-best po*comprehend*high-level-goal po*probe*with-high-level-goal x2
 p*fixate*interleave-best p*fixate*bottom-up ao*goal-select*comprehend*create-token
 a*goal-select*proposed-during p*probe*new-high-level-goal*best x2 p*generic*indifferent x3
 500 O: 03241 (fixate :action s-model-constructor reconsider)
 ao*fixate chunk-1288 ao*fixate chunk-1289 ao*fixate chunk-1290
 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1291 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1292 **chunk-630 chunk-756**
 ao*comprehend*unpack-fixation-object ao*fixate*unpack-fixation-object chunk-1293
 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x3
 po*probe*with-important-object x2
 p*probe*new-important-object*best p*probe*new-attribute*best p*probe*new-important-object*best
 p*generic*indifferent x2
 501 O: 03250 (comprehend reconsider :imp-obj action)
 p*fixate*interleave-best x3 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
chunk-1278 chunk-1279 po*imagine*action

p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer a*state*new-important-object*best
x2 p*imagine*refract p*imagine*worst-after-new-output a*fixate*newest p*generic*indifferent
p*probe*retrieved-by-probe*best x6

502 O: 03238 (fixate :condition problem-space create-operator)
ao*fixate chunk-1294 ao*fixate chunk-1295 ao*fixate chunk-1296
p*generic*terminate-and-reject
a*wm*unpack-applied-om x3 p*generic*reject
ao*fixate*unpack-fixation-object chunk-1297 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-1298 **chunk-73 chunk-290 chunk-438 chunk-483**
chunk-568 chunk-806 ao*comprehend*unpack-fixation-object ao*fixate*unpack-fixation-object
chunk-1299 ao*comprehend*unpack-fixation-object
a*wm*unpack-applied-om x3
po*probe*with-important-object x2
p*probe*new-important-object*best p*probe*new-attribute*best p*probe*new-important-object*best
p*generic*indifferent x2

503 O: 03255 (comprehend create-operator :imp-obj condition)
p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
chunk-1278 chunk-1279 po*imagine*action
p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer a*state*new-important-object*best
x2 p*imagine*refract p*imagine*worst-after-new-output a*fixate*newest p*generic*indifferent
p*probe*retrieved-by-probe*best x6

504 O: 03237 (fixate :condition type s-model-constructor)
ao*fixate chunk-1300 ao*fixate chunk-1301 ao*fixate chunk-1302
p*generic*terminate-and-reject p*probe*new-attribute*best p*probe*new-important-object*best
a*wm*unpack-applied-om x3 p*generic*reject
ao*fixate*unpack-fixation-object chunk-1303 **chunk-488 chunk-623**
ao*comprehend*unpack-fixation-object ao*fixate*unpack-fixation-object chunk-1304 **chunk-490**
chunk-625 ao*comprehend*unpack-fixation-object
a*wm*unpack-applied-om x2
po*probe*with-important-object
p*probe*new-important-object*best p*generic*indifferent

505 O: 03239 (comprehend type :imp-obj condition)
p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
chunk-1278 chunk-1279 po*imagine*action
p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer a*state*new-important-object*best
x2 p*imagine*refract p*imagine*worst-after-new-output a*fixate*newest p*generic*indifferent
p*probe*retrieved-by-probe*best x6

506 O: 03236 (fixate :condition annotation construction-done)
ao*fixate chunk-1305 ao*fixate chunk-1306 ao*fixate chunk-1307
p*generic*terminate-and-reject
a*wm*unpack-applied-om x3 p*generic*reject
ao*fixate*unpack-fixation-object chunk-1308 **chunk-1169** ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-1309 ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-1310 ao*comprehend*unpack-fixation-object
a*wm*unpack-applied-om x3
po*probe*with-important-object x2
p*probe*new-important-object*best p*probe*new-attribute*best p*probe*new-important-object*best
p*generic*indifferent x2

507 O: 03263 (comprehend construction-done :imp-obj condition)
ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied **chunk-1278 chunk-1279**
po*imagine*action
p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer a*state*new-important-object*best

x2 p*imagine*refract p*imagine*worst-after-new-output a*fixate*newest p*generic*indifferent
p*probe*retrieved-by-probe*best x6

508 O: 03225 (comprehend terminate-s-model-constructor :imp-obj sp)
ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:sp f:terminating-sps f:sp-parts
chunk-1288 chunk-1289 chunk-1290 chunk-1294 chunk-1295 chunk-1296
chunk-1300 chunk-1301 chunk-1302 chunk-1305 chunk-1306 chunk-1307
chunk-1278 chunk-1279 po*imagine*action
p*generic*reject a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om*fixated **chunk-1311**
chunk-1311 ao*probe*unpack-probe-om-to-superop-om*fixated x3 **chunk-1314 chunk-1314**
ao*probe*unpack-probe-om-to-superop-om*fixated x3 **chunk-1317 chunk-1317**
ao*probe*unpack-probe-om-to-superop-om*fixated x3 **chunk-1320 chunk-1320**
ao*probe*unpack-probe-om-to-superop-om*fixated x2 a*state*new-important-object*best p*imagine*refract
p*imagine*worst-after-new-output a*fixate*newest p*generic*indifferent
ao*probe*unpack-probe-om-to-superop-om a*wm*unpack-applied-om p*probe*retrieved-by-probe*best x3

509 O: 03223 (comprehend propose-return-operator :imp-obj sp)
ao*goal-select*new-goal*probe **chunk-1324** ao*probe*goal p*generic*terminate-and-reject
ao*comprehend*applied f:high-level-goal-cues f:sp f:sp-parts **chunk-1278 chunk-1279**
po*imagine*action
p*probe*repeated-goal*worst x3 p*generic*reject a*state*applied-newer
p*probe*new-high-level-goal*best x2 a*state*new-important-object*best p*imagine*refract
p*imagine*worst-after-new-output a*fixate*newest p*generic*indifferent
p*probe*retrieved-by-probe*best x3

510 O: 03227 (comprehend propose-return-operator :assertion-fix-recent)
ao*comprehend*applied f:high-level-goal-cues f:sp f:sp-parts ao*comprehend*create-dp-on-om
po*display*print*high-level-goal po*display*print-sp*when-paying-attention **chunk-1278 chunk-1279**
a*state*applied-newer a*wm*unpack-applied-om x7 **chunk-1277** p*display*dunk-comprehend
a*display*t770 p*display*reject-duplicates a*display*t770 p*generic*indifferent x2
po*display*scroll*to-sp **chunk-1177** p*display*dunk-comprehend po*comprehend*sp-parts x2
po*comprehend*proposal-context a*dp*unpack-applied-om
p*display*dunk-comprehend a*display*t770 a*dp*unpack-applied-om x3
ao*goal-select*comprehend*create-token a*goal-select*proposed-during
ao*goal-select*comprehend*create-token a*goal-select*proposed-during
ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent x4
chunk-1277

511 O: 03270 (display scroll:to-sp propose-return-operator (t770))
ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
a*wm*unpack-applied-om p*generic*reject ao*display*print*t770
d*print*t730 p*generic*indifferent

512 O: 03227 (comprehend propose-return-operator :assertion-fix-recent)
chunk-1170 f:high-level-goal-cues f:sp f:sp-parts po*display*scroll*to-sp **chunk-1277**
po*comprehend*proposal-context
a*dp*unpack-applied-om x3 p*display*reject-duplicates a*display*t774
ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent x2
chunk-1178
a*dp*unpack-applied-om

513 ==>S: S94 (operator no-change)
a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
a*state*important-objects a*substate*create-time
po*probe*with-previous-goal po*probe*with-part x2 po*fixate*current-context
po*probe*with-high-level-goal x2 po*fixate*assertion x2 po*probe*with-important-object
p*generic*indifferent
p*probe*fixated-recently*worst p*probe*lhs-best a*fixate*newest x3 p*probe*fixated-recently*worst
p*probe*repeated-goal*worst p*probe*fixated-recently*worst p*probe*repeated-goal*worst
p*probe*fixated-recently*worst x2 p*generic*indifferent

514 O: 03280 (attend not-newest constant193)
 ao*attend*old-regions chunk-1325 p*generic*terminate-and-reject
 a*dp*unpack-applied-om p*generic*reject
 po*probe*where-was-i p*probe*fixated-recently*best x5
 p*probe*where-was-i*best p*generic*indifferent

515 O: 03316 (comprehend terminate-s-model-constructor :imp-obj sp)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first f:sp f:terminating-sps
 f:sp-parts **chunk-1288 chunk-1289 chunk-1290 chunk-1294 chunk-1295 chunk-1296**
chunk-1300 chunk-1301 chunk-1302 chunk-1305 chunk-1306 chunk-1307
 ao*substate*count-first
 p*probe*repeated-goal*worst x3 p*generic*reject a*state*applied-newer
 p*fixate*fixated-recently-in-view*best x2

516 O: 03293 (fixate terminate-s-model-constructor :assertion)
 ao*fixate chunk-1326 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om p*generic*reject p*probe*rhs-when-sp
 ao*fixate*unpack-fixation-object chunk-1327 **chunk-1274** ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om
 po*comprehend*assertion*real po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*probe*repeated-goal*worst p*probe*fixated-recently*best p*generic*indifferent x2

517 O: 03292 (fixate propose-return-operator :assertion)
 ao*fixate chunk-1328 p*generic*terminate-and-reject
 a*wm*unpack-applied-om p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1329 **chunk-1175 chunk-1270**
 po*display*print*high-level-goal ao*comprehend*unpack-fixation-object
 p*display*dunk-comprehend a*display*t774 a*wm*unpack-applied-om p*generic*indifferent
 po*comprehend*assertion*real po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*probe*repeated-goal*worst p*probe*fixated-recently*best p*generic*indifferent x2

518 O: 03324 (display print:high-level-goal propose-return-operator (t774))
 ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
 a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
 a*wm*unpack-applied-om p*generic*reject ao*display*print*t774
 d*print*t774

519 O: 03227 (comprehend propose-return-operator :assertion-fix-recent)
 f:high-level-goal-cues f:sp f:sp-parts **chunk-1178 chunk-1277** po*display*print*high-level-goal
 po*comprehend*proposal-context
 p*display*reject-duplicates ao*goal-select*comprehend*create-token a*goal-select*proposed-during
 p*generic*indifferent x2

520 ==>S: S97 (operator no-change)
 a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
 a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
 po*attend
 p*generic*indifferent

521 O: 03332 (attend t774 constant205)
 ao*attend chunk-1330 ao*attend*mark-locally p*generic*terminate-and-reject
 a*dp*unpack-applied-om x3 p*generic*reject
 a*wm*newest-from-not-newest
 ao*attend*previous-not-newest chunk-1331 po*probe*with-previous-goal po*probe*with-part x2
 po*fixate*current-context po*probe*with-high-level-goal x2 po*probe*with-important-object
 p*probe*fixated-recently*worst p*probe*lhs-best a*fixate*newest p*probe*fixated-recently*worst

p*probe*repeated-goal*worst p*probe*fixated-recently*worst x2 p*probe*repeated-goal*worst
 p*probe*fixated-recently*worst p*probe*repeated-goal*worst p*probe*fixated-recently*worst x2
 p*generic*indifferent

522 O: 03337 (comprehend lhs :part)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 po*fixate*condition x4 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x2
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x2 p*generic*indifferent x4
 a*fixate*dont-interleave-best p*fixate*newest*interleave-best a*fixate*dont-interleave-best
 p*fixate*newest*interleave-best x2
 p*fixate*interleave-best x2

523 O: 03375 (fixate :condition type u-model-constructor)
 ao*fixate chunk-1332 ao*fixate chunk-1333 ao*fixate chunk-1334
 p*generic*terminate-and-reject ao*substate*count-second p*probe*new-attribute*best
 p*probe*new-important-object*best
 a*wm*unpack-applied-om x3 p*generic*reject p*probe*rhs-when-sp
 ao*fixate*unpack-fixation-object chunk-1335 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1336 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x2
 po*probe*with-important-object
 p*probe*new-important-object*best p*generic*indifferent

524 O: 03352 (comprehend type :imp-obj condition)
 p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*second
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer

525 O: 03335 (comprehend rhs :part)
 p*fixate*interleave-best ao*goal-select*new-goal*probe chunk-1337 ao*probe*goal
 p*generic*terminate-and-reject ao*comprehend*applied f:high-level-goal-cues
 po*fixate*action po*imagine*action p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 p*probe*new-high-level-goal*best x2
 a*fixate*newest p*imagine*refract p*imagine*worst-after-new-output x2 a*fixate*newest
 p*generic*indifferent x2
 p*fixate*newest*interleave-best
 p*fixate*interleave-best

526 O: 03271 (comprehend rhs :sp-parts)
 ao*comprehend*applied f:high-level-goal-cues ao*comprehend*create-dp-on-om
 a*state*applied-newer a*wm*unpack-applied-om x2 **chunk-1277 chunk-1178 chunk-1331**
chunk-489 chunk-624 chunk-1284 ao*comprehend*unpack-fixation-object **chunk-1276**
 a*dp*unpack-applied-om x3
 a*wm*unpack-applied-om a*dp*unpack-applied-om x3
chunk-1277 chunk-1331

527 ==>S: S98 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*probe*with-previous-goal po*probe*with-part x2 po*fixate*current-context
 po*fixate*binding-attribute*param po*probe*with-high-level-goal x2 po*fixate*binding-attribute*target
 po*probe*with-important-object x7 po*fixate*action x2 p*attend*old-regions*reject
 p*generic*indifferent
 p*probe*fixated-recently*worst p*probe*repeated-goal*worst p*probe*lhs-best a*fixate*newest x3
 p*probe*fixated-recently*worst x5 a*fixate*newest x2 p*generic*indifferent
 p*fixate*newest*interleave-best x2

528 O: 03385 (comprehend lhs :part)

p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 po*fixate*condition x9 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 a*fixate*newest p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x4
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x4 p*generic*indifferent x9
 p*fixate*newest*interleave-best a*fixate*dont-interleave-best p*fixate*newest*interleave-best x4
 a*fixate*dont-interleave-best
 p*fixate*interleave-best p*fixate*bottom-up p*fixate*interleave-best p*fixate*bottom-up
 p*fixate*interleave-best p*fixate*bottom-up p*fixate*interleave-best
 529 O: 03408 (fixate :action new-operator op)
 ao*fixate chunk-1338 ao*fixate chunk-1339 ao*fixate chunk-1340
 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1341 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1342 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1343 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x3
 po*probe*with-important-object x2
 p*probe*new-important-object*best p*probe*new-attribute*best p*probe*new-important-object*best
 p*generic*indifferent x2
 530 O: 03421 (comprehend op :imp-obj action)
 p*fixate*interleave-best x5 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*second
 po*imagine*action
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer p*imagine*refract a*fixate*newest
 p*generic*indifferent
 531 O: 03388 (fixate condition operator* op :bind-param)
 ao*fixate chunk-1344 ao*fixate chunk-1345 ao*fixate chunk-1346 ao*fixate
chunk-1347 ao*fixate chunk-1348 ao*fixate chunk-1349 p*generic*terminate-and-reject
 p*probe*new-attribute*best x2
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1350 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1351 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1352 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1353 **chunk-80 chunk-296 chunk-474**
 ao*comprehend*unpack-fixation-object ao*fixate*unpack-fixation-object chunk-1354
 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x5
 po*comprehend*operator-condition po*probe*with-important-object
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 p*generic*indifferent x2
 ao*goal-select*new-goal*fixate/imagine chunk-1355 ao*goal-select*new-goal*fixate/imagine x2
 532 O: 03424 (comprehend operator* :op-cond)
 ao*comprehend*applied f:operator-condition*part-of-lhs ao*comprehend*create-dp-on-om
 a*state*applied-newer a*wm*unpack-applied-om **chunk-1178 chunk-1331 chunk-1277**
chunk-1330 chunk-1170 chunk-1177 chunk-83 a*dp*unpack-applied-om x3
 a*dp*unpack-applied-om x9 a*wm*unpack-applied-om
chunk-1331 chunk-1178 chunk-1277 a*wm*newest-from-not-newest po*comprehend*sp-parts
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent
 533 ==>S: S99 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*probe*with-previous-goal po*probe*with-part x2 po*fixate*current-context
 po*probe*with-high-level-goal po*fixate*binding-attribute*target po*fixate*assertion x2

po*probe*with-important-object x4 p*attend*old-regions*reject p*generic*indifferent
 p*probe*lhs-best a*fixate*newest x4 p*probe*repeated-goal*worst p*generic*indifferent
 534 O: 03433 (comprehend lhs :part)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 po*fixate*condition p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x5
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x5 p*generic*indifferent
 a*fixate*dont-interleave-best p*fixate*newest*interleave-best a*fixate*dont-interleave-best
 p*fixate*newest*interleave-best x5
 p*fixate*interleave-best p*fixate*top-down p*fixate*interleave-best p*fixate*top-down
 p*fixate*interleave-best p*fixate*top-down x2 p*fixate*interleave-best x2
 535 O: 03455 (fixate :condition operator* op)
 ao*fixate chunk-1358 ao*fixate chunk-1359 ao*fixate chunk-1360
 p*generic*terminate-and-reject ao*substate*count-second p*probe*new-attribute*best
 p*probe*new-important-object*best
 a*wm*unpack-applied-om x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1361 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1362 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x2
 po*comprehend*new-operator po*probe*with-important-object x2
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*probe*new-important-object*best
 x2 p*generic*indifferent x3
 ao*goal-select*new-goal*fixate/imagine chunk-1363 ao*goal-select*new-goal*fixate/imagine
 536 O: 03461 (comprehend op :new-operator)
 ao*comprehend*applied ao*comprehend*create-dp-on-om po*display*run*to-expected-op
 a*state*applied-newer chunk-1277 chunk-1178 chunk-1331 p*display*dunk-comprehend
 a*display*t811 p*generic*indifferent
chunk-80 chunk-296 chunk-474 chunk-1353 ao*comprehend*unpack-fixation-object
chunk-1276 a*dp*unpack-applied-om x3
 a*wm*unpack-applied-om a*dp*unpack-applied-om x3
 po*comprehend*operator-condition chunk-1277 chunk-1331
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent
 537 ==>S: S100 (operator no-change)
 a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
 a*state*important-objects a*substate*create-time
 po*probe*with-previous-goal po*probe*with-part x2 po*fixate*current-context
 po*fixate*binding-attribute*param po*fixate*binding-attribute*target po*fixate*assertion x2
 po*probe*with-important-object x3 p*attend*old-regions*reject p*generic*indifferent
 p*probe*lhs-best a*fixate*newest x5 p*probe*repeated-goal*worst x2 p*generic*indifferent
 p*fixate*newest*interleave-best
 538 O: 03473 (comprehend lhs :part)
 p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 po*fixate*condition x9 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x5
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x4 p*generic*indifferent x9
 a*fixate*dont-interleave-best p*fixate*newest*interleave-best x5 a*fixate*dont-interleave-best
 p*fixate*interleave-best p*fixate*top-down p*fixate*interleave-best p*fixate*top-down x2
 p*fixate*interleave-best x2
 539 O: 03478 (fixate condition operator* op :bind-param)
 ao*fixate chunk-1365 ao*fixate chunk-1366 ao*fixate chunk-1367 ao*fixate

chunk-1368 ao*fixate chunk-1369 ao*fixate chunk-1370 p*generic*terminate-and-reject
 ao*substate*count-second p*probe*new-attribute*best x2 p*probe*new-important-object*best
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1371 ao*comprehend*unpack-fixation-object **chunk-1350**
 ao*fixate*unpack-fixation-object chunk-1372 ao*comprehend*unpack-fixation-object **chunk-1351**
 ao*fixate*unpack-fixation-object chunk-1373 ao*comprehend*unpack-fixation-object **chunk-1341**
 ao*fixate*unpack-fixation-object chunk-1374 ao*comprehend*unpack-fixation-object **chunk-1352**
 ao*fixate*unpack-fixation-object chunk-1375 ao*comprehend*unpack-fixation-object **chunk-1354**
 a*wm*unpack-applied-om x5
 po*display*run*to-expected-op po*comprehend*new-operator
 p*display*dunk-comprehend a*display*t811 ao*goal-select*comprehend*create-token
 a*goal-select*proposed-during p*generic*indifferent x2
 540 O: 03497 (display run:to-expected-op (t811))
 ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
 a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
 a*wm*unpack-applied-om p*generic*reject ao*display*print*t811
 d*print*t811 p*generic*indifferent
 541 O: 03461 (comprehend op :new-operator)
chunk-1178 chunk-1277 chunk-1331 po*display*run*to-expected-op
 p*display*reject-duplicates a*display*t817 p*generic*indifferent
 542 ==>S: S101 (operator no-change)
 a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
 a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
 po*attend
 p*generic*indifferent
 543 O: 03502 (attend t811 constant214)
 ao*attend chunk-1376 ao*attend*mark-locally p*generic*terminate-and-reject
 a*dp*unpack-applied-om x3 p*generic*reject
 a*wm*newest-from-not-newest
 ao*attend*previous-not-newest chunk-1377 po*probe*with-previous-goal po*probe*with-part x2
 po*fixate*current-context po*fixate*binding-attribute*target po*fixate*selected-operator
 po*fixate*selected-id po*fixate*assertion x2 po*probe*with-important-object x3
 a*dp*unpack-applied-om p*probe*lhs-best p*probe*repeated-goal*worst x2 p*generic*indifferent
 a*fixate*newest x6
 p*fixate*newest*interleave-best x2
 544 O: 03505 (comprehend lhs :part)
 p*fixate*interleave-best x2 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 po*fixate*condition x9 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x5
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x4 p*generic*indifferent x9
 a*fixate*dont-interleave-best x2
 545 O: 03512 (fixate o29 :selected-id)
 ao*fixate chunk-1378 ao*fixate chunk-1379 ao*fixate chunk-1380
 p*generic*terminate-and-reject ao*substate*count-second
 a*wm*unpack-applied-om x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1381 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1382 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1383 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x3
 po*display*print-operator x2

p*display*dunk-comprehend a*display*t817 p*display*dunk-comprehend a*display*t817 p*generic*indifferent x2

546 O: 03531 (display print-op (t817))
 ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
 a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
 a*wm*unpack-applied-om p*generic*reject ao*display*print*t817
 d*print*t817

547 O: 03461 (comprehend op :new-operator)
chunk-1178 chunk-1277 chunk-1331 chunk-1377 po*display*run*to-expected-op
 po*display*print-operator x2
 p*display*reject-duplicates x3 p*generic*indifferent x3

548 ==>S: S102 (operator no-change)
 a*subgoal*wm-pointer a*subgoal*hold-back-wm-pointer-until-attend a*state*fixate-meta-attributes
 a*substate*initialize-goal-set a*state*important-objects a*substate*create-time
 po*attend
 p*generic*indifferent

549 O: 03536 (attend t817 constant215)
 ao*attend chunk-1384 ao*attend*mark-locally p*generic*terminate-and-reject
 a*dp*unpack-applied-om x3 p*generic*reject
 a*wm*newest-from-not-newest
 ao*attend*previous-not-newest chunk-1385 po*probe*with-previous-goal po*probe*with-part x2
 po*fixate*current-context po*fixate*binding-attribute*target po*fixate*selected-operator
 po*fixate*assertion x2 po*fixate*augmentation po*probe*with-important-object x3
 a*dp*unpack-applied-om p*probe*lhs-best p*probe*repeated-goal*worst x2 p*generic*indifferent
 a*fixate*newest x6
 p*fixate*newest*interleave-best

550 O: 03539 (comprehend lhs :part)
 p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 po*fixate*condition x9 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x5
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x4 p*generic*indifferent x9
 a*fixate*dont-interleave-best x2

551 O: 03548 (fixate new-operator o24 :augmentation)
 ao*fixate chunk-1386 ao*fixate chunk-1387 ao*fixate chunk-1388 ao*fixate
chunk-1389 ao*fixate chunk-1390 ao*fixate chunk-1391 p*generic*terminate-and-reject
 ao*substate*count-second
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1392 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1393 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1394 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1395 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1396 ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x5
 po*fixate*bind-op*id po*probe*with-attribute po*comprehend*objects-attribute
 a*fixate*newest p*probe*new-attribute*best ao*goal-select*comprehend*create-token
 a*goal-select*proposed-during p*generic*indifferent x3
 ao*goal-select*new-goal*fixate/imagine chunk-1397

552 O: 03567 (comprehend new-operator :objects-att)
 ao*comprehend*applied ao*comprehend*create-dp-on-om
 a*state*applied-newer **chunk-1377 chunk-1331 chunk-1277 chunk-1178 chunk-1385**

chunk-1330 chunk-1170 chunk-1177 a*dp*unpack-applied-om x5
a*dp*unpack-applied-om x9
chunk-1377 chunk-1331 chunk-1178 chunk-1277

553 ==>S: S103 (operator no-change)
a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set
a*state*important-objects a*substate*create-time
po*probe*with-previous-goal po*fixate*current-context po*probe*with-attribute
po*fixate*binding-attribute*target po*fixate*selected-operator po*fixate*bind-op*id po*fixate*assertion
x2 po*probe*with-important-object p*attend*old-regions*reject p*generic*indifferent
a*fixate*newest p*probe*repeated-goal*worst a*fixate*newest x5 p*generic*indifferent x9

554 O: O3577 (fixate o24 operator* s-constructor16 :bind-op-id)
ao*fixate chunk-1398 ao*fixate chunk-1399 p*generic*terminate-and-reject
ao*substate*count-first p*probe*new-attribute*best
a*wm*unpack-applied-om x2 p*generic*reject
ao*fixate*unpack-fixation-object chunk-1400 **chunk-473** ao*comprehend*unpack-fixation-object
ao*fixate*unpack-fixation-object chunk-1401 ao*comprehend*unpack-fixation-object
a*wm*unpack-applied-om x2
po*probe*with-important-object
p*probe*new-important-object*best p*generic*indifferent

555 O: O3580 (comprehend operator* :imp-obj condition)
ao*probe*goal p*generic*terminate-and-reject ao*substate*count-second ao*comprehend*applied*first
f:operator-condition*part-of-lhs
p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer ao*probe*unpack-probe-om-to-superop-om
chunk-1402
a*wm*unpack-applied-om
chunk-1358 chunk-1359 chunk-1360 chunk-83 po*comprehend*sp-parts po*probe*with-part
ao*probe*unpack-probe-om-to-superop-om*fixated chunk-1403
ao*probe*unpack-probe-om-to-superop-om*fixated chunk-1404
ao*probe*unpack-probe-om-to-superop-om*fixated x3 chunk-1407
ao*probe*unpack-probe-om-to-superop-om chunk-1408 ao*goal-select*comprehend*create-token
a*goal-select*proposed-during p*probe*lhs-best p*generic*indifferent x2
a*wm*unpack-applied-om x4
po*comprehend*new-operator po*probe*with-important-object x2 po*comprehend*sp-parts
po*probe*with-part
ao*goal-select*comprehend*create-token a*goal-select*proposed-during
ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent x5

556 O: O3586 (comprehend lhs :part)
ao*goal-select*new-goal*probe chunk-1409 ao*probe*goal p*generic*terminate-and-reject
ao*comprehend*applied*second f:operator-condition*lhs-means-rhs
po*fixate*condition x9 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer
p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x4
p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x5 p*generic*indifferent x9
a*fixate*dont-interleave-best x2

557 O: O3584 (comprehend lhs :sp-parts)
ao*comprehend*applied f:operator-condition*lhs-means-rhs ao*comprehend*create-dp-on-om
a*state*applied-newer a*wm*unpack-applied-om **chunk-1385 chunk-1277 chunk-1178**
chunk-1331 chunk-1377
chunk-1384 chunk-1376 chunk-1276 a*dp*unpack-applied-om x5
a*dp*unpack-applied-om x9
chunk-1385 chunk-1377 chunk-1277 chunk-1331 a*wm*newest-from-not-newest

558 ==>S: S104 (operator no-change)
a*subgoal*wm-pointer po*attend*old-regions a*state*fixate-meta-attributes a*substate*initialize-goal-set

a*state*important-objects a*substate*create-time
 po*probe*with-previous-goal po*probe*with-part x2 po*fixate*current-context
 po*fixate*binding-attribute*param po*fixate*binding-attribute*target po*fixate*selected-operator
 po*fixate*selected-id po*fixate*assertion x2 po*fixate*augmentation po*probe*with-important-object
 x3 po*fixate*condition p*attend*old-regions*reject p*generic*indifferent
 p*probe*lhs-best p*probe*repeated-goal*worst a*fixate*newest x9
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x5
 p*fixate*invariant-feature*dont-interleave-best a*fixate*newest x4 p*generic*indifferent
 p*fixate*newest*interleave-best a*fixate*dont-interleave-best x2

559 O: 03610 (comprehend lhs :part)

p*fixate*interleave-best ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*first
 f:operator-condition*lhs-means-rhs ao*substate*count-first
 p*generic*reject a*state*applied-newer

560 O: 03621 (fixate new-operator o24 :augmentation)

ao*fixate chunk-1410 ao*fixate chunk-1411 ao*fixate chunk-1412 ao*fixate
chunk-1413 ao*fixate chunk-1414 ao*fixate chunk-1415 p*generic*terminate-and-reject
 ao*substate*count-second p*probe*new-attribute*best
 a*wm*unpack-applied-om x6 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1416 ao*comprehend*unpack-fixation-object **chunk-1392**
 ao*fixate*unpack-fixation-object chunk-1417 ao*comprehend*unpack-fixation-object **chunk-1393**
 ao*fixate*unpack-fixation-object chunk-1418 ao*comprehend*unpack-fixation-object **chunk-1394**
 ao*fixate*unpack-fixation-object chunk-1419 ao*comprehend*unpack-fixation-object **chunk-1395**
 ao*fixate*unpack-fixation-object chunk-1420 ao*comprehend*unpack-fixation-object **chunk-1352**
chunk-1374 ao*fixate*unpack-fixation-object chunk-1421 ao*comprehend*unpack-fixation-object
chunk-1396
 a*wm*unpack-applied-om x6
 po*probe*with-attribute
 p*probe*new-attribute*best p*generic*indifferent

561 O: 03608 (comprehend new-operator |(previous-goal)|)

ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied*second
 p*probe*repeated-goal*worst x2 p*generic*reject a*state*applied-newer

562 O: 03637 (fixate :condition problem-space create-operator)

ao*fixate chunk-1422 ao*fixate chunk-1423 ao*fixate chunk-1424
 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1425 **chunk-1297** ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1426 **chunk-73 chunk-290 chunk-438 chunk-483**
chunk-568 chunk-806 chunk-1298 ao*comprehend*unpack-fixation-object
 ao*fixate*unpack-fixation-object chunk-1427 **chunk-1299** ao*comprehend*unpack-fixation-object
 a*wm*unpack-applied-om x3
 po*probe*with-important-object x2
 p*probe*new-important-object*best p*probe*new-attribute*best p*probe*new-important-object*best
 p*generic*indifferent x2

563 O: 03642 (comprehend create-operator :imp-obj condition)

ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer

564 O: 03636 (fixate :condition type s-model-constructor)

ao*fixate chunk-1428 ao*fixate chunk-1429 ao*fixate chunk-1430
 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x3 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1431 **chunk-488 chunk-623 chunk-1303**
 ao*comprehend*unpack-fixation-object ao*fixate*unpack-fixation-object chunk-1432 **chunk-489**
chunk-624 chunk-1284 ao*comprehend*unpack-fixation-object ao*fixate*unpack-fixation-object
chunk-1433 **chunk-490 chunk-625 chunk-1304** ao*comprehend*unpack-fixation-object

a*wm*unpack-applied-om x3
 po*probe*with-important-object po*imagine*s-model-constructor x4 po*probe*with-important-object
 p*probe*new-important-object*best p*probe*new-attribute*best a*fixate*newest p*imagine*refract
 a*fixate*newest x2 p*imagine*refract a*fixate*newest p*probe*new-important-object*best
 p*generic*indifferent x6

565 O: 03650 (comprehend s-model-constructor :imp-obj condition)
 ao*probe*goal p*generic*terminate-and-reject ao*comprehend*applied f:s-construct-builds-chunk
 p*probe*repeated-goal*worst p*generic*reject a*state*applied-newer a*state*new-important-object*best
 ao*probe*unpack-probe-om-to-superop-om chunk-1434 ao*probe*unpack-probe-om-to-superop-om
chunk-1435
 a*wm*unpack-applied-om x2
 po*probe*with-important-object
 p*probe*retrieved-by-probe*best p*generic*indifferent

566 O: 03648 (imagine condition s-constructor16)
 ao*imagine chunk-1436 p*probe*non-important-after-imagine*worst x4
 p*generic*terminate-and-reject
 a*wm*unpack-applied-om x2 p*generic*reject
 ao*fixate*unpack-fixation-object chunk-1437 **chunk-475** ao*comprehend*unpack-fixation-object
 ao*fixate*mark-imagined-object x2 chunk-1438
 ao*imagine*imagined-but-seen chunk-1439 a*wm*unpack-applied-om
 a*wm*unpack-applied-om po*probe*with-important-object p*imagine*refract x2
 po*comprehend*recalled-condition po*probe*with-important-object p*probe*new-important-object*best
 p*generic*indifferent
 ao*goal-select*comprehend*create-token a*goal-select*proposed-during p*generic*indifferent x2
 ao*goal-select*new-goal*fixate/imagine chunk-1440

567 O: 03656 (comprehend s-constructor16 :recalled-condition)
 ao*comprehend*applied f:operator ao*comprehend*create-dp-on-om po*display*scroll*to-sp
 a*state*applied-newer a*wm*unpack-applied-om **chunk-1178 chunk-1331 chunk-1277**
chunk-1377 chunk-1385 p*display*dunk-comprehend a*display*t845 p*generic*indifferent
chunk-1330 chunk-1170 chunk-1177 a*dp*unpack-applied-om x5
 a*dp*unpack-applied-om x9
chunk-1331 chunk-1377 chunk-1178 chunk-1277

568 O: 03660 (display scroll:to-sp s-constructor16 (t845))
 ao*comprehend*remember-display-command p*generic*terminate-and-reject ao*display*emulator
 a*topstate*clean-up-old-comprehends-and-displays p*comprehend*best*when-for-last-displayed-region
 a*wm*unpack-applied-om p*generic*reject ao*display*print*t845
 d*print*t503 d*print*t373 d*print*t364 d*print*t323

569 O: 03656 (comprehend s-constructor16 :recalled-condition)

References

- Altmann, E.M. (1993). Learning scope, task analysis, and sharable components. *Proceedings of the Second International Workshop on Multistrategy Learning*. Fairfax, VA: George Mason University.
- Altmann, E.M. and Yost, G.R. (1992). *Expert-System Development in Soar: A tutorial* (Tech Rep CMU-CS-92-151). Carnegie Mellon University School of Computer Science.
- Altmann, E.M., Larkin, J.H., and John, B.E. (1995). Display navigation by an expert programmer: A preliminary model of memory. *CHI 95 Conference Proceedings*. New York: ACM Press.
- Anderson, J.R. (1994). *Learning and Memory*. New York: Wiley.
- Anderson, J.R., and Bower, G.H. (1972). Recognition and retrieval processes in free recall. *Psychological Review*, 79, 97-123.
- Bauer, M. and John, B.E. (1995). Modeling time-constrained learning in a highly interactive task. *CHI 95 Conference Proceedings*. New York: ACM Press.
- Brooks, R.E. (1975). *A model of human cognitive behavior in writing code for computer programs*. Doctoral dissertation, Carnegie Mellon University School of Computer Science.
- 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., and Newell, A. (1983). *The Psychology of Human-Computer Interaction*. Hillsdale NJ: L Erlbaum.
- Chase, W.G. and Ericsson, K.A. (1982). Skill and working memory. In G.H. Bower (Ed), *The psychology of learning and motivation* Vol. 16, pp. 1-68. New York: Academic Press.
- Davies, S.P. (1994). Knowledge restructuring and the acquisition of programming expertise. *International Journal of Human-Computer Studies*, 40(4), 703-726.
- Davies, S.P. (1996). Display-based problem-solving strategies in computer programming. *Empirical Studies of Programmers: 6th workshop* (pp 59-76). Norwood NJ: Ablex.
- Detienne, F., and Soloway, E. (1990). An empirically-driven control structure for the process of program understanding. *International Journal of Man-Machine Studies*, 33(3), 323-342.
- Ericsson, K.A., and Kintsch, W. (1995). Long-term working memory. *Psychological Review*, 102(2), 211-245.
- Ericsson, K.A. and Simon, H.A. (1992). *Protocol Analysis: Verbal reports as data*. Cambridge: MIT Press.
- Ericsson, K.A., and Staszewski, J.J. (1989). Skilled memory and expertise: Mechanisms of exceptional performance. In D. Klahr and K. Kotovsky (Eds), *Complex Information Processing: The impact of Herbert A. Simon*. Hillsdale NJ: L Erlbaum.
- Gellenbeck, E.M. and Cook, C.C. (1991). An investigation of procedure and variable names as beacons during program comprehension. *Empirical Studies of Programmers: 4th workshop* (pp 65-79). Norwood NJ: Ablex.

- Golledge, R.G. (1991). Cognition of physical and built environments. In T. Garling and G.W. Evans (Eds), *Environment, Cognition, and Action*. New York: Oxford University Press.
- Gray, W.D. and 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.
- Howes, A. (1993). Recognition-based problem solving. *Proceedings of the 15th Annual Conference of the Cognitive Science Society* (pp 551-556). Hillsdale NJ: L Erlbaum.
- Howes, A. (1994). A model of the acquisition of menu knowledge by exploration. *CHI 94 Conference Proceedings* (pp 445-451). New York: ACM Press.
- Howes, A., and Young, R.M. (1996). The role of cognitive architectures in modelling the user: Soar's learning mechanism. Submitted to *Human-Computer Interaction* for special issue on Cognitive Architectures and HCI.
- Howes, A., and Young, R.M. (1996). Learning consistent, interactive and meaningful task-action mappings: a computational model. *Cognitive Science*, Vol. (in press).
- Huffman, S.C. (1994). *Instructable Autonomous Agents*. Doctoral dissertation, The University of Michigan Department of Electrical Engineering and Computer Science.
- 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.
- John, B.E. (1996). Task matters. In D.M. Steier and T.M. Mitchell (Eds), *Mind Matters: A tribute to Allen Newell*. Mahwah, NJ: L Erlbaum.
- John, B.E., and Kieras, D.E. (1994). *The GOMS Family of Analysis Techniques: Tools for design and evaluation* (Tech Rep CMU-CS-94-181). Carnegie Mellon University School of Computer Science.
- Kant, E., and Newell A. (1984). Problem solving techniques for the design of algorithms. *Information Processing & Management*, 20(1-2), 97-118.
- Katz, I.R. (1988). *Transfer of Knowledge in Programming*. Doctoral dissertation, Carnegie Mellon University Department of Psychology.
- Kintsch, W. (1970). Models for free recall and recognition. In D.A. Norman (Ed), *Models of Human Memory*. New York: Academic Press.
- Kintsch, W. (1988). The role of knowledge in discourse comprehension: A construction-integration model. *Psychological Review*, 95(2), 163-182.
- Kitajima, M., and 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.
- Laird, J.E. (1984). *Universal subgoalting*. Doctoral dissertation, Carnegie Mellon University School of Computer Science.
- 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.
- Laird, J.E., Congdon, C.B., Altmann, E.M., and Doorenbos, R. (1993). Soar User's Manual: Version 6, Edition 1. <<http://www.isi.edu/soar/users-manual/html/soar6-manual.info.Top.html>>.
- 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.
- Larkin, J.H., and Simon, H.A. (1987). Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 11, 65-99.

- Letovsky, S. (1986). Cognitive processes in program comprehension. *Empirical Studies of Programmers* (pp 58-79). Norwood NJ: Ablex.
- Lewis, C.H. (1988). How and why to learn why: Analysis-based generalization of procedures. *Cognitive Science*, 12, 211-256.
- Lewis, R.L. (1993). *An Architecturally-Based Theory of Human Sentence Comprehension*. Doctoral dissertation, Carnegie Mellon University School of Computer Science.
- Mannes, S.M. and Kintsch, W. (1991). Routine computing tasks: Planning as understanding. *Cognitive Science*, 15, 305-342.
- Mantei, M.M. (1982). *Disorientation Behavior in Person-Computer Interaction*. Doctoral dissertation, University of Southern California.
- Mertz, J.S., Jr. (1995). *Using a Cognitive Architecture to Design Instructions*. Doctoral dissertation, Department of Engineering & Public Policy, Carnegie Mellon University.
- Miller, C.S. (1993). *Modeling Concept Acquisition in the Context of A Unified Theory of Cognition*. Doctoral dissertation, Department of Computer Science and Electrical Engineering, the University of Michigan.
- Mitchell, T.M., Keller, R.M., and Kedar-Cabelli, S.T. (1986). Explanation-Based Generalization: A Unifying View. *Machine Learning*, 1(1), 47-80.
- Nelson, G., Lehman, J.F., and John, B.E. (1994). Integrating cognitive capabilities in a real-time task. *Proceedings of the 16th Annual Conference of the Cognitive Science Society*.
- Newell, A. (1990). *Unified Theories of Cognition*. Cambridge: Harvard University Press.
- Newell, A. and Simon, H.A. (1972). *Human Problem Solving*. Englewood Cliffs NJ: Prentice-Hall.
- Newell, A., Yost, G.R., Laird, J.E., Rosenbloom, P.S., and Altmann, E.M. (1991). Formulating the problem-space computational model. In R.F. Rashid (Ed), *CMU Computer Science: A 25th Anniversary Commemorative*. New York: ACM Press.
- Payne, S.J. (1991). Display-based action at the user interface. *International Journal of Man-Machine Studies*, 35, 275-289.
- Peck, V.A. and John, B.E. (1992). Browser-Soar: A computational model of a highly interactive task. *Proceedings of CHI'92 Conference*. Monterey, CA.
- Pennington, N. (1987). Comprehension strategies in programming. *Empirical Studies of Programmers: 2nd workshop* (pp 100-113). Norwood NJ: Ablex.
- 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.
- Rich, E. and Knight, K. (1991). *Artificial Intelligence (2nd edition)*. New York: McGraw-Hill.
- Richman, H.B., Staszewski, J.J., and Simon, H.A. (1995). Simulation of expert memory using EPAM IV. *Psychological Review*, 102(2), 305-330.
- 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. *CHI 94 Conference Proceedings* (pp 438-444). New York: ACM Press.
- Rieman, J., Young, R.M., and Howes, A. (1996). A dual-space model of iteratively deepening exploratory learning. *International Journal of Human-Computer Studies*, Vol. in press.
- Rist, R.S. (1995). Program structure and design. *Cognitive Science*, 19, 507-562.
- Rosenbloom, P.S., Laird, J.E., and Newell, A. (1987). Knowledge level learning in Soar. *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI87)* (pp 499-504).

- Rosenbloom, P.S.; Newell, A; and 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 Associates.
- Rosenbloom, P.S., Laird, J.E., and Newell, A. (editors) (1992). *The Soar Papers: Research on integrated intelligence*. Cambridge: The MIT Press.
- Sheil, B.A. (1981). The psychological study of programming. *ACM Computing Surveys*, 13(1), 102-120.
- Soloway, E.M. and Ehrlich, K. (1984). Empirical studies of programming knowledge. *IEEE Transactions on Software Engineering*, SE-10(5), 595-609.
- Spohrer, J.C., and Solway, E. (1989). Simulating student programmers. *IJCAI* (pp 543-549). Morgan Kaufmann.
- Steier, D.M. (1987). Cypress-Soar: A case study of search and learning in algorithm design. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp 327-330).
- Steier, D.M. (1989). *Automating Algorithm Design Within an Architecture for General Intelligence*. Doctoral dissertation, Carnegie Mellon University School of Computer Science.
- Thorndyke, P.W. (1981). Spatial cognition and reasoning. In J.H. Harvey (Eds), *Cognition, Social Behavior, and the Environment*. Hillsdale, NJ: Erlbaum.
- Tulving, E. (1983). *Elements of Episodic Memory*. New York: Oxford University Press.
- Van Dijk, T. and Kintsch, W. (1983). *Strategies of Discourse Comprehension*. New York: Academic Press.
- VanLehn, K., Ball, W., and Kowalski, B. (1989). Non-LIFO execution of cognitive procedures. *Cognitive Science*, 13(3), 415-465.
- 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.
- Von Mayrhauser, A. and Vans, A.M. (1995). Program comprehension during software maintenance and evolution. *Computer*, 28(8), 44-55.
- Wiedenbeck, S. (1991). The initial stage of comprehension. *International Journal of Man-Machine Studies*, 35, 517-540.
- Wu, Q. (1992). *Knowledge Transfer Among Programming Languages*. Doctoral dissertation, Carnegie Mellon University Department of Psychology.

Table of Contents

1. Introduction	1
1.1. Outline of the thesis	3
2. The data	5
2.1. Task environment and overview of session	5
2.1.1. Global observations of the programmer's navigation	5
2.1.2. Scrolling from long-term memory	6
2.1.3. The studied interval of the session	7
2.2. Overview of the program and the language	8
2.3. The studied scrolling events	10
2.3.1. Scrolling event 1	13
2.3.2. Scrolling event 2	15
2.3.3. Scrolling event 3	15
2.3.4. Scrolling event 4	18
2.3.5. Scrolling event 5	20
2.4. Summary: memories that lead to scrolling	20
3. The model	23
3.1. Overview of the model	23
3.2. Knowledge: expert, external, and episodic	25
3.3. Soar as the underlying architecture	27
3.3.1. Soar's learning mechanism	27
3.4. Mechanisms to retrieve information from the display	28
3.4.1. Attending: making contact with the display	28
3.4.1.1. Noticing when features become hidden	28
3.4.2. Fixating: copying features from display to WM	29
3.4.2.1. Encoding feature memories	31
3.4.2.2. Encoding episodic memories	31
3.5. Mechanisms to retrieve information from LTM	32
3.5.1. Probing: retrieving information about a related object	32
3.5.2. Imagining: retrieving features to the mind's eye	34
3.5.3. Coverage of probing and imagining	34
3.5.4. Limited working memory	34
3.6. Mechanisms for deliberation	35
3.6.1. Comprehension-goal selection: converging evidence of relevance	35
3.6.2. Subgoal selection: general heuristics	37
3.6.3. Widening the search for the next goal	37
3.7. Mechanisms to change the display	38
3.7.1. Command-goal selection: Immediate relevance	38
3.7.2. The display emulator	38
3.8. Summary: mechanisms and knowledge	39
4. The model's behavior	43

4.1. Challenges in describing the model's behavior	43
4.1.1. Conventions used in the figures	44
4.2. Scrolling event 1	44
4.2.1. The display	46
4.2.2. Model behavior during encoding episode	47
4.2.3. Model behavior during recall episode	48
4.2.4. Summary	52
4.3. Scrolling event 2	52
4.3.1. Model behavior during encoding episode	54
4.3.2. Model behavior during recall episode	55
4.3.3. Summary	58
4.4. Scrolling event 3	58
4.4.1. The model's explanation of why the programmer scrolled	61
4.5. Scrolling event 4	61
4.6. Scrolling event 5	64
4.7. Summary and discussion: images, episodes, and program state	66
5. Measures of the model	69
5.1. Fitting the keystroke protocol	69
5.1.1. The programmer's commands	69
5.1.2. The model's commands	71
5.1.3. Measures of the fit	71
5.1.3.1. Missed programmer commands	72
5.1.3.2. Disregarded programmer commands	73
5.1.4. Reuse: command-proposal rules	73
5.1.5. Summary: constraints from fitting the data	74
5.2. Rules and rule firings	74
5.2.1. Rule counts by category of knowledge and mechanism	74
5.2.2. Reuse: firings compared to selections	77
5.2.3. Summary: numbers as confirmation	78
6. Further characterization of the model	79
6.1. The chain of knowledge that leads to scrolling	79
6.1.1. Where expert-novice differences might reside	80
6.1.2. Why the model scrolls so little	80
6.1.3. Summary: knowledge mediates access to external information	82
6.2. What the model learns	82
6.2.1. Feature memories	82
6.2.2. Recoded facts	83
6.2.3. Episodic memories for features	83
6.3. Why a 2-goal persistence limit on working memory?	84
6.4. Goal selection for purposeful yet flexible behavior	85
6.4.1. Goal duration in terms of programmer time	85
6.4.2. Flexibility through goal-independent subgoals	86
6.5. Limitations of the model	87
6.5.1. Comprehension limited to information-gathering	87
6.5.2. Images limited to being pre-loaded	88
6.6. Comparison to IDXL	88
6.6.1. Incremental comprehension	89
6.6.2. Encoding and retrieval of episodic knowledge	90
6.7. A speculative extension to a more detailed world model	91
6.7.1. Alternative methods for retrieving an identifier	93
6.7.2. Discussion: implications of more detailed motor behavior	94
6.8. Hypotheses concerning screen clutter	95
6.9. Summary	96

7. Episodic long-term working memory	97
7.1. Evidence for episodic LT-WM in our data	98
7.1.1. Rapid and reliable encoding	98
7.1.2. Rapid and flexible retrieval	98
7.1.3. Interference-resistant encoding?	99
7.2. Mechanisms for episodic LT-WM in our model	100
7.2.1. Rapid and reliable encoding	100
7.2.2. Rapid and flexible retrieval	101
7.2.3. No interference-resistant encoding	101
7.2.4. Broader evidence for episodic LT-WM in the model's behavior	102
7.2.4.1. Re-encoding and retrieval of a fact	102
7.3. Discussion: cognitive architecture as constraining theory	104
7.4. Summary: episodic LT-WM compared to LT-WM	105
8. Contributions and future work	107
8.1. Cognitive science: Episodic long-term working memory	107
8.1.1. Directions for future work	108
8.2. Programming psychology: A (limited) model of real work	109
8.2.1. Directions for future work	110
8.3. Concluding summary	110
Appendix A. Model diagram	111
A.1. Retrieving information from display and LTM	113
A.2. Selecting comprehension goals and commands	113
A.3. Constructing and reconstructing working memory	114
Appendix B. Mapping the model to Soar	115
B.1. The vocabulary problem in describing the model	115
B.2. Goals, subgoals, and Soar operators	116
B.3. The goal selection mechanism	116
B.3.1. Discussion: Goal selection constrained by data, not architecture	118
B.4. Guide to fixate and imagine mechanisms	119
B.5. Data chunking: deliberate learning of generalized rules	120
Appendix C. Model code and indices	123
C.1. Table of rules by knowledge and mechanism category (Figure 29, p. 75)	124
C.2. Table of model commands mapped to proposal rules	126
C.3. Model code	127
C.3.1. comprehend.soar	127
C.3.2. display.soar	136
C.3.3. fixate.soar	156
C.3.4. mechanism.soar	165
C.4. Index of rules	176
Appendix D. Detailed model traces for scrolling events	179
D.1. Scrolling event 1 - Figure 21, p. 45	180
D.1.1. Encoding episode	180
D.1.2. Recall episode	180
D.2. Scrolling event 2 - Figure 22, p. 53	183
D.2.1. Encoding episode	183
D.2.2. Recall episode	183
D.3. Scrolling event 3 - Figure 23, p. 59	185
D.3.1. Encoding episode	185
D.3.2. Recall episode, scrolling event 3	186
D.4. Scrolling event 4 - Figure 24, p. 62	188

D.4.1. Encoding episode	188
D.4.2. Recall episode, scrolling event 4	189
D.5. Scrolling event 5 - Figure 25, p. 65	191
D.5.1. Encoding episode	191
D.5.2. Recall episode	191
Appendix E. Display contents, protocol data, and model trace	193
E.1. Aligned display, protocol, and trace	193
E.2. Table of model commands mapped to programmer's keystrokes	225
Appendix F. Rule-level trace of the model	227
References	321

List of Figures

Figure 1: Scrolling events, by screens covered	6
Figure 2: Timeline of long-term memory scrolling events	7
Figure 3: Description of features in the domain	9
Figure 4: Glossary of terms in the domain	11
Figure 5: Overview of scrolling events	12
Figure 6: Scrolling event 1, programmer's behavior	14
Figure 7: Scrolling event 2, programmer's behavior	16
Figure 8: Scrolling event 3, programmer's behavior	17
Figure 9: Scrolling event 4, programmer's behavior	19
Figure 10: Scrolling event 5, programmer's behavior	21
Figure 11: The model comprehends an object by retrieving information	24
Figure 12: The model issues commands to generate new output and scroll	25
Figure 13: Examples of expert knowledge in the model	26
Figure 14: Persistence in WM of a feature that becomes hidden	29
Figure 15: The model uses display, WM, and goal to propose fixation subgoals	30
Figure 16: The model selects a fixation subgoal and encodes memories	30
Figure 17: The model uses WM and LTM to propose probe subgoals	33
Figure 18: The model selects a probe subgoal, recalling and recoding facts	33
Figure 19: Mechanisms in the model	40
Figure 20: Expert, episodic, and external knowledge available to the model	41
Figure 21: Scrolling event 1, model's behavior	45
Figure 22: Scrolling event 2, model's behavior	53
Figure 23: Scrolling event 3, model's behavior	59
Figure 24: Scrolling event 4, model's behavior	62
Figure 25: Scrolling event 5, model's behavior	65
Figure 26: Model and programmer commands	70
Figure 27: The model fails to issue a run command (miss 2)	72
Figure 28: Reuse of command-proposal knowledge	73
Figure 29: Distribution of pre-loaded and encoded rules, and <i>firing counts</i>	75
Figure 30: Proposal rules vs. selections for goals and subgoals, showing reuse	77
Figure 31: Comparison of main categories of rules and firings	78
Figure 32: Imagine subgoals and their intersection with fixate subgoals	81
Figure 33: Goal and subgoal frequency in terms of programmer time	86
Figure 34: Model as it stands, scrolling to u20	92
Figure 35: Model as speculated, scrolling to u20	92
Figure 36: The model recoding a fact and retrieving the new memory	103
Figure 37: Complete diagram of the model	112