Natural Language Comprehension in Soar: Spring 1991

Jill Fain Lehman, Richard L. Lewis, and Allen Newell

March 29, 1991
CMU-CS-91-117

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

Abstract

Multiple types of knowledge (syntax, semantics, pragmatics, etc.) contribute to establishing the meaning of an utterance. Delaying the use of a knowledge source during language processing introduces computational inefficiencies in the form of overgeneration, making it difficult to satisfy the real-time constraint of 200 to 300 words per minute for adult comprehension. On the other hand, ensuring that all relevant knowledge is brought to bear as each word in the sentence is understood is a difficult design problem. As a solution to this problem, we describe in detail the current version of NL-Soar, a language comprehension system that integrates disparate knowledge sources automatically. Through experience, the nature of the understanding process changes from deliberate, sequential problem solving to recognitional comprehension that applies all the relevant knowledge sources simultaneously to each word. The dynamic character of the system results directly from its implementation within the Soar architecture.

This research was sponsored in part by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC, U. S. Air Force, Wright-Patterson AFB, Ohio 45433-6543 under Contract F33615-90-C-1465, ARPA Order No. 7597, and in part by The Markle Foundation, and a National Science Foundation Graduate Fellowship.

1. Introduction: A brief history of NL-Soar

For three and a half years the NL-Soar project has worked toward the goal of creating a general natural language capability for Soar, implemented as a set of problem spaces and operators within the architecture. The motivation for this goal has two sources, one corresponding to each of Soar's two roles as a research effort. In conjunction with Soar as a unified theory of human cognition [26, 31], our motivation is to create an account of human language capability that is consistent with psycholinguistic data (the present system addresses only comprehension, but accounting for generation and acquisition is part of our goal as well). In conjunction with Soar as an integrated, intelligent architecture, our motivation is to provide a common, uniform skill for use by other Soar tasks that involve language. As a system that has not yet met all its goals, NL-Soar is still evolving. Yet, recent changes have produced a version that is significantly different from its predecessors [27, 31]. Our purpose in this report, then, is to describe the system in sufficient detail that its characteristics can be compared and contrasted both with prior Soar versions and with non-Soar systems. In this section, we trace the ideas that led to the current instantiation.

The one idea that has consistently determined the basic design of NL-Soar has been the notion of the *comprehension operator*, first introduced in the William James Lectures in 1987 [31]. The idea of the comprehension operator is a general one, extending beyond language to vision and the other ways in which we comprehend the environment. With respect to language, however, the comprehension operator brings to bear all the knowledge about a word in a given context to produce data structures in working memory that can be used by later comprehension and by problem solving. The application of the comprehension operator to each successive word in the utterance gives NL-Soar's processing behavior the same character as *adult recognitional comprehension*. Namely, the system proceeds left-to-right through the utterance, one word at a time, at a near-constant rate[1], arriving at the end of the utterance with a representation of its meaning complete in working memory. Thus, comprehension operators can be viewed as a realization of Just and Carpenter's *immediacy of interpretation principle* [17] within the Soar architecture.

An important implication of this view is that the instance of the comprehension operator for a given word must access integrated knowledge, simultaneously bringing to bear all the sources involved in establishing meaning. The idea of integrated knowledge sources in comprehension is not new; it has been explored in many natural language systems, in a variety of guises, for reasons of efficiency and robustness if not cognitive constraint. Partial integration of some knowledge sources has been achieved, for example, through the use of semantic grammars [5, 8, 37], domain-specific syntactic grammars [6, 13, 38, 51], semantic annotations to syntactic rules [39, 40], blackboard-style architectures [11, 32, 45], and precompilation techniques [46]. NL-Soar's *comprehension operator principle* goes further along this path of integration, however, by requiring that *all* knowledge sources be accessed in a single processing step.

---

[1] Adult reading rates average 200 to 300 msec per word, corresponding to approximately two or three operator applications in the standard mapping of Soar onto the time-scale of human cognition (see [31]). As described in Section 2.1, NL-Soar actually implements recognitional comprehension with a two operator cycle (attend and comprehend) applied to each word in turn.

The 1988 version of NL-Soar maintained the commitment to comprehension operators and added a new commitment: to annotated models as the data structures produced by the comprehension process [31]. The choice of annotated models as a representation was driven in large part by the system's integration with two other Soar projects: one related to instruction-taking (BI-Soar, [27]), and one related to explaining human behavior in immediate reasoning tasks (IR-Soar, [34]), including syllogistic reasoning. Johnson-Laird had introduced annotated models as an alternative to both pure models and logic-based representations in his work on syllogisms [16]. A pure model enforces a strict one-to-one correspondence between elements in the model and what is being represented. An annotated model, on the other hand, is a variation of the paradigm in which exceptions to the one-to-one rule are permitted that increase the power of the representation language without allowing (and paying the penalty of) the full expressiveness of first-order logic.

Although they differed from the semantic grammar systems with respect to their knowledge sources and meaning representations, the 1987 and 1988 versions of NL-Soar were quite similar to those systems in other ways. Each is a member of what Winograd [50] calls the *situation-action parsers*. As such, their grammars were represented as situation-action rules (also called productions in NL-Soar). A situation-action rule is triggered by structures in working memory, and specifies an action to be taken on those structures. A comprehension operator, then, consists of a set of productions which, together, tell the system what changes to make to the representation of the utterance when a word is encountered in any context. In the early versions of NL-Soar, as in ELI [37], CA [5], and BORIS [8], these situation-action rules were hand-coded. In addition to building up the representation, the rules also created explicit expectation structures in working memory to be matched in later processing. As a result, the process of hand-coding rules required anticipating at design time all of the structures and expectations that needed to be in working memory or had to be removed from memory at all the different points in processing. Thus, maintaining and extending these systems, as well as predicting their behavior when new knowledge was added, were complex and difficult tasks.

The current NL-Soar system (Spring 1991) is, like its predecessors, based on the idea of comprehension operators. It also preserves the 1988 version's commitment to annotated models as a representation. It overcomes the problem of hand-coding comprehension operators, however, by having them arise automatically and incrementally via Soar's learning mechanism. This change allows us to add the knowledge that the system needs for comprehension in a modular and extendable way without giving up the integration demanded by cognitive constraints. As a result, we will see that NL-Soar incorporates two different comprehension processes. *Recognitional comprehension* brings all knowledge sources to bear in a single processing step. When NL-Soar is performing recognitionally its processing has the character of adult comprehension described above. In contrast, *deliberate comprehension* searches through the space of possible interpretations for the current word in a series of steps that access distinct knowledge sources. When NL-Soar is performing deliberately, its processing has the same character as many non-integrated systems. It is the co-existence of these two types of processing and the automatic transformation of knowledge from deliberate to recognitional form that makes

NL-Soar unique.[2]

NL-Soar's performance relies on structures and mechanisms that come from three distinct sources: the Soar architecture, the particular problem spaces and operators chosen for NL-Soar, and the representations chosen for capturing the content of an utterance. Section 2 outlines what each of the three contributes to the system. Section 3 then brings these individual contributions together in a series of examples of the system's performance. We summarize the characteristics and behavior of the system in Section 4.

2. System overview

In this section we demonstrate how comprehension operators arise automatically and incrementally in NL-Soar. We do this first in a general way by showing the Soar architecture's role in comprehension (Section 2.1), without specifying either the results of the comprehension process or the deliberate problem-solving behavior that results from a failure to comprehend recognitionally. Representing the results of comprehension is discussed in Section 2.2. Deliberate problem solving and NL-Soar's problem spaces and operators are then discussed in Section 2.3.

2.1. NL-Soar and the Soar architecture

NL-Soar is implemented within, and takes advantage of, the Soar architecture [23, 31]. Soar belongs to a family of cognitive theories that share important features both in terms of psychological mechanisms and methods of use. The family includes, among others, Anderson's ACT* architecture [2, 43], the work of VanLehn [20, 47, 48], Larkin [25], Kieras and Polson [35], Langley [24], Thibideau, Just, and Carpenter [45], as well as previous work by Newell and Simon [30, 41, 42].

These theories characterize human cognition as goal-directed problem solving. Under this conceptualization, tasks are formulated in terms of *problem spaces* in which *operators* are selected and applied to the *current state* until a *desired state* achieving the goal is reached. The knowledge of when to choose a new goal, when to apply an operator, and what changes the operator creates in the state are contained in a long-term memory, usually realized as *productions*. A production is a type of condition-action (or situation-action) pair; if the conditions are met, then the actions are taken (the production is said to *fire*). Thus, problem solving occurs by matching the conditions in the productions against working-memory elements in the current state and taking the consequent actions to produce a new state. Most of the systems mentioned above also have a learning mechanism to capture the result of problem solving in new productions. The learning mechanism acts as a knowledge compilation device, encapsulating a set of operator/state transitions as a single condition-action pair. Although the systems differ somewhat in how they instantiate these ideas—for example, whether there is a single production

---

[2]We have omitted a 1990 version of NL-Soar [26] from this brief history. In terms of key features (automatic acquisition of comprehension operators and the use of annotated models), the 1990 version did not differ from the current NL-Soar. The two systems did differ in other respects, however. Notably, the 1990 version used an all-paths, bottom-up parsing algorithm and a chart-like structure containing standard phrase-structure constituents. As we describe in Section 2, the current system uses a single-path, combined top-down and bottom-up strategy, and a head and modifier representation for the utterance.

memory or an additional declarative memory, whether all productions fire in parallel or only one production may fire at a time, whether learning occurs automatically or deliberately—the basic model is shared.

As a particular member of this family, Soar can also be described as a system that formulates tasks in terms of problem spaces, operators, and states. Problem solving proceeds in a sequence of *decision cycles*. Each decision cycle accumulates knowledge from long-term *recognition memory* by allowing all the productions whose conditions match working memory elements in the current state to fire in parallel. The knowledge that is added to the state represents preferences concerning the next step to take. Once quiescence is reached (no more productions fire) a *decision procedure* examines the preferences in order to choose a new problem space, operator, or state. If enough knowledge from production memory has accumulated to make the decision procedure's choice unequivocal, the preferred next step is taken and the next decision cycle is entered. If, on the other hand, Soar does not know how to proceed in a problem space because the accessed knowledge does not suggest a next step, or there is conflicting knowledge suggesting more than one step, an *impasse* occurs. In response to an impasse, Soar creates a subgoal and a new problem space in which to acquire the missing knowledge (an impasse within the new space will have the same effect, i.e. Soar creates its own goal-subgoal hierarchy automatically as a result of being unable to proceed). Once an impasse has been resolved by problem solving in the subspace, the learning mechanism (called *chunking* in Soar) combines the conditions that gave rise to the impasse with the result that resolved the impasse in a new production that avoids the impasse in the future. Thus, Soar's learning mechanism is automatic rather than deliberate, being invoked whenever an impasse is resolved.

Figure 2-1 illustrates this process with a number of decision cycles that are common to all of NL-Soar's problem-solving traces, and in many ways the essence of its processing. Decision cycles are numbered along the left margin; in subsequent discussion we use the notation d<number> to refer to a particular decision cycle. The symbols G, P, S, and O stand for goal, problem space, state, and operator, respectively. An impasse is indicated by an arrow and indentation. Ellided processing prior to a decision cycle is indicated by an ellipsis in the left margin.

```
...5   P: Comprehension
   6   S: Comprehension-state
   7   O: attend(John)
   8   O: comprehend(John)
   9   ==>G: operator no-change
  10         P: Language
  11         S: Language-state
...Build: p65
...Build: p76
  21   O: attend(knows)
```

Figure 2-1: A portion of the Soar trace for *John knows Sharyn*, before chunking.

In decision cycles d0 through d4 (not shown), Soar's default initial goal, problem space, and state are chosen and a new sentence beginning with the word *John* becomes available through

Soar I/O functions. By the end of the fifth decision cycle, the Comprehension problem space has been chosen. Its initial state is created in d6.

In d7, the result of the decision procedure is to execute the attend operator which adds the next uncomprehended word in the sentence to the state as the new word. The presence of a new word on the state fires the NL-Soar production propose-comprehend-operator, shown with its English paraphrase below (hereafter, we will show only the English paraphrase). As described earlier, a production is a condition-action pair. The condition appears after the production name but before the arrow, while the action portion appears after the arrow. A condition is given as a set of attribute/value pairs with the attribute indicated by an ‘‘^’’ and its value (or values) immediately following. Values in angle brackets (e.g., <word>) are variables that are bound in the match against working memory. As indicated in the paraphrase at the right, propose-comprehend-operator says that if the state associated with a Comprehension space has a new word on it, then create a preference to comprehend that word. D8 in Figure 2-1 shows that when the decision procedure is finished for this cycle, the comprehend operator has, indeed, been selected.

```
(sp propose-comprehend-operator
  (goal <g> ^problem-space <p> ^state <s>)
  (problem-space <p> ^name comprehension)
  (state <s> ^new-word <word>)
-->
  (goal <g> ^operator <o> +)
  (operator <o> ^name comprehend ^object <word>))
```

Propose-comprehend-operator:
If the problem-space is Comprehension and
the state has a new word on it
Then create a preference to apply the
comprehend operator to the new word

In d5 through d8, there was enough knowledge accessible from long-term memory to make the decision procedure’s choice unequivocal. In the next decision cycle, however, Soar does not know how to proceed in the Comprehension space, because it does not have immediately available from recognition memory the knowledge of how to implement the comprehend operator for the word *John*. Thus, an impasse arises and Soar creates a subgoal to acquire the missing knowledge (d9). Productions in NL-Soar then propose the Language problem space and its initial state as the appropriate space and state to use to acquire this knowledge (d10 and d11). Problem solving continues in this space as before, with each decision cycle resulting in the selection of a problem space, state, operator, or new subgoal. As noted above, when the desired state is reached in the problem space for a subgoal, a chunk is created that avoids the impasse in the future. The chunk makes the knowledge accessible in the problem space that led to the impasse (the *pre-impasse environment*). Thus, p65 and p76 avoid the impasse that led to the subgoal in d9 (we will examine them in more detail in the next section).[3] Their conditions test for the working memory elements from the pre-impasse environment that were needed to reach the desired state in the lower space. Their actions create preferences for the working memory

---

[3]We have skirted the truth a bit in this explanation. In reality, p65 and p76 are created not by reaching a desired state in the Language space, but by virtue of returning results from Language to Comprehension. One can think of this use of chunking as a method of detecting pieces of the desired state as they occur during problem solving. Once a piece of the desired state is present, the appropriate changes are made to the superstate and a chunk is automatically formed. An interesting side effect of this form of learning is that chunks can trigger other chunks. In our example, p65 is formed first, while p76 is formed as a result of later problem solving. This means that the results returned when creating p65 become part of the pre-impasse environment for p76. It happens that p76 relies on some of p65’s results. Thus, chunking puts those attributes and values in p76’s conditions. Should the situation present at d9 occur again, p65 will fire first, triggering p76 which then fires. Since the decision procedure waits until quiescence, however, both chunks will fire within a single decision cycle.

elements' new attributes and values. Thus, next time we encounter the word *John* in similar circumstances, p65 and p76 will fire, avoiding the problem solving in d9 through d20. In other words, p65 and p76 are two of the productions that form part of the comprehension operator for *John*. Although they arose via deliberate problem solving in response to a failure to comprehend recognitionally, the knowledge they contain about what to do when comprehending *John* is now immediately available in the Comprehension space. As Figure 2-2 illustrates, chunking has transformed the sequential application of a series of operators into a single instance of simple production match.

```
...5    P: Comprehension
   6    S: Comprehension-state
   7    O: attend(John)
   8    O: comprehend(John)
   Firing: p65, p76
   9    O: attend(knows)
```

Figure 2-2: A portion of the Soar trace for *John knows Sharyn*, after chunking.

2.2. NL-Soar's representation of utterances

In the previous section, we showed how the Soar architecture supports two separate but interdependent types of comprehension process: recognitional and deliberate. Regardless of which process is evoked by a particular utterance, however, the outcome is the same: a *single* interpretation of the utterance in the form of two related annotated models. The *utterance model* represents the structure of the utterance. It is built primarily to facilitate constructing the *situation model* which represents the utterance's meaning. The two models are tied via reference relations from objects in the utterance model to objects in the situation. Together, the two models provide the context under which comprehension proceeds. The limit of two models is not an absolute—additional models may be needed as new sources of knowledge (for example, a discourse level) are added to the system.

The utterance model represents the structure of the utterance using a head and modifier approach [50] based on dependency theory. In essence, a dependency structure is one in which each node corresponds directly to a word (never to an intermediate constituent as in a phrase-structure grammar) and the arcs specify modifying relations between words. A variety of dependency theories have evolved in linguistics (for example, [12, 14, 28, 29]), differing both in the particular kind of structure allowed (tree [12, 28, 29] or graph [14]) and in the particular set of relations employed (standard grammatical relations [28] or a set of distinguished but non-standard labels [12, 14, 29]). NL-Soar uses a graph structure and grammatical relations based on Winograd's outline of English [50] to label its arcs.[4] Like other implemented systems, NL-Soar distinguishes that subset of its nodes that correspond to open constituents. The combination of this small *active edge set* with a graph size linear in the number of words in the sentence

[4]Currently, only syntactic structures requiring trees are processed by the system. Our commitment to a graph structure comes, therefore, not from the implementation but from the theoretical limitations of tree structures for handling long-distance dependencies [14].

significantly constrains the search required to attach each word.[5]

The situation model is a graph structure representing the objects, properties and relations that are described in the utterance. The objects and their properties are represented by the nodes. The arcs between nodes represent the relations between objects; arc labels consist of a combination of thematic roles, case roles and lexical items. The latter class is quite large (including, for example, both ‘‘on’’ and ‘‘atop’’), reflecting the current system’s lack of commitment to an ontology of primitives. In the remainder of this section, we examine a number of examples of utterances and their models to make these descriptions more concrete.

The four examples that follow are taken from two of the three task domains to which the system has been applied. Example 1 comes from a sentence verification task that is part of work on comprehending instructions and stimuli in immediate-reasoning experiments [27, 34]. Example 2 comes from another immediate-reasoning experiment, a syllogism task. Examples 3 and 4 are taken from our work on a natural language interface to a robot arm [15, 22]. The third domain we have worked in relates to modeling human comprehension of garden path sentences. The examples in Section 3 are drawn from that domain. In addition to making our model representation more concrete, these sentences also serve to illustrate the syntactic, semantic, and pragmatic capabilities present in NL-Soar.

Example 1. In the sentence verification task, a user is first shown a picture of simple geometric shapes, followed by a statement about the picture. The user must judge whether the statement is true or false of the scene. Thus, in this task, the situation model is delivered prior to the utterance through visual comprehension (the form and content of the situation model are the same regardless of whether visual or linguistic comprehension produces it). Judging the sentence true or false is, in terms of language comprehension, essentially a matter of resolving the references in the utterance model against that situation. Figure 2-3 shows an example of the experimental stimuli (top), the situation model assumed to have been delivered by visual comprehension (right), and the utterance model (left) after comprehending *The star above a circle is dark* in the context of the given situation model. Since referents for the all the objects and relations mentioned in the utterance exist in the situation, the sentence can be judged true.

Nodes in the utterance model are represented graphically by an oval for each word in the input. Nodes are connected by arcs labeled with grammatical relations in boldface. Double ovals designate the active edge set. Objects in the situation model are represented by squares. Properties of and relations among objects are in boldface, with relations represented by arcs. Note that *star* in the utterance model is italicized (indicating it represents a word in the input) while ‘‘star’’ in the situation model is not (indicating it represents the concept). Dotted arcs between the two models indicate the referent relations established by comprehension. These relations may change over the course of the comprehension process. When the word *star* is encountered, for example, its referent is ambiguous (o1 or o2). While the word *above* does not

[5]One of the traditional advantages of dependency grammar over constituent grammars has been that the former does not represent word order and can, therefore, easily process discontinuous structures (for example, *wearing earrings* in *A man walked by wearing earrings* [50]). The current implementation of the active edge set in NL-Soar assumes contiguity, however. Thus, although the active edge constrains search and eliminates expensive chunks [44], some way to relax the notion is required.

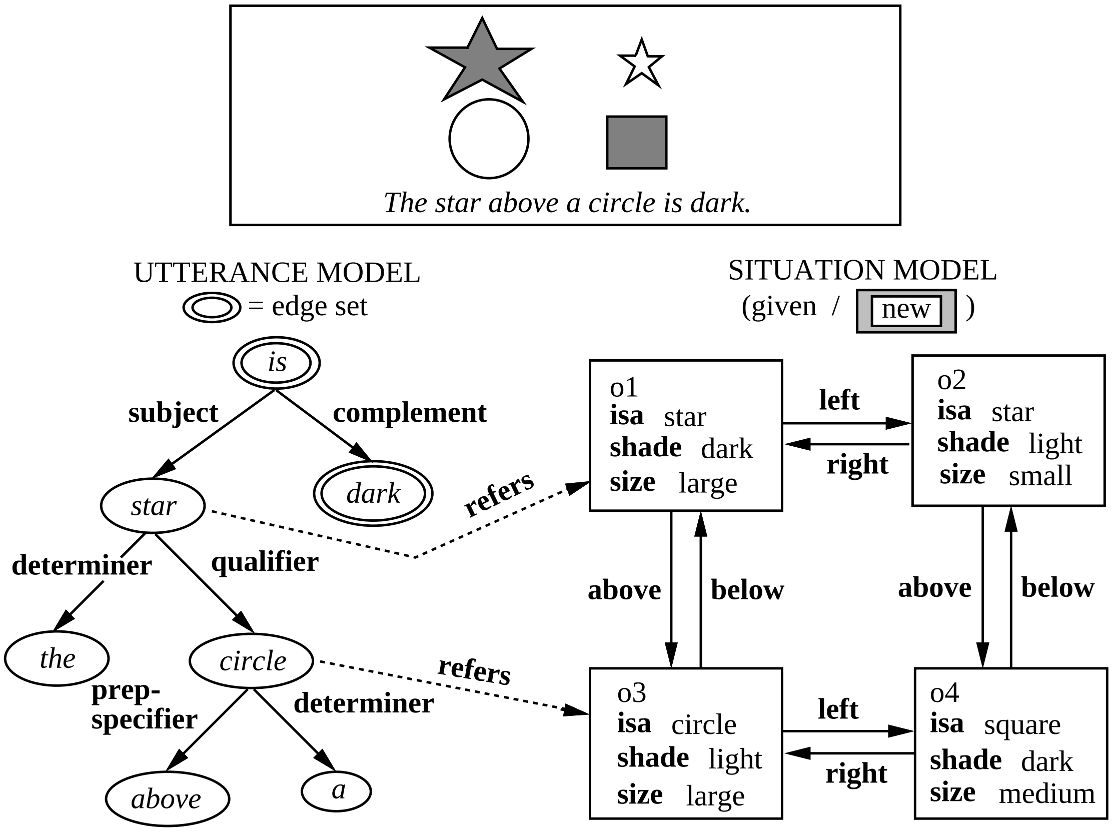

Figure 2-3: The models resulting from comprehension of *The star above a circle is dark.*

disambiguate the reference—each of o1 and o2 is **above** some object—the word *circle* constrains the reference to o1.

Example 2. In the syllogism task, a conclusion must be generated that follows from two related premises (although IR-Soar [34] provides the conclusion in a form similar to the utterance model, NL-Soar does not yet perform the language generation). Suppose the two premises are *All artists are chemists* and *All chemists are beekeepers*. The unshaded portion of the situation model in Figure 2-4 shows the meaning assigned to the first premise, while the utterance model and shaded portion of the situation model correspond to comprehension of the second premise. Note that the situation model continues to grow (that is, processing of the second sentence goes on in terms of the context established by the first sentence) although the utterance model is treated as a temporary structure.[6] The reference to more than one artist/chemist/beekeeper is indicated by the use of an annotation (many) in the situation model. This is an example of the limited way in which annotated models may violate a strict one-to-one

[6]Although NL-Soar can process multiple, connected sentences, the size of the resulting situation model has, to date, been kept quite small. This has allowed us to ignore temporarily the issue of establishing and changing focus of attention that is a natural by-product of larger situation models.

correspondence between model elements and what is being represented.[7] Here, with a single object we represent that there is more than one artist that is also a chemist and a beekeeper. What cannot be represented within the annotated model paradigm, however, is the universal quantifier. The limited representational power of annotated models appears to be an important factor in explaining human performance in this task [16, 33].

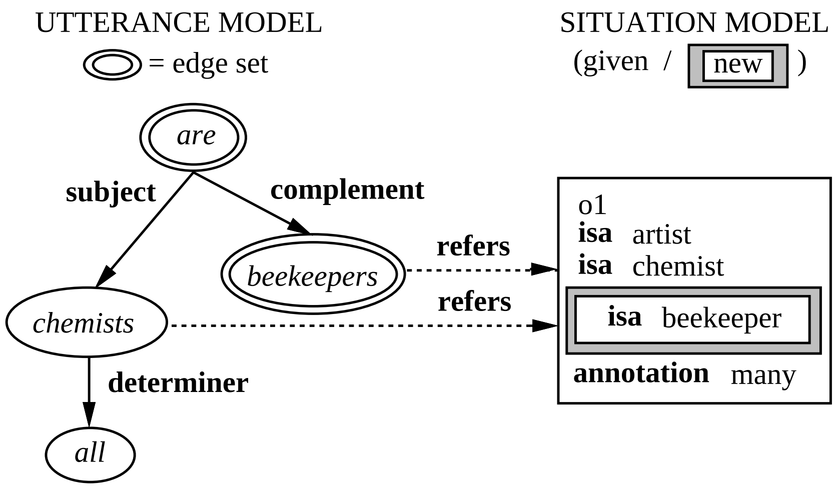

Figure 2-4: The models resulting from comprehension of *All chemists are beekeepers* after having read *All artists are chemists.*

Examples 3 and 4. The last two examples contrast comprehension of the same sentence (*Put the red block on the blue block on the table*) under two different situations. They demonstrate the potential biasing effect of referent resolution within the single-path process. In the situation depicted at the top of Figure 2-5 and in the corresponding situation model, there is no red block that is on a blue block so the phrase *on the blue block* is considered as the beginning of the locative and the full description, *on the blue block on the table*, is eventually resolved to indicate o3. The interpretation of the whole utterance thus has the effect of moving o1 onto o3.

In the situation depicted in Figure 2-6, however, there is a red block on a blue block (o4) as well as a blue block on the table (o3). Thus, the two possible points for attachment of *on the blue block* cannot be distinguished on the basis of the existing situation. If the attachment to the noun is proposed first, the models shown in Figure 2-6 will result, otherwise the system will produce those of the previous figure. Thus, the meaning of the utterance will be consistent with, but may be underconstrained by, the knowledge sources available.

In addition to making the model representation concrete, the examples given above elucidate a number of other characteristics of NL-Soar. They show, for example, that it is a single-path comprehender, producing only one interpretation of an utterance. They indicate that the system's syntactic coverage is limited to fairly simple constructions (actually corresponding to about

[7]The use of the many annotation to indicate plurality is only one possible choice. As an alternative, one might represent the plural by a small, fixed number of identical objects.

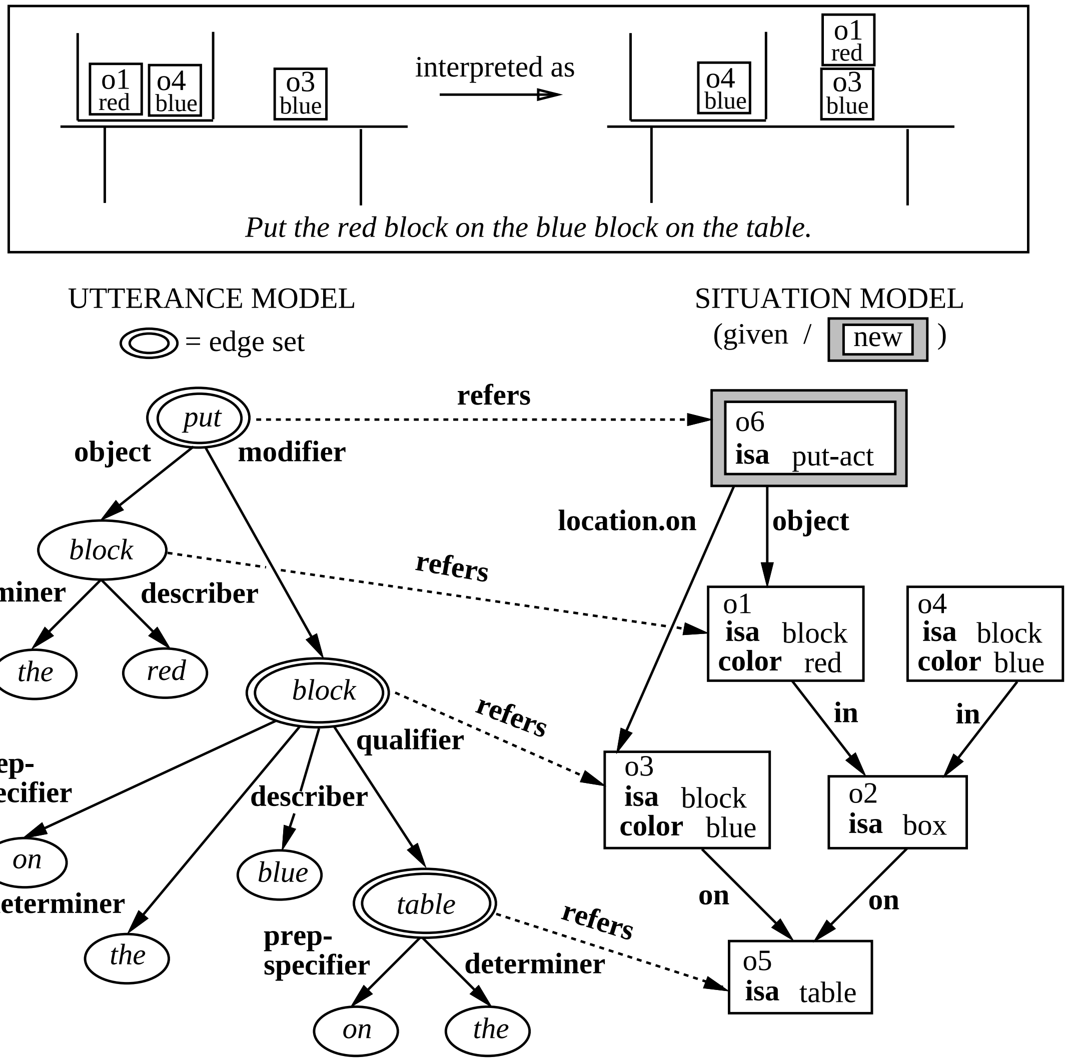

Figure 2-5: The models for *Put the red block on the blue block on the table* when there is no red block on a blue block.

seventy-five percent of James Allen's description of basic English [1]). The system's representation of meaning is fairly simple as well, and currently lacks an ontology of primitive properties and relations. Finally, they demonstrate that NL-Soar processes connected text, maintaining that connectivity through referent resolution to previously established objects and events in the situation.

2.3. NL-Soar's problem spaces and operators

How do we map a sequence of words into an utterance model and situation model? While the architecture provides a framework for answering this question, it does not provide the answer itself. The missing pieces in our description of NL-Soar, then, are the actual problem spaces and operators implemented within the architecture discussed in Section 2.1 and creating the structures discussed in Section 2.2. These are shown in Figure 2-7. The problem space names

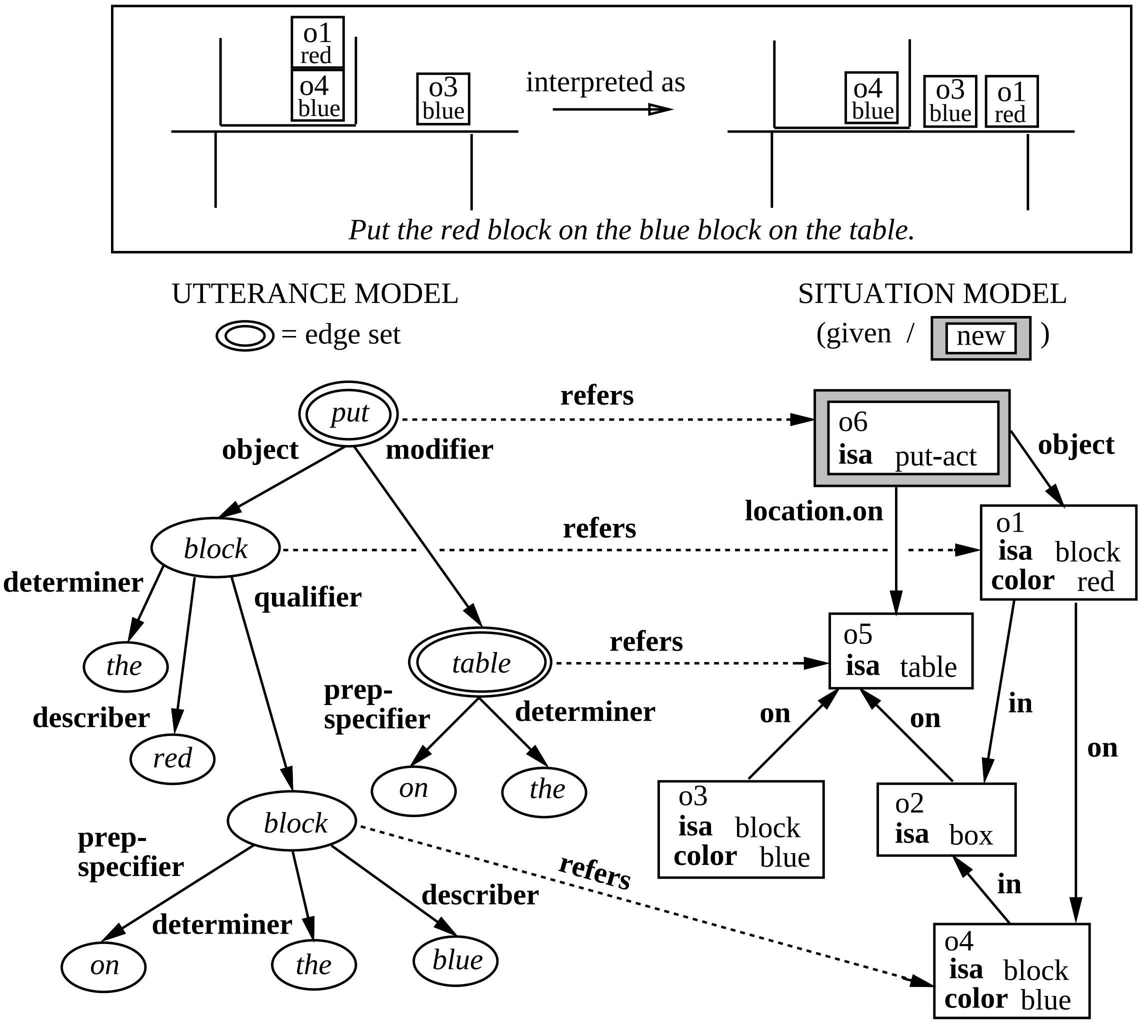

Figure 2-6: The models for *Put the red block on the blue block on the table* when there is a red block on a blue block.

are listed along with their operators at the left of the figure, with a brief description of the problem space's behavior at the far right. In the middle portion of the figure, problem spaces are represented by triangles, states by circles, and operators by arrows. The hierarchical relationship among the problem spaces is indicated downward by impasses and upward by chunking.

At the top of the hierarchy is the Comprehension space, seen previously in d5 through d8 of the trace in Figure 2-1. This is the space with the character of adult recognitional comprehension. It is the space of comprehension operators, in which attention proceeds left-to-right through the sentence, each word being comprehended in turn, its meaning established by a single operator application that brings all the relevant knowledge to bear simultaneously. It is the space in which comprehension requires no search, being achieved, instead, through parallel, constant-time production match. It is also the space whose knowledge of how to comprehend arises automatically through chunking.

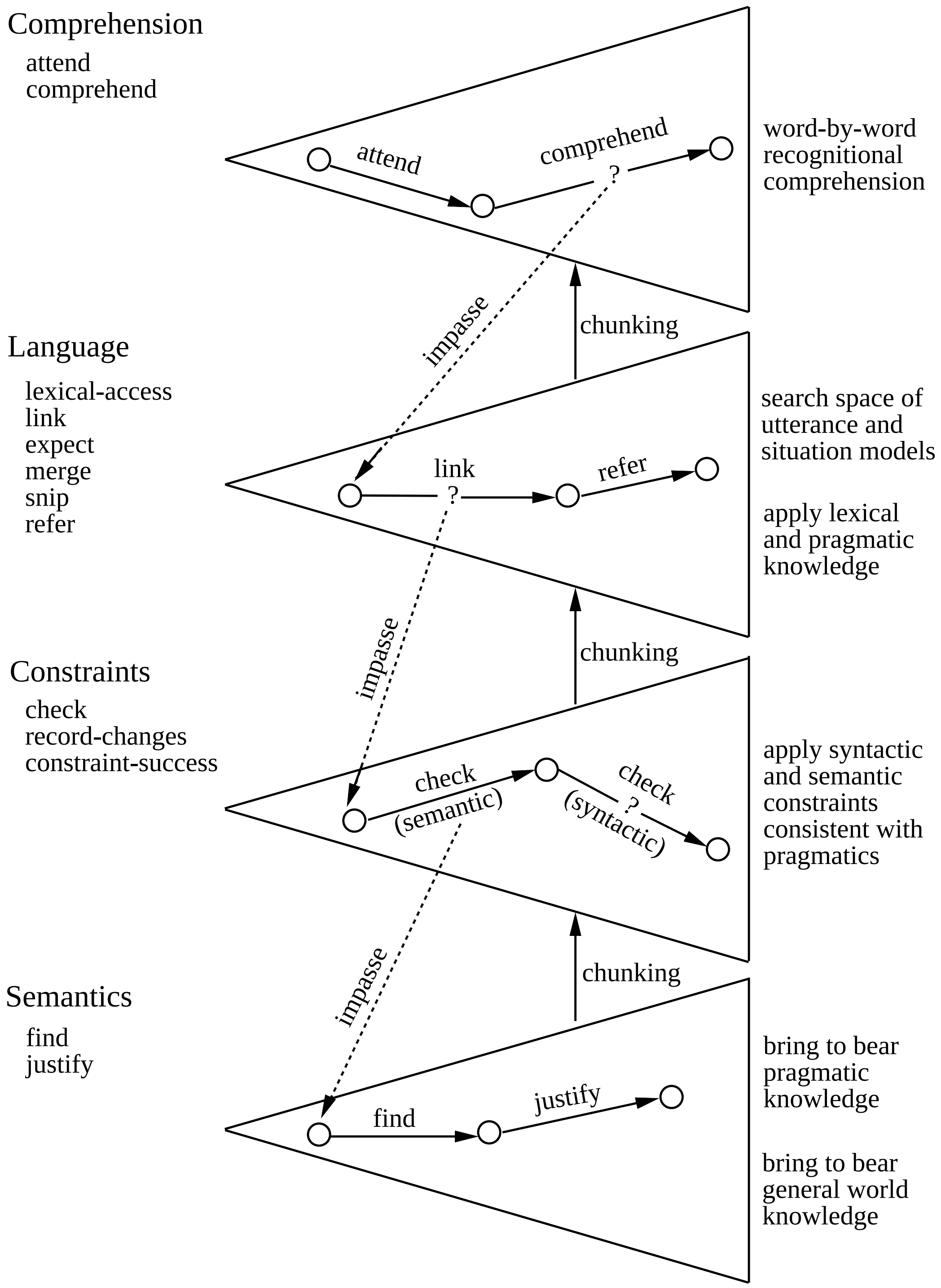

Figure 2-7: NL-Soar's problem spaces and operators.

As we saw in Figure 2-1, when comprehension cannot proceed recognitionally, an impasse arises leading to a goal to acquire the knowledge missing from Comprehension by search. This search occurs in the Language space, whose operators can be tried in various sequences to find a way of attaching the new word and its meaning to the current utterance and situation models. The lexical-access operator provides knowledge about the different grammatical roles and semantic senses of the word that guides these attachments. The link, expect, merge, and snip operators all result, directly or indirectly, in the attachment of the new word to the utterance model (see Section 3 for greater detail). The refer operator, on the other hand, is responsible for changes to the situation model as well as tying the utterance and situation models together. Regardless of the amount of search required to find an attachment, the result of problem solving in Language is to resolve the impasse on the comprehend operator. Chunking combines the conditions that led to the impasse with the results of the search to augment NL-Soar's recognitional comprehension capability.

The Constraints space is the source of the syntactic, semantic, and pragmatic knowledge that permits or vetos the implementation of a link, expect, merge, or snip operator proposed in the Language space (pragmatic knowledge is also used in Language, but not to constrain attachments). Unless the syntactic, semantic, and pragmatic knowledge relevant to an attachment has been made available in Language through chunking in the past, a proposed attachment leads to an impasse so that all pertinent constraints can be brought to bear via check operators (we will see specific examples of check operators in Section 3). In simplest terms, we can consider the Constraints space as a kind of oracle. Via an impasse Language asks the question, *Is this attachment syntactically, semantically, and pragmatically consistent?* and Constraints resolves the impasse by answering *yes* or *no* (*yes* is indicated by using the constraint-success operator after all check operators have passed, *no* by resolving the Language impasse after the failure of any single check operator). These terms are too simplistic, however—the real result of resolving the impasse is not just a *yes* or *no* but a chunk whose conditions capture the critical syntactic, semantic, and pragmatic features of the context that led to that answer, and whose actions record changes to be made to the utterance model. Because the chunk becomes part of the knowledge in Language, syntactic, semantic, and pragmatic knowledge becomes part of the deliberate comprehension search. In turn, these three types of knowledge become integrated with Language's lexical and pragmatic knowledge in the comprehension operator chunks returned from Language to Comprehension.

The problem space at the bottom of the hierarchy shown in Figure 2-7 is Semantics. When the Constraints space does not have the knowledge needed to verify an attachment's semantic consistency, a search for that knowledge occurs here. The Semantics space has two ways of validating an attachment: pragmatic justification and inference. In the former, the find operator looks at the current situation model for instances of the proposed attachment. If an instance of the relation is already present, it must be semantically correct. If, for example, there is already a blue block in the situation, then blue must be a legitimate modifier for block. If no evidence is found in the situation model, the justify operator uses NL-Soar's general world knowledge to infer a validating relation.[8] Thus, if no blue block was present, justify would conclude that *blue* could modify *block* because blue modifies physical-objects and block **isa** physical-object.

---

[8]Actually, a justify operator is usually implemented in another instance of the Constraints space via an impasse. In this Constraints space, check operators are proposed that access world knowledge. The system's world knowledge currently consists of a fairly sparse **isa** hierarchy. We are considering ontologies such as Penman's Upper Model [3] as candidate knowledge sources for this space. Such an ontological commitment will, of course, have consequences for the set of properties and relations used in the situation model as well.

Chunks from Semantics bring the reasoning done in the lower space into Constraints in production form. There it becomes part of Constraints' problem solving, to be returned in its chunks to Language and so on up to Comprehension. In this way, even the semantic and pragmatic knowledge brought to bear at the bottom of the hierarchy is integrated into comprehension operators.

In this description of NL-Soar's problem spaces and operators we have focused on the constant flow of knowledge upward toward Comprehension. It is important to remember, however, that chunking results in increased efficiency in each problem space in the hierarchy by transforming search in the lower spaces into direct operator implementation in the higher space under similar circumstances. Still, chunking is only one aspect of the process of language comprehension in NL-Soar. The interplay of chunking, search, annotated models, and knowledge about language and the world is demonstrated next.

3. The Comprehension Process

The previous section described the building blocks from which NL-Soar has been constructed: the Soar architecture, a particular set of problem spaces and operators, and a representation of the result of the comprehension process as models. We are now ready to show how each of these pieces contributes to the general model of adult recognitional comprehension outlined in Section 1.

We build up a detailed picture of the working system by examining increasingly complex variations on a simple sentence. We begin in Section 3.1 by extending our analysis of the sentence fragment in Figure 2-1 to include the entire sentence, *John knows Sharyn*. This example introduces specific instances of many of the operators described in Section 2.3, shows concretely how the utterance and situation models are constructed, and demonstrates how comprehension operators arise in a bottom-up fashion. In Section 3.2, we examine a variation of this initial sentence that demonstrates how comprehension operators come to include top-down knowledge. This variation also allows us to look at the generality and transfer of the chunks learned in the first example. The example in Section 3.3 builds on knowledge attained from problem solving in the first two examples. It completes our description of NL-Soar by demonstrating the system's limited recovery capacity when its single-path mechanism is led astray.

3.1. Example 1: Simple, bottom-up construction of comprehension operators

We begin by looking in detail at a simple declarative sentence taken from our work on garden path phenomena. The sentence is *John knows Sharyn*, considered in the context of an initially empty situation model. As a point of reference, Figure 3-1 shows what the processing trace looks like when NL-Soar is capable of pure recognitional behavior for this sentence. The utterance model and situation model are also shown.

Soar interacts with the external environment via state changes in its Top problem space. In the trace above, the presence of new input on the state triggers the proposal of the comprehend-input operator (d3). Since the knowledge that implements this operator is not available in the Top problem space, an impasse arises in d4 and a subgoal is created to resolve the impasse. The comprehend-input operator is implemented in the Comprehension space. This space is established along with its initial state in d5 and d6. Next, each word in the utterance is attended

0 G: g1
1 P: Top
2 S: Top-state

Type your input > *John knows Sharyn.*

3 O: comprehend-input
4 ==>G: operator no-change
5 P: Comprehension
6 S: Comprehension-state
7 O: attend(*John*)
8 O: comprehend(*John*)
9 O: attend(*knows*)
10 O: comprehend(*knows*)
11 O: attend(*Sharyn*)
12 O: comprehend(*Sharyn*)
13 O: attend(.)
14 O: comprehend(.)

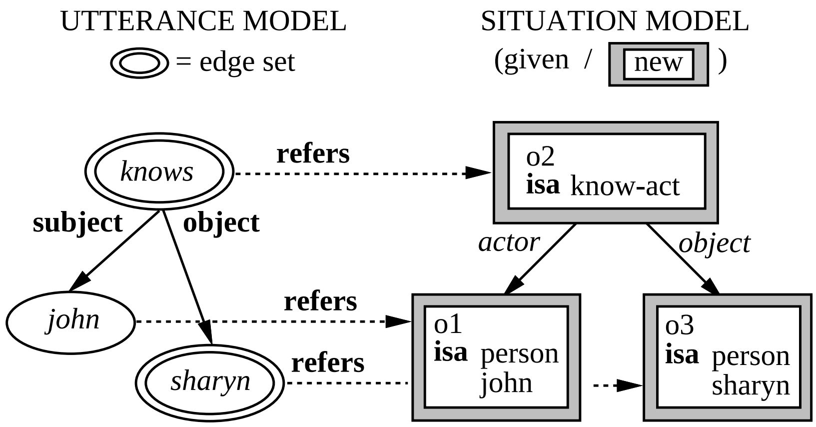

Figure 3-1: Expected recognitional behavior for *John knows Sharyn* and the resulting utterance and situation models.

to and comprehended in turn. The comprehension of each word requires only a single operator—no search is involved.

Let us now consider, as we began to in Section 2.1, what happens when the comprehension operators needed to understand this utterance are not yet available. As we saw in Figure 2-1, when a word has been attended to but the comprehension operator for that word does not exist or is inappropriate in the current context, an impasse is detected by the Soar architecture and a subgoal is created to resolve that impasse. NL-Soar then attempts to resolve the impasse via deliberate problem solving in the Language space. The result of that problem solving are chunks that extend the comprehension operator for the word being comprehended. Figure 3-2 shows this process for the word *John*, including those portions of the trace ellided in Figure 2-1.

Processing through d8 proceeds as in Figure 3-1, with the attend operator creating a node for the word *John* to be linked to the utterance model. At d9, however, the lack of knowledge to implement the comprehension operator for *John* is detected as an impasse and a subgoal is created to resolve it. The first step in the Language space (d12) is to access the lexical definitions of *John* (*John* has only one, but, in general, a word may be polysemous). Each lexical definition is given as a *profile* of attributes and values that represent one interpretation of the word. The production that accesses the lexical entry for *John* is given below. In this and future productions, the *&* symbol separates the multiple values that may fill an attribute while the word *and* separates clauses in the condition and action. In addition, we will sometimes use labels of the form a<number> in the right margin to provide easier reference to pieces of the production. As shown, this profile includes syntactic information (a1 and a2), semantic classifications (a3 and a4), the grammatical roles that the interpretation may assign or receive in the utterance model (a5), and information used in establishing referential ties to the situation model (a6 and a7).

```
  0    (see Figure 3-1)
...7   O: attend(John)
  8    O: comprehend(John)
  9    ==>G: operator no-change
  10       P: Language
  11       S: Language-state
  12       O: lexical-access(John)
  13       O: link(John, first-word)
  14       ==>G: operator no-change
  15           P: Constraints
  16           S: Constraints-state
  17           O: record-changes
  18           O: check(progress)
  19           O: constraint-success
  Build: p32
  Build: p65
  20       O: refer(John)
  Build: p76
  21   O: attend(knows)
```

Figure 3-2: Deliberate comprehension of *John* in *John knows Sharyn.*

```
Language*lexical-access*JOHN:
If    the problem space is Language and
      the lexical-access operator is being applied for John
Then add to the state a profile with
          subclass = count & proper                         a1
          number = 3s                                       a2
          most-specific-class = john                        a3
          base-level-class = person                         a4
          receives = subject & object                       a5
          unused referring-information = isa john           a6
          template = isa john & isa person                  a7
```

Once the lexical definitions are available in working memory, the next step in the Language space is to find those profiles with a role to assign or receive that matches a role to be received or assigned by a profile on an active edge in the utterance model. For every (assigner, receiver, role) triple that can be found, a link operator is proposed. If more than one link is proposed, they are tried in an arbitrary order. In our example, *John* has only one definition and is the first word in the sentence, so only the first-word link is proposed (d13). Note that because NL-Soar is a single-path parser, even if many link operators are proposed, only the first successful link is implemented as changes to the utterance and situation models (what we mean by ‘‘successful’’ will become clear in the discussion that follows).

The proposal of a link operator means that an assigner/receiver match has been found between the new word and an active edge of the utterance model. The actual implementation of the link, however, is permitted only after all the relevant constraints on the link have been passed. To insure that all constraints pass, the link operator is implemented via search in the Constraints space (see d14-d16 in Figure 3-2).

There are two kinds of constraints that must be satisfied before a link is made. The first kind are the semantic and syntactic constraints triggered by the specific link itself (for example, a number agreement constraint is triggered by a link for the subject role). Semantic constraints may also bring pragmatic information to bear, as explained in Section 2.3. In the process of passing these constraints some features on the profile of the assigner or receiver may be modified (this is similar to setting or modifying the contents of registers in an ATN-style parser [50] or to unifying feature vectors in a unification-style parser [40]). For instance, the number feature on a regular verb's profile always changes when the verb is linked to the subject (it begins as 1s, 2s, 1p, 2p, 3p and changes to whichever subset agrees with the subject nounphrase). As another example, the roles that remain to be assigned or received may change, allowing exocentric constructions.[9] When all the syntactic and semantic constraints have passed, the record-changes operator puts the current word's correct sense and the mutually consistent feature sets for the assigner and receiver profiles in a *change record* (d17). The use of a declarative structure to record changes ensures that the sets of compatible features remain tied together during subsequent processing (why this is important will become clear as we follow the change record back up the goal hierarchy to the Comprehension space). The change record for *John* as the first word is quite trivial: it simply preserves the profile we've already seen.

The second kind of constraint that must be passed is called the *progress* constraint. The progress constraint forces NL-Soar to maintain a single, cohesive interpretation of the utterance at all times. It does this by ensuring that all nodes in the utterance model will still be connected after the current Language operator (in this case link) has been implemented (a connected utterance model implies a well-formed situation model, because all semantic constraints and pragmatic information have been taken into account). It is always checked after a change record has been created (d18), so that the effect of those changes on the connectedness of the resulting model can be examined. The progress constraint is passed trivially for the first word in a sentence, but we will see in Section 3.1 that passing the progress check can be quite complex.

With all constraints passed, a final state has been reached in the Constraints space. Recall that a chunk is created whenever an impasse is resolved or a result is returned to a higher space. In d19 both events occur. First, the constraint-success operator signals that the impasse in the Language space has been successfully resolved by notating the link operator with both *constraints-passed* and the change record. This leads to the creation of chunk p32, shown below:

Chunk p32:
If the problem space is Language and
    the link operator is assigning the first-word role to a profile, p, for a word
Then mark the link operator as having all constraints passed and
    add a change record to the link operator that
    makes the word's only profile p and
    links the word to the utterance model as the first word

---

[9]An exocentric construction is one that cannot function as a substitute for its head. Noun-verb constructions are exocentric, since the resulting phrase is neither a noun nor a verb. In contrast, an adjective-noun construction is *endocentric*, because it can be substituted for the noun in order to further elaborate a sentence [50].

Next, productions in the Language space return the change record to the Comprehension space as an intermediate result, creating p65 (below), part of the comprehension operator for *John*. The appearance of the change record in the Comprehension space immediately triggers productions that make the changes to the model, recompute the active edge set, and copy the new model and edge set down to the Language-state. Thus, the changes that establish the correct profile for *John* in the Comprehension and Language spaces are actually made during d19.

Chunk p65:
If the problem space is Comprehension and
the Comprehension-state has no edge and
the Comprehension-state has an incoming word, w, and
the comprehend operator is being applied to w and
w is *John*
Then add a change record to the comprehend operator that
links w to the utterance model as the first word and
makes w's profile
subclass = count & proper
number = 3s
most-specific-class = john
base-level-class = person
receives = subject & object
unused referring-information = isa john
template = isa john & isa person

Note that despite having returned an intermediate result, we have not yet resolved the impasse on the comprehend operator by reaching a final state in the Language space. The last operator on the path to a final state in the Language space brings pragmatic information to bear on the interpretation by finding the referent set for *John*. The refer operator is triggered by the unused referring-information on the profile (see p65, above). In the simplest case, the refer operator selects objects already in the situation model that match the new word's referring-information. If, as in our example, there is no matching situation object, one is created from the profile's template. As we saw in Section 2.2, in more complex situations, the system's preference for interpreting descriptions as referring to pre-existing objects in the situation model may bias the interpretation of an utterance.

With a referent established, the final state in the Language space has been reached and the impasse that led to the subgoal in d9 resolved. Chunk p76, also part of the comprehension operator for *John*, results:

Chunk p76:
If the problem space is Comprehension and
the Comprehension-state has a word, w, on the edge and
the comprehend operator is being applied for w and
w's profile has a template and unused referring-information = isa john and
the Comprehension-state has a situation model and
the situation model has no object that isa john
Then add to the situation an object that instantiates the template and
mark that object created and
add that object to the profile as a referent
mark the referring-information as used

In fact, p76 is triggered by p65, as described in Section 2.1 (p65 places unused referring-information on the profile, which p76, in turn, uses to create a referent and connect the two models). Together the two chunks give the system recognitional comprehension of *John* when it appears as the first word in a sentence and no referents are available in the situation model.

Now that we have followed a change record from its creation in Constraints back to its role as part of the comprehension operator, we are in a position to explain why this declarative structure is necessary. The movement of knowledge in the system from deliberate to recognitional form will eventually result in chunks whose simultaneous firing indicates local ambiguity in the utterance. If all the chunks made their changes directly to the models, the system could detect that conflicting interpretations were possible but could not detect which changes were caused by which interpretation. The use of a change record creates the opportunity for the ambiguity to be detected before any changes are made, and allows knowledge to be brought to bear to make a principled selection of a single interpretation.[10] Once a particular interpretation's change record has been selected, other productions that rely on the established profile will fire to complete the comprehension operator's implementation in the same decision cycle.

The example of deliberate reasoning we saw in processing *John* was quite simple. Although we discussed the problem solving steps taken in the Language space in some detail, the contribution of the Constraints space was minimal, and the lower spaces (as outlined in Section 2.3) were not required. By following the comprehension process for the word *knows*, we can see the contribution of the other spaces more clearly. Figure 3-3 continues where Figure 3-2 stopped, with NL-Soar attending to the word *knows*.

In d21, the attend operator creates a node for the word to be linked into the utterance model. Then, as expected, an impasse occurs when the system tries to comprehend the word recognitionally (d22 and d23). The Language space is chosen to find out how to comprehend *knows*, with the lexical-access operator chosen as the first step (d24-d26). The production that establishes the lexical entry for *knows* is shown below:

Language*lexical-access*KNOWS:
If the problem space is Language and
the lexical-access operator is being applied to *knows*
Then add a profile to the state with
subclass = conjugate
number = 3s
tense = present
most-specific-class = know-act
base-level-class = cognitive-act
assigns = subject & object & clausal-object
requires = role1 & role2 a1
role1 = subject
role2 = object & clausal-object a2
maps = subject to actor & object to object & clausal-object to thought a3
restricts = actor to isa animate & object to isa thing & thought to isa act a4
unused referring-information = isa know-act
template = isa know-act & isa act

[10]The ambiguity is detected as an attribute impasse [21] on the change-record attribute of the comprehend operator. At present, the system resolves such impasses arbitrarily unless one interpretation has a referent in the situation model and the other interpretation does not. In that case, the impasse is decided in favor of the interpretation with the existing referent.

```
...21   O: attend(knows)
   22   O: comprehend(knows)
   23   ==>G: operator no-change
   24         P: Language
   25         S: Language-state
   26         O: lexical-access(knows)
   27         O: link(knows, John, object)
   28         ==>G: operator no-change
   29               P: Constraints
   30               S: Constraints-state
   31               O: check(word-order)
   Build: p89
   32         O: link(knows, John, subject)
   33         ==>G: operator no-change
   34               P: Constraints
   35               S: Constraints-state
   36               O: check(word-order)
   37               O: check(np-well-formedness)
   38               O: check(number-agreement)
   39               O: check(semantic-consistency)
   40               ==>G: operator no-change
   41                     P: Semantics
   42                     S: Semantics-state
   43                     O: find
   44                     O: justify
...50               O: record-changes
   51               O: check(progress)
   52               O: constraint-success
   Build: p186
   Build: p187
   53         O: refer(knows)
   Build: p202
   54         O: refer(knows)
   Build: p212
```

Figure 3-3: Deliberate comprehension of *knows* in *John knows Sharyn*.

It contains the same type of information that we saw in the definition of *John*, with a few noteworthy additions. In particular, information is included that dictates roles that must be filled (a1), roles that are mutually-exclusive (a2, as interpreted by other productions), the mapping between grammatical utterance roles and relations in the situation model (a3), and restrictions on what types of situation objects can fill a role (a4). The influences of these new attributes will be seen below.

Recall that the single active edge (*John*) can receive either a subject or object role. Since *knows* can assign either role, two link operators have been proposed at the end of the decision procedure in d26. With no knowledge yet available to prefer one role over the other, the object link is arbitrarily instantiated first. The knowledge of whether or not to implement the link resides in the Constraints space, so an impasse occurs. In d31 a check operator is applied in order to verify the word order for the object link. Since the syntactic object must follow the verb, the proposed link fails this constraint. That failure is captured in a general way as chunk p89 in the Language space. In the future, no impasse will be necessary to know that it is incorrect to link a receiver on the edge of the utterance model and an assigner that is the new word via the object role.

Since the object link fails, the subject link is proposed in d32. Once in the Constraints space, this link passes three syntactic tests: word order (d36), well-formedness of the nounphrase (d37), and number agreement between the subject and verb (d38). The productions that signal success for these constraints are given below:

```
Constraints*word-order*subject-verb*ok:
If      the problem space is Constraints and
        the check operator for word order on (assigner, receiver, role) is being applied and
        the role is subject and
        the receiver is on the edge
Then mark the constraint as having passed

Constraints*np-well-formed*proper-noun*ok:
If      the problem space is Constraints and
        the check operator for np-well-formedness of a node is being applied and
        the node’s profile has subclass = proper
Then mark the constraint as having passed

Constraints*number-agreement*ok:
If      the problem space is Constraints and
        the Constraints-state has a change record and
        the check operator for number-agreement on an assigner and receiver is being applied and
        the assigner and receiver profiles have a number value in common
Then mark the constraint as having passed and
        put the common number value(s) for the assigner and receiver on the change record
```

The word-order check makes certain that the syntactic subject precedes the verb. The well-formedness check makes certain that a full nounphrase is present; it passes here because the nounphrase is a proper name (well-formedness fails, for example, for a singular noun without a determiner). The number-agreement check detects that both *John* and *knows* have number 3s. Notice, however, that this production may restrict the values found on the number attribute in the profile of the receiver and assigner to the set that they have in common.

In d39, a semantic check is proposed to ensure that the referent of the receiver is a semantically appropriate actor for the act. This check operator is implemented in the Semantics space (d40-d42). Recall that proof of semantic validity can be achieved in either of two ways in the system. The first way is to find an instance of the situation being described already in the situation model (d43). In our example, if previous context had already introduced into the situation model an example of John knowing something, we would have the necessary evidence that John is a legitimate actor for a know-act. If the system cannot find the evidence, it must search for it by deliberate reasoning (d44). This process, not elaborated further in Figure 3-3, occurs in a new instance of the Constraints space in which check operators are proposed based on *know's* restrictions for the actor. In our example, the knowledge that John is a person (available from the situation model) combines with general knowledge that people are animate to justify the semantic consistency of the link.[11]

Having passed the syntactic and semantic constraints triggered by the subject link, the system creates a change record (d50) and moves on to the progress check (d51). As was the case with the first-word link, the subject link passes this check trivially (linking a new node to a connected structure leaves a connected structure). Thus, the constraint-success operator allows processing to return to the Language space (creating p186, analogous to p32 above) where a partial result can be returned to the Comprehension space (p187, analogous to p65 above). Chunk p187 is shown below. Note that its conditions consist of exactly those attributes and values— the word *knows*, the number, subclass, base-level-class, and receives for *John*—that were available prior to the impasse in d22 that were needed to pass *all* of the constraints on the link.

Chunk p187:
If the problem space is Comprehension and
the Comprehension-state has node p2 on the edge and
p2's profile has
number = 3s
subclass = proper
base-level-class = person
receives = subject and
the Comprehension-state has an incoming node n1 and
n1 corresponds to the word *knows* and
the comprehend operator is being applied to n1
Then add a change record to the comprehend operator that
links p2 to n1 as subject and
makes p2's profile
number = 3s
tense = present
assigns = subject & object & clausal-object
etc.

When the result is returned, the change record is immediately implemented by productions in the Comprehension space: the profile for *knows* is established, the new edge set is computed, and a subject link is created between *John* and *knows* in the utterance model. This information is then copied down to the Language-state where the unused referring information in the profile triggers the refer operator (d53). The refer operator creates an object for the act in the situation model (see Figure 3-1) and returns it as a partial result to Comprehension, thereby creating p202. To complete the deliberate comprehension of *knows*, another refer operator uses the maps attribute in *knows*' profile to establish the actor relation between *knows*' referent and *John*'s referent. This final change to the situation model is captured by p212 (d54).

Together p187, p202, and p212 contain all the changes to the models required by *knows* in the current context. Despite their similarity in being part of *knows*' comprehension operator, the three chunks nevertheless differ significantly in their generality, and, therefore, in their

---

[11]One could imagine extending the present Semantics space to incorporate episodic memory. In other words, the system might reason that *John* is a viable actor for some act not because it finds the act and actor already in the situation model, or because of general knowledge, but because the juxtaposition of act and actor *reminds* it of some episode in the past in which John performed that action [18, 19].

likelihood of transferring to other contexts. None of the productions relies on the appearance of the particular lexeme *John* in the subject position of the sentence, for example, but p187 does test for the lexeme *knows* in order to perform the lexical access. Chunk p202, on the other hand, does not test for *knows*, but does require that the verb's referring-information be a know-act. Thus, p202 will transfer to morphological variants of *knows* while p187 will not (depending upon the degree of semantic discrimination in the lexicon, p202 may also transfer to related verbs, for example, *thinks*). Chunk p212 is more general still, testing for none of *John*, *knows* or referring-information as a know-act, and relying only on the existence of a subject link in the utterance model between two nodes that already have referents. Through comprehending *John knows*, then, we have added to the system recognitional knowledge that contributes to understanding entirely novel utterances.

Figure 3-4 shows deliberate comprehension of the last word in our first example sentence. The dynamic for *Sharyn* is similar to that for *knows* and *John*. The comprehend operator impasses, resulting in lexical access, linking, and referent resolution in the Language space. *Sharyn*'s lexical definition (d60) is analogous to *John*'s. The proposed link operator (d61) for the object role must pass semantic and syntactic checks similar to those passed by the subject link (d63-d72). When the constraints have been passed, the change record is returned as an intermediate result (d78-d80), producing p296 (analogous to p187). To finish the process, a referent for *Sharyn* is created (d81, p320) and linked to the referent for *knows* with an object relation (d82, p328). The utterance and situation models now look like those in Figure 3-1. The comprehension operator for *Sharyn* is now partially defined by the chunks p296, p320, and p328.

At the beginning of this section, in Figure 3-1, we showed what we would expect NL-Soar's processing to look like if the system could recognitionally comprehend our example sentence. We then assumed that the system did not have the appropriate knowledge immediately available in the Comprehension space, and demonstrated how deliberate comprehension would proceed for each word in turn (Figures 3-2 (*John*), 3-3 (*knows*), and 3-4 (*Sharyn*)). In each case, there were two distinct aspects to deliberate comprehension. First, search over multiple knowledge sources in NL-Soar's lower problem spaces produced additions to the models that described the structure and the meaning of the sentence. Second, Soar's general learning mechanism integrated the knowledge from the disparate sources into a form that could be immediately brought to bear with a single comprehend operator. Figure 3-5 demonstrates that through this compilation and integration process, the system has achieved the expected behavior.

3.2. Example 2: Learning top-down knowledge in the comprehension operator

The lexical-access, link, and refer operators were all that was needed in the Language space to deliberately construct the utterance and situation models for the sentence *John knows Sharyn*. But these operators are not always adequate for deliberate comprehension. To see why not, recall that link operators are proposed only when an (assigner, receiver, role) triple can be produced using the profile for a node on the edge set and a profile for the word being comprehended. Now consider the sentence *Sharyn knows a chemist*. When processing reaches the word *a*, which receives only a determiner role, the only node that is on the edge represents *knows*, which does not assign a determiner. Consequently, no link operator can be proposed. The problem is that link, as we have defined it, provides only word-driven or bottom-up knowledge during comprehension. Contexts such as the one that arises for *a*, on the other hand, require expectation-driven or top-down processing if each word is to be interpreted immediately. In this section we

```
...55   O: attend(Sharyn)
   56   O: comprehend(Sharyn)
   57   ==>G: operator no-change
   58         P: Language
   59         S: Language-state
   60         O: lexical-access(Sharyn)
   61         O: link(knows, Sharyn, object)
   62         ==>G: operator no-change
   63               P: Constraints
   64               S: Constraints-state
   65               O: check(word-order)
   66               O: check(np-well-formedness)
   67               O: check(semantic-consistency)
   68               ==>G: operator no-change
   69                     P: Semantics
   70                     S: Semantics-state
   71                     O: find
   72                     O: justify
...78               O: record-changes
   79               O: check(progress)
   80               O: constraint-success
   Build: p295
   Build: p296
   81         O: refer(Sharyn)
   Build: p320
   82         O: refer(knows)
   Build: p328
   83   O: attend(.)
   84   O: comprehend(.)
```

Figure 3-4: Deliberate comprehension of *Sharyn* in *John knows Sharyn*.

show how this top-down processing arises naturally in NL-Soar by examining the behavior of the Language space's expect and merge operators during comprehension of *Sharyn knows a chemist*. To simplify the exposition, we assume that the system now contains the chunks built during comprehension of *John knows Sharyn*.

Figure 3-6 shows processing for the first two words. Although the word *Sharyn* was seen in the sentence from the previous example, it was not encountered as the first word in the sentence. Consequently, part of the current comprehension operator for *Sharyn* (specifically chunk p296 which establishes the change record) is inappropriate to this context. Consequently, an impasse arises (d9) and processing continues in the Language space. Although there was no transfer in the Comprehension space, there may, nevertheless, be transfer in the lower problem spaces. Indeed, p32 is a chunk accessible in Language that resulted from problem solving in the Constraint space for *John* as the first word. Since the conditions for that link apply in the current situation, the chunk transfers, creating the correct change record immediately (contrast d13-d19

0 G: g1
1 P: Top
2 S: Top-state

Type your input > *John knows Sharyn.*

3 O: comprehend-input
4 ==>G: operator no-change
5 P: Comprehension
6 S: Comprehension-state
7 O: attend(*John*)
8 O: comprehend(*John*)
Firing: p65, p76
9 O: attend(*knows*)
10 O: comprehend(*knows*)
Firing: p187, p202, p212
11 O: attend(*Sharyn*)
12 O: comprehend(*Sharyn*)
Firing: p296, p320, p328
13 O: attend(.)
14 O: comprehend(.)

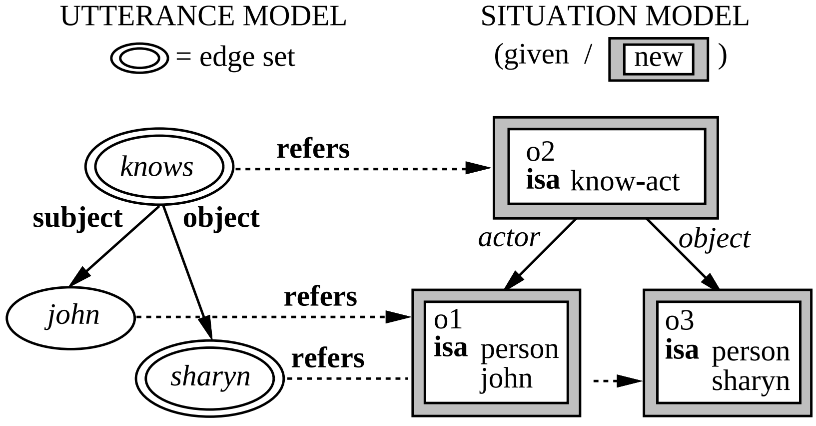

Figure 3-5: Recognitional comprehension of *John knows Sharyn* after chunking.

in Figure 3-2 with d13 in 3-6). The change record is then returned as a partial result, creating p330, and extending *Sharyn*'s comprehension operator to the current context. Since p330's change record is immediately instantiated in the Comprehension-state, a piece of unused referring information appears in the node's profile. This triggers transfer of chunk p320, the portion of *Sharyn*'s comprehension operator that creates a referent in the situation model. Note that this transfer is in the Comprehension space, bypassing the need for a refer operator to follow the link in Language, and completing the comprehension of *Sharyn*. *Knows* is then attended to and its comprehension operator fires in full. The mixture of deliberate and recognitional comprehension seen in processing these two words is normal in NL-Soar. Knowledge acquired from other contexts (even for other words) may transfer to the current context, permitting completely recognitional comprehension. If not, deliberate search occurs automatically to fill in those parts of a word's comprehension operator that are new to the context. The deliberate search may also involve a combination of chunked behavior and search in subspaces.

Figure 3-7 shows the utterance and situation models after the word *a* has been attended to and picks up the trace at this same point, d17.[12] The usual steps follow through d21, at which point no link operator can be proposed. When no link is possible, expect is proposed. When implemented, expect simply creates an empty node in the utterance model and causes adjustments to the edge set. Like link, the implementation of expect is mediated by constraints (d23-d26). Unlike link, however, there are no syntactic or semantic constraints associated with

[12]We have elided many steps (some of which result in chunks) in Figure 3-7 to make the overall structure of the processing clearer.

```
0 G: g1
1 P: Top
2 S: Top-state
   Type your input > Sharyn knows a chemist.
3 O: comprehend-input
4 ==>G: operator no-change
5        P: Comprehension
6        S: Comprehension-state
7        O: attend(Sharyn)
8        O: comprehend(Sharyn)
9        ==>G: operator no-change
10             P: Language
11             S: Language-state
12             O: lexical-access(Sharyn)
13             O: link(Sharyn, first-word)
Firing: p32
Build: p330
Firing: p320
14       O: attend(knows)
15       O: comprehend(knows)
Firing: p187, p202, p212
16       O: attend(a)
```

Figure 3-6: Mixed deliberate/recognitional comprehension of *Sharyn knows* in *Sharyn knows a chemist.*

the creation of an expectation object, so only the progress constraint is relevant (d27). For each previous word we have examined, the progress constraint was passed trivially. This was because the notion of progress exists in the system to ensure that a connected utterance model (and, thus a well-formed situation model) is maintained; the previous instances of link maintained connectedness. This is not the case with expect—the simple creation of a new node does not connect either that node or the current word (*a*) into the utterance model. In order to pass the progress constraint, then, the system must answer the question, ‘‘Does positing the existence of some future word allow us to create a cohesive model?’’ NL-Soar answers this question by using Soar’s lookahead processing capability to search ahead in a copy of the Language space. This lookahead search is *not* a lookahead in the input; rather, it is a lookahead in the space of utterance and situation models that can result from creating the expectation.

The remaining decision cycles in Figure 3-7 demonstrate how the lookahead search progresses. The impasse on expect’s progress (d28) results in the creation of a new Language space which, for clarity of exposition, we name Language$_1$ (d29 and d30). Language$_1$ is an exact copy of the Language space in d20 with a single exception: the expect operator now has an annotation on it that allows the system to simply assume expect’s progress by creating the expectation node (d31). Once this new node is part of Language-state$_1$, a link operator is proposed to link *a* to the expectation. The processing in d33 through d45 is, therefore, simply normal constraint checking

```
...17  O: comprehend(a)
   18  ==>G: operator no-change
   19      P: Language
   20      S: Language-state
   21      O: lexical-access(a)
   22      O: expect
   23      ==>G: operator no-change
   24          P: Constraints
   25          S: Constraints-state
   26          O: record-changes
   27          O: check(progress)
   28          ==>G: operator no-change
   29              P: Language_1
   30              S: Language-state_1
   31              O: expect(just-do-it)
   32              O: link(a, *exp*, determiner)
   33              ==>G: operator no-change
   34                  P: Constraints_1
   35                  S: Constraints-state_1
...37                  O: check(determiner-np-agreement)
...44                  O: record-changes
   45                  O: check(progress)
   46                  ==>G: operator no-change
   47                      P: Language_2
   48                      S: Language-state_2
   49                      O: link(a, *exp*, determiner, just-do-it)
   50                      O: refer(*exp*)
   51                      O: link(knows, *exp*, object)
   52                      ==>G: operator no-change
   53                          P: Constraints_2
   54                          S: Constraints-space_2
...68                          O: record-changes
   69                          O: check(progress)
   70                          O: constraint-success
```

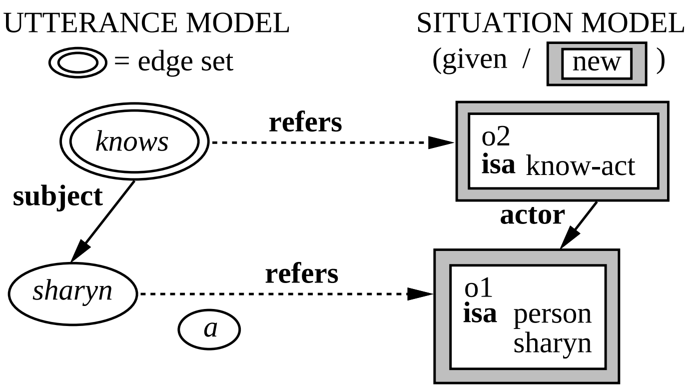

Figure 3-7: Models after attending to *a* and lookahead search during comprehension of *a* in *Sharyn knows a chemist*.

on the proposed link. In other words, there is nothing special about Language$_1$—it is just the same old Language space. In it, NL-Soar makes a change to a local copy of the models by creating the expectation. Then, the same type of processing used in the previous example plays out the consequences of that change.

Figure 3-8 shows the utterance and situation models in Language$_1$ after the link between *a* and the expectation has been made. Notice that the expectation object is no longer empty. As a result of the syntactic constraints brought to bear by the link (d37), the system now knows that the word that this expectation is holding a place for must have number 3s. The expectation includes other constraining information from other checks, as well. There is no special mechanism at work here—each bit of constraining information is added to the expectation's profile in the same way that the feature sets for profiles of regular words are dynamically constrained.

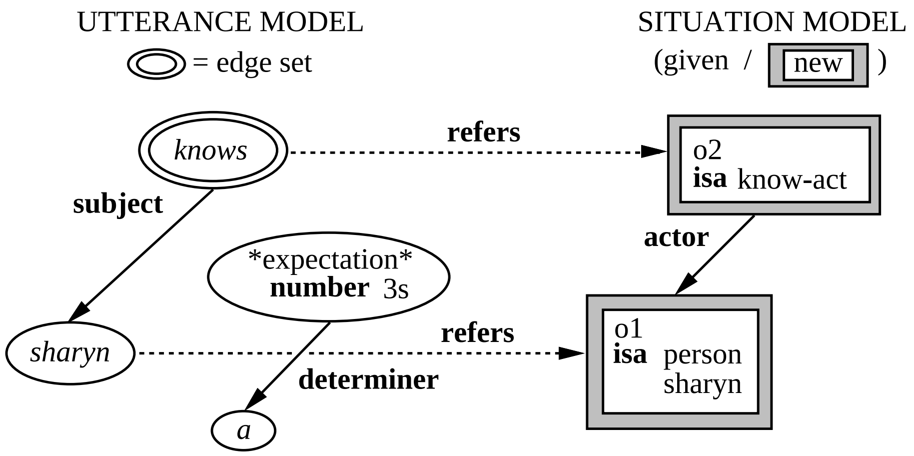

Figure 3-8: Hypothetical model change during lookahead search for *Sharyn knows a*.

Since the expectation node is still not linked into the utterance model, the progress check at d45 cannot be passed and the lookahead search must continue. The second step in lookahead (d46 through d69) is analogous to the first step. A copy of the Language_1 space is created (d47) that is identical to its predecessor except that it is assumed that the link between *a* and the expectation makes progress. Again, we index this new Language space (as Language_2) for clarity. The link is instantiated using the change record created prior to the progress check (d44). Next, a refer operator establishes a referent for the expectation in the situation model (d50). In d51, a link operator is proposed to connect the expectation to *knows* via an object role. As always, the proposed link must pass all constraints (d52 through d69). This time, however, we find that all constraints pass, including progress, bringing an end to the lookahead search.

To understand the system's behavior in Figure 3-7, we must examine it at both global and local levels. At the global level, the system still has made no changes to the utterance and situation models in the Language space. It has only found a sequence of operators (expect, link, refer, and link) which, if applied, will result in model changes that leave the models well-connected. At the local level, however, the system has actually done the work of applying those operators to transform the models, albeit on private copies. Further, the application of each operator acted to constrain the features of the expectation and, hence, the future word in the sentence that could play the appropriate roles. So, depending on the point of view, NL-Soar has done both a great deal of work and almost no work at all. Figure 3-9 demonstrates how the recursive use of the Language and Constraints problem spaces during lookahead combines with chunking to automatically transfer work from the local to the global level. We repeat the relevant portions of Figure 3-7 in Figure 3-9 to make the unwinding of the goal hierarchy easier to follow.

We begin in Figure 3-9 where we left off in Figure 3-7. In d70, all the constraints on linking *knows* to the expectation have passed (including progress). As a result, the change record that implements that link is returned to Language_2, and p444 is created. Remember that our use of the index is just an expository convenience—the chunk, itself, tests for the token ‘‘language’’ in the problem space name. The implementation of the link in Language_2 resolves the progress impasse in d45, creating p445. The success of progress, in turn, signals total constraint success in Constraints_1 (d71).

```
...19   P: Language
...22   O: expect
  23    ==>G: operator no-change
  24       P: Constraints
...27      O: check(progress)
  28       ==>G: operator no-change
  29          P: Language_1
...31         O: expect(just-do-it)
  32          O: link(a, *exp*, determiner)
  33          ==>G: operator no-change
  34             P: Constraints_1
...45            O: check(progress)
  46             ==>G: operator no-change
  47                P: Language_2
...51               O: link(knows, *exp*, object)
...52               ==>G: operator no-change
  53                   P: Constraints_2
...70                  O: constraint-success
  Build: p444
  Build: p445
  71             O: constraint-success
  Build: p446
  Build: p447
  72       O: constraint-success
  Build: p448
  Build: p449
  73    O: link(a, *exp*, determiner)
  Firing: p446
  Build: p474
  74    O: refer(*exp*)
  Build: p500
  75    O: link(knows, *exp*, object)
  Firing: p444
  Build: p504
  Firing: p328
  76  O: attend(chemist)
```

Figure 3-9: Unwinding the lookahead search.

Now the pattern repeats. The constraint success in d71 returns a change record implementing the link between *a* and the expectation to Language_1, creating p446. Again, the chunk refers to the token ‘‘language’’ not ‘‘language_1’’. The implementation of the link resolves the progress impasse in d27, creating p447. The success of progress signals total constraint success in Constraints (d72).

The constraint success in d72 returns a change record to the Language space that implements the original expect operator, creating p448. Language then returns the change record as a partial result to the Comprehension space to form part of the comprehension operator for *a* (and p449). This part of *a*'s comprehension operator creates an empty expectation node in the utterance model. When the new node is copied down to Language, the Language-state becomes exactly the same as it was in Language_1 at d31 (when the system had just assumed that positing the expectation would make progress). Consequently, a link between *a* and the expectation is proposed in d73 just as it was in d32. At d73, however, knowledge is available that was not available at d32. Specifically, long-term memory now contains p446 which implements the link between *a* and the expectation. Since the conditions in Language are identical to those that gave rise to chunk p446, the chunk fires. In this way, the change record linking *a* to the expectation appears on the Language-state where it is returned as a partial result to Comprehension, creating p474 and adding another piece to *a*'s comprehension operator. Chunk p474 is shown below:

```
Chunk p474:
If     the problem space is Comprehension and
       the Comprehension-state has
            node p1 whose word is to the left of
            node n1 whose word is to the left of
            node e1 and
       the comprehension operator is being applied to n1 and
       n1 corresponds to the word a and
       e1 is an expectation and
       p1 assigns the object role restricted to isa thing
Then add a change record to the comprehend operator that
       links n1 to e1 with the determiner role and
       makes e1's profile
            number = 3s
            assigns = describer & qualifier
            receives = subject & object
            most-specific-class = thing and
       makes n1's profile
            number = 3s
            subclass = indefinite & demonstrative
            receives = determiner
            referring-information = isa thing
```

The chunk encapsulates all the conditions on the link's path to progress (the existence of the expectation, the determiner, and the verb without an object) and preserves the constraints imposed by that path (the number and most-specific-class of the expectation).

Having changed the utterance model to reflect the link to *a*, the system must perform referent resolution on the expectation in the Language space. This happens in d74, resulting in p500. Once the expectation has a referent the system is back in familiar territory: the current state of the Language space (d75) now looks like it did in d51. Consequently chunk p444 transfers, creating the change record for the link between *knows* and the expectation in Language. The partial result is returned, building p504 in Comprehension. The instantiation of the change record in Comprehension triggers p328, a very general referent resolution chunk that transfers from the sentence *John knows Sharyn* to create the object relation in the situation model. Figure 3-10 shows the utterance and situation models now that comprehension of *a* is complete.

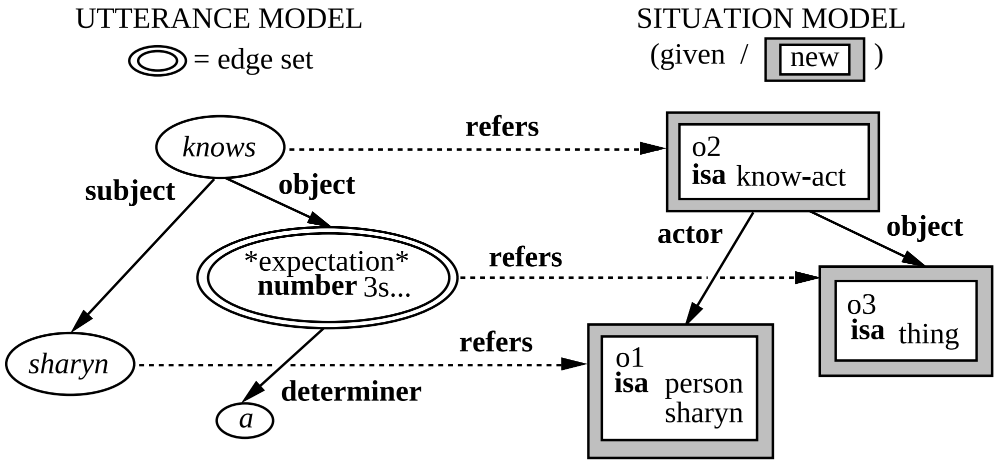

Figure 3-10: The models after comprehension of *a* in *Sharyn knows a chemist*.

Before moving onto the last word in the sentence (*chemist*), let's examine, again, what has happened from global and local points of view. In the global view, the system found an interpretation for the current word (*a*) by positing the existence of a future word in the sentence that would link to both *a* and the utterance model created for *Sharyn knows*. It did this by searching through the set of models permitted by the syntactic, semantic, and pragmatic knowledge in the system. Along the way, it collected a number of feature restrictions that the future word must meet. In the local view, the operators the system used at each step in this search were just the normal Language and Constraints operators. Thus, the work that was done at the local level was no different than the work that was done in the previous sentence. But at the global level the system appears to have done something very different from the simple linking in the previous example. The glue that holds these two views together is the notion of progress: each step in the lookahead search was in the service of the higher goal of finding a cohesive pair of structures in which *a* was connected to the utterance model. No actual changes were made until it was certain that such a cohesion was possible. Once it was proven that comprehension could make progress by positing an expectation, the actual changes came through chunk transfer. If we pull back further, from deliberate comprehension to recognitional comprehension, this global/local distinction disappears completely. Viewed from the Comprehension space, all the search just results in the usual specification of a comprehension operator for the current word and context that simultaneously brings to bear all the relevant knowledge sources and makes all the relevant model changes.

Our analysis of the comprehension of *a* introduced the expect operator. An analysis of the comprehension of *chemist* introduces expect's counterpart: merge. The merge operator recognizes the appearance of the expected word in the input and replaces the expectation object with a regular utterance node for that word. As shown in Figure 3-11, d82, the merge operator is proposed when no link is available and there is an expectation object in the edge set.[13] As with link and expect, before merge can be implemented, it must satisfy a variety of constraints. A syntactic consistency check (d86) ensures that the syntactic features in *chemist*'s lexical definition are consistent with links that have been made for the expected word. In our example, this means that the word's number must be 3s to maintain the validity of the link to the determiner *a*. The simple token match used for establishing syntactic consistency is inadequate to ensure semantic consistency, however. Instead, a regular semantic check must be done (d87 through d98), which includes subgoaling into the Semantics space to justify semantic consistency through simple inference. In our example, the Semantics space provides the knowledge that a word with base-level-class person is consistent with the expectation that the most-specific-class be a thing.

```
...76      O: attend(chemist)
  77       O: comprehend(chemist)
  78       ==>G: operator no-change
  79          P: Language
  80          S: Language-state
  81          O: lexical-access(chemist)
  82          O: merge
  83          ==>G: operator no-change
  84             P: Constraints
  85             S: Constraints-state
  86             O: check(match-syntactic-consistency)
  87             O: check(semantic-consistency)
...98            O: record-changes
  99             O: check(progress)
  100            O: constraint-success
  Build: p564
  Build: p565
  101         O: refer(chemist)
...105     O: attend(.)
```

Figure 3-11: Merging *chemist* with the expectation.

Since the profile for *chemist* is consistent with the restrictions placed on the word that can fulfill the expectation, the change record created in d98 simply replaces the expectation with the current word. The progress check passes trivially (because the expectation was already fully-connected to the model) resulting in the Language chunk (p564) and then the Comprehension chunk (p565). Deliberate comprehension of *chemist* finishes in the Language space after a number of refer operators establish the relevant changes to the situation model. Figure 3-12 shows the final forms of both the utterance and situation models for the sentence.

[13]It is not necessary that the expected word be the next one in the sentence. If, for example, our sentence had been *Sharyn knows the taller chemist*, the word *taller* would have been attached to the expectation as a describer, changing the most-specific-class on the expectation's profile from thing to physical-object. Again, there is no special mechanism for handling expectations; the dynamic constraint of features is part of NL-Soar's normal processing.

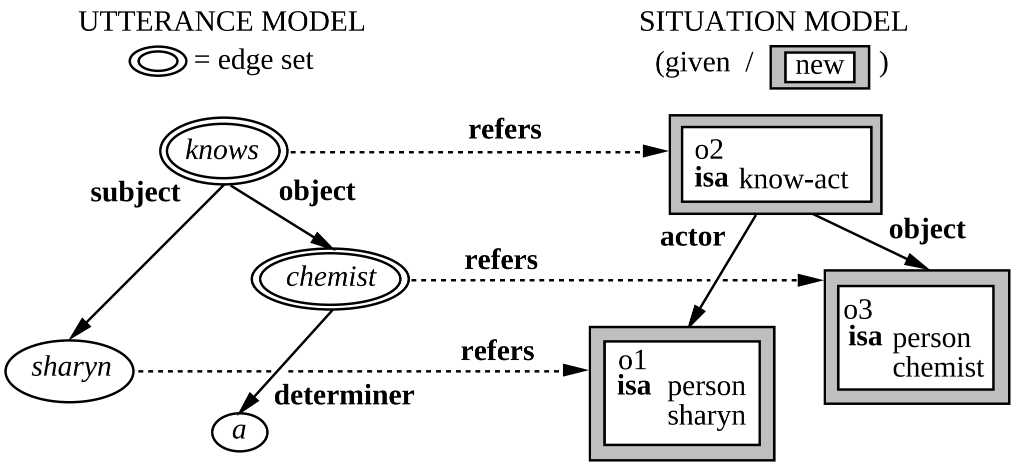

Figure 3-12: The models constructed during comprehension of *Sharyn knows a chemist*.

In this section, we examined in detail two operators in the Language space: expect and merge. These operators give NL-Soar the capability of using top-down, or expectation-driven, knowledge during both deliberate and recognitional comprehension. This additional capability does not arise from adding new linguistic knowledge to the system, however. The expect operator is simply proposed when no link is possible. The merge operator is proposed when no link is possible and there is an expectation node in the edge set. Both operators are implemented reusing the same problem spaces, operators, and knowledge that already existed for link. Rather, the additional capability arises because of the notion of progress and the use of multiple problem spaces in a lookahead search. Thus, in general, deliberate comprehension can involve the combination of bottom-up and top-down application of multiple knowledge sources. No matter which knowledge sources and which type of search is required, however, the result is still the integration of those sequential actions into a set of productions triggered by a single operator in Comprehension.

3.3. Example 3: Recognitional repair

Having added expect and merge to lexical-access, link, and refer, it might appear that the system now has all the functional capability it needs. As we pointed out in Section 2.2, however, NL-Soar is a single-path parser, constructing only one interpretation of the sentence at a time. Using both bottom-up and top-down knowledge cannot guarantee that the current interpretation is the correct one. Consider the sentence, *John knows Sharyn knows a chemist*. From our first example (Section 3.1) we know that when processing reaches *Sharyn* it will be assigned the object role of *knows*. This is incorrect; the correct interpretation assigns *Sharyn* as the subject of the second *knows* and the whole clause *Sharyn knows a chemist* as the object of the first *knows*. Yet the incorrect assignment cannot be detected until the next word is encountered (the second *knows*), after the object link between *knows* and *Sharyn* has already been made. Thus, what NL-Soar still needs is a way to undo an incorrect interpretation. The snip operator provides this capability. The snip operator itself contains no special knowledge; it acts only as a disrupting influence, removing an existing link. The actual repair of the models must be performed by Language's other operators. In this section, we examine the behavior of snip in the sentence

John knows Sharyn knows a chemist assuming the existence of those chunks produced during the previous two examples. For clarity, in the figures and discussion that follow, we index the two instances of *knows* as knows_m, for main, and knows_c, for clause.

Figure 3-13 (d0 through d12) takes us quickly through *John knows Sharyn*, via chunks that transfer from the example in Section 3.1. In d14, we encounter the second instance of *knows*. Although the verb is familiar, its current comprehension operator is inappropriate for this context, so an impasse arises. As the first step in Language (d18), the lexical-access operator finds the same definition for the verb shown in Section 3.1. A link is then proposed to treat the main verb as the clausal object of the embedded verb (d19).[14]. The link fails in the Constraints space due to a word-order violation. Since no other link is proposed, the expect operator is proposed. What follows in the 129 decision cycles ellided from d24 to d153 is the lookahead search that uncovers that positing an expectation node will not allow the current interpretation to make progress. Thus, at d153, the snip operator is proposed to remove the object link that exists between *Sharyn* and $\textit{knows}_m$.

Why is the object link snipped? How does the system know some other link in the utterance model is not the source of the error? It doesn't. The object link is snipped because it is the only link connected to a node in the edge set (see Figure 3-1). A snip operator is proposed for every link connected to a node in the edge set. Had there been more than one such link, snipping each would have been tried in turn until a snip was found that allowed the re-establishment of a single, coherent pair of models. The choice to limit the snip operator to links on the edge is motivated by psycholinguistic data on garden path phenomena [4, 7, 9, 10, 36, 49]. The data indicate that there is a class of sentences that people find impossible to comprehend recognitionally. The classic example of a garden path sentence is *The horse raced past the barn fell* [4]. At *raced*, people almost invariably commit to the single interpretation of an active, past-tense verb rather than the beginning of a reduced relative clause. When they reach *fell*, which reveals the error of their initial interpretation, they cannot recover, feeling instead that the sentence makes no sense. Such sentences stand in contrast to our current example, *John knows Mary knows a chemist*, in which an initial wrong commitment seems to be overcome without pause when the disambiguating word is encountered. Preliminary evidence in studying garden path and non-garden path sentences supports the idea of limiting any repair mechanism to the active edge. Thus, the model of recognitional comprehension in NL-Soar makes a prediction: non-garden path sentences cause no problem for people because they allow for recognitional repair via chunking over a snip operator at the edge, while garden path sentences preclude such a repair.

We said that when link and expect fail, a snip operator is tried for every link in the active edge set until one is found that allows the re-establishment of a single, cohesive pair of models. The latter condition is met by the same interaction of the progress check and lookahead search we encountered with expect. Figure 3-14 illustrates. In response to an impasse on the progress check for performing the snip (d158), a copy of the Language space is created in which a snip is implemented just to see what will happen (d162). This leads to a link proposal connecting

---

[14]The reader who paused to look back at the lexical definition of *knows* will note that *knows* does not appear to receive a clausal-object. The clausal-object value is added to the verb's receives feature dynamically, when the verb is linked to its subject. This is an example of how NL-Soar handles exocentric constructions

0 G: g1
1 P: Top
2 S: Top-state

Type your input > *John knows Sharyn knows the chemist.*

3 O: comprehend-input
4 ==>G: operator no-change
5 P: Comprehension
6 S: Comprehension-state
7 O: attend(*John*)
8 O: comprehend(*John*)
Firing: p65,p76
9 O: attend($knows_m$)
10 O: comprehend($knows_m$)
Firing: p187,p202,p212
11 O: attend(*Sharyn*)
12 O: comprehend(*Sharyn*)
Firing: p296,p320,p328
13 O: attend($knows_c$)
14 O: comprehend($knows_c$)
15 ==>G: operator no-change
16 P: Language
17 S: Language-state
18 O: lexical-access($knows_c$)
19 O: link($knows_c$, $knows_m$, clausal-object)
...24 O: expect
...153 O: snip($knows_m$, *Sharyn*, object)

Figure 3-13: Discovering the need to undo a previous commitment in *John knows Sharyn knows*.

$knows_c$ to *Sharyn* as subject in d164. Progress for this link must then be established (d173), resulting in a copy of the $Language_1$ space in which the link is asserted in order to see what follows (d174-177). What follows is another link proposal, this time assigning the clausal-object role from $knows_m$ to $knows_c$. Notice that snip only undid a link; the relinking occurred via operators we have already encountered.

Processing in d184 through d204 is exactly like the unwinding of the goal stack we saw in the previous example—as we pop out of each Constraints space, chunks are built that transfer to implement the successful sequence in the original Language space (d206 through d212). The progression of operators executed in Language, as well as the final linked utterance model for *John knows Sharyn knows*, is show graphically in Figure 3-15. The productions that are returned to Comprehension (p1425, which contain the change record for the snip; p1447, which contains the change record for the subject link; and p1514, which contain the change record for the clausal-object link), all become part of the comprehension operator for *knows* in the current context. The remainder of the sentence *(the chemist)* is comprehended recognitionally through the firing of chunks transferring from Example 2.

...153 O: snip(*knows*$_m$, *Sharyn*, object)
154 ==>G: operator no-change
155 P: Constraints
156 P: Constraints-state
157 O: record-changes
158 O: check(progress)
159 ==>G: operator no-change
160 P: Language$_1$
161 P: Language-state$_1$
162 O: snip(*knows*$_m$, *Sharyn*, object, just-do-it)
163 O: refer(*Sharyn*)
164 O: link(*knows*$_c$, *Sharyn*, subject)
165 ==>G: operator no-change
166 P: Constraints$_1$
...173 O: check(progress)
174 ==>G: operator no-change
175 P: Language$_2$
...177 O: link(*knows*$_c$, *Sharyn*, subject, just-do-it)
...183 O: link(*knows*$_m$, *knows*$_c$, clausal-object)
Build: p1133
203 O: constraint-success
Build: p1421
Build: p1422
204 O: constraint-success
...Build: p1424
Build: p1425
Firing: p346
...206 O: link(*knows*$_c$, *Sharyn*, subject)
...Firing: p1421
Build: p1447
Firing: p202,p212
...210 O: link(*knows*$_m$, *knows*$_c$, clausal-object)
...Firing: p1133
Build: p1513
...212 O: attend(*the*)

Figure 3-14: Finding a new interpretation for *John knows Sharyn knows*.

Having chosen to pursue only a single interpretation of the sentence at a time, it was necessary that the system be given some way to undo a wrong commitment. In this section, we examined the snip operator, which affords the system a limited capability for repair. The capability is limited for two reasons. First, it is limited because snip, like expect and merge, introduces no additional linguistic knowledge—the operator can only disrupt the current model's structure which must be repaired by the same linguistic knowledge available during normal interpretation.

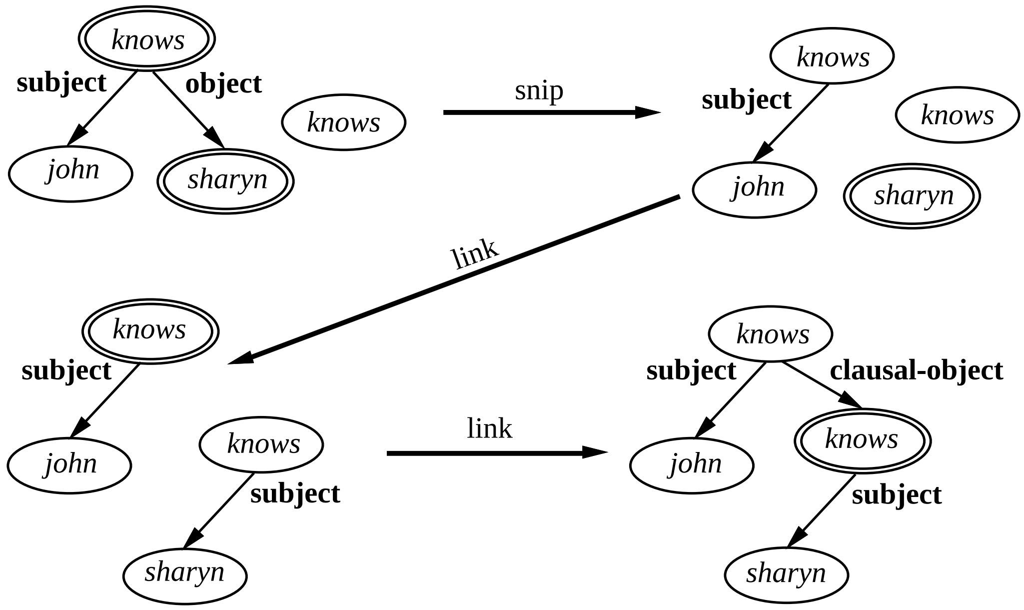

Figure 3-15: The progression of Language operators found through lookahead search to reinterpret *John knows Sharyn* given the next word *knows*.

Second, the repair capability is limited because only links in the edge set may be undone.[15] The snip operator is also like expect and merge in another respect: even if deliberate comprehension involves undoing an old link and creating one or more new ones through the application of multiple knowledge sources, the processing that results in Comprehension is still recognitional in nature.

4. Conclusion

Our purpose in this report is to describe NL-Soar in sufficient detail that its characteristics can be compared and contrasted with both prior Soar versions and non-Soar language comprehension systems. Since the current system has not yet achieved our goal of providing a general language capability for Soar, it is reasonable to assume that extensions in the form of new problem spaces, operators, and models will be needed to meet the demands of increasing capabilities. Nevertheless, the current system achieves a basic functionality with respect to comprehension, and from our description and examples, the following encapsulation is possible:

1. **Comprehension operators assign meaning.** Like previous versions of the system, the current NL-Soar uses comprehension operators as its method of mapping language to meaning. A comprehension operator is a set of productions that, together, tell the system

---

[15]Another way to look at NL-Soar's repair mechanism is as an alternative to backtracking in a single-path system. Because the system does not implement backtracking, it does not need to store a large history of possible alternative imterpretations in the utterance model. If the supposition proves true that limiting the snip operator to the edge set eliminates only those utterances people are unable to comprehend recognitionally, then this form of repair may explain how people can both maintain a single, favored interpretation and still recognitionally change that interpretation despite their memory limitations.

what changes to make to the representation of the utterance when a word is encountered in any context. Moreover, comprehension operators take into account, simultaneously, all the knowledge sources that contribute to determining meaning. While the use of productions places NL-Soar in the company of other situation-action parsers, the requirement of total integration of knowledge sources goes beyond previous attempts at integration. The nature of the processing that results from comprehension operators is similar in many respects to adult recognitional comprehension.

2. **Comprehension operators arise automatically.** Integrated comprehension operators are produced automatically by the constant conversion of knowledge from a form that is accessible only through deliberate problem solving to a form that is immediately accessible from production memory (i.e., recognitionally). This incremental conversion process relies on the chunking mechanism of the Soar architecture. It is unique to NL-Soar.

3. **Annotated model represents meaning.** NL-Soar uses an annotated model to represent the situation described by an utterance. Although not as restrictive as a pure model representation, annotated models are, nevertheless, less expressive than first-order logic. Thus, NL-Soar is more constrained in its representation of meaning than systems that use a logic-based formalism. This source of constraint seems to have some empirical evidence in its support [16, 33, 34].

4. **Annotated model represents utterance structure.** NL-Soar also uses an annotated model to represent the structure of the utterance. The utterance model does maintain a strict one-to-one correspondence between words and nodes, unlike phrase-structure based systems. Such a linear relation between the length of the input and the size of the model improves processing efficiency.

5. **System is single-path.** Unlike all-paths parsers, NL-Soar keeps only a single interpretation of the utterance at any time. In addition to being more efficient than all-paths parsers, single-path parsers seem to be a better characterization of human comprehension as well [7, 9, 36, 49]. On the other hand, single-path parsers have the disadvantage that they cannot guarantee the correct interpretation of an utterance, only an interpretation consistent with the knowledge available (see also 7, below).

6. **System combines bottom-up and top-down knowledge.** Processing in NL-Soar takes advantage of both bottom-up, or word-driven, knowledge and top-down, or expectation-driven knowledge. The combination of processing techniques is generally accepted to be more efficient than either technique alone. Which type of knowledge NL-Soar uses at any given point in deliberate comprehension arises naturally from the system’s notion of progress. Regardless of which type is used, however, the knowledge that results retains its recognitional character.

7. **System has a limited capacity for repair.** To try to ensure that its single interpretation is consistent with the knowledge available, NL-Soar has a limited capacity for repairing incorrect interpretations. This limited repair is not a general backtracking mechanism, and can undo commitments only at the active edge. The decision to limit repair is motivated by psycholinguistic evidence of such limitations in adult recognitional behavior. As in the case of mixed bottom-up and top-down processing, repair during deliberate comprehension does not change the recognitional character of NL-Soar at the comprehension operator level.

The automatic construction of comprehension operators from extendable and disparate knowledge sources addresses a long-standing dichotomy between approaches that favor separate phases of comprehension (usually progressing from morphology to syntax to semantics and beyond) and those that favor integration. The distinction between deliberate and recognitional comprehension in NL-Soar is evidence that a modular approach can co-exist with and evolve into an integrated one. Thus, the dynamic of the system offers a possible bridge between the autonomy-of-syntax hypothesis (and by extension, autonomy-of-semantics, etc.) and the notion of integration that usually accompanies the immediacy-of-interpretation principle. Each assumption is represented, but at different levels in the processing. Although we have not answered the genesis question (*Where does the knowledge in the lower problem spaces come from?*), NL-Soar does demonstrate how knowledge of different types could be acquired incrementally and opportunistically and still result, asymptotically, in essentially seamless performance.

References

1. Allen, J., *Natural Language Understanding,* Benjamin/Cummings, Menlo Park, CA, 1987.

2. Anderson, J. R., *The Architecture of Cognition,* Harvard University Press, Cambridge, Massachusetts, 1983.

3. Bateman, J. A., Kasper, R. T., Moore, J. D., and Whitney, R. A., ‘‘A general organization of knowledge for natural language processing: The Penman Upper Model,’’ Tech. report, USC/Information Sciences Institute, March 1990.

4. Bever, T. G., ‘‘The cognitive basis for linguistic structures,’’ in *Cognition and the Development of Language,* Hayes, J. R., ed., Wiley, New York, 1970.

5. Birnbaum, L. and Selfridge, M., ‘‘Conceptual analysis of natural language,’’ in *Inside Computer Understanding,* Schank, R. C. and Riesbeck, C. K., eds., Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1981.

6. Burton, R. R. and Brown, J. S., ‘‘Toward a natural-language capability for computer-assisted instruction,’’ in *Procedures for Instructional Systems Development,* O’Neill, H., ed., Academic Press, New York, 1979.

7. Crain, S. and Steedman, M., ‘‘On not being led up the garden path: the use of context by the psychological syntax processor,’’ in *Natural Language Parsing,* Dowty, D. R. and Karttunen, L. and Zwicky, A. M., eds., Cambridge University Press, Cambridge, U.K., 1985.

8. Dyer, M. G., *In-Depth Understanding,* MIT Press, Cambridge, Massachusetts, 1983.

9. Frazier, L. and Rayner, K., ‘‘Making and correcting errors during sentence comprehension: Eye movements in the analysis of structurally ambiguous sentences,’’ *Cognitive Psychology,* Vol. 14, 1982, pp. 178-210.

10. Gibson, E., ‘‘Recency preference and garden-path effects,’’ *Twelfth Annual Conference of the Cognitive Science Society,* 1990, pp. 372-379.

11. Hauptmann, A. G., Young, S. R., and Ward, W. H., ‘‘Using dialog-level knowledge sources to improve speech recognition,’’ *Proceedings of the Seventh National Conference on Artificial Intelligence, American Association for Artificial Intelligence,* 1988, pp. 729-733.

12. Hays, D. G., ‘‘Dependency theory: a formalism and some observations,’’ *Language,* Vol. 40, No. 4, 1964, pp. 511-525.

13. Hendrix, G. G., ‘‘LIFER: A natural language interface facility,’’ *SIGART Newsletter,* February 1977, pp. 25-26.

14. Hudson, R., *Word Grammar,* Basil Blackwell, Oxford, England, 1984.

15. Huffman, S. B., ‘‘A Natural-language system for interaction with problem-solving domains: Extensions to NL-Soar’’, Report on directed-study research. University of Michigan. Unpublished.

16. Johnson-Laird, P., *Mental Models*, Harvard University Press, Cambridge, Massachusetts, 1983.

17. Just, M. A., and Carpenter, P. A., *The Psychology of Reading and Language Comprehension,* Allyn and Bacon, Boston, Massachusetts, 1987.

18. Kolodner, J. L., ‘‘Maintaining organization in a dynamic long-term memory,’’ *Cognitive Science*, Vol. 7, 1983, pp. 243-280.

19. Kolodner, J. L., ‘‘Reconstructive memory: A computer model,’’ *Cognitive Science*, Vol. 7, 1983, pp. 281-328.

20. Kowalski, B. and VanLehn, K., ‘‘Inducing subject models from protocol data,’’ *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, 1988, pp. 623-629.

21. Laird, J. E., Congdon, C. B., Altmann, E. and Swedlow, K., ‘‘Soar User’s Manual: Version 5.2,’’ Tech. report, Electrical Engineering and Computer Science, University of Michigan, October 1990.

22. Laird, J. E., Yager, E. S., Tuck, C. M., and Hucka, M., ‘‘Learning in tele-autonomous systems using Soar,’’ *Proceedings of the NASA Conference on Space Telerobics,* 1989, forthcoming.

23. Laird, J. E., Newell, A., and Rosenbloom, P. S., ‘‘Soar: An architecture for general intelligence,’’ *Artificial Intelligence,* Vol. 33, 1987, pp. 1-64.

24. Langley, P., ‘‘Language acquisition through error recovery,’’ *Cognition and Brain Theory,* Vol. 5, 1982, pp. 211-255.

25. Larkin, J. H., ‘‘Enriching formal knowledge: A model for learning to solve problems in physics,’’ in *Cognitive Skills and Their Acquisition,* Anderson, J. R., ed., Erlbaum, 1981.

26. Lewis, R. L., Huffman, S. B., John, B. E., Laird, J. E., Lehman, J. F., Newell, A., Rosenbloom, P. S., Simon, T., and Tessler, S. G., ‘‘Soar as a unified theory of cognition: Spring 1990,’’ *Twelfth Annual Conference of the Cognitive Science Society*, 1990, pp. 1035-1042.

27. Lewis, R. L. and Newell, A. and Polk, T. A., ‘‘Toward a Soar theory of taking instructions for immediate reasoning tasks,’’ *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society,* 1989, pp. 514-521.

28. McCord, M. C., ‘‘A new version of slot grammar,’’ Tech. report, IBM Research Division RC 14506, 1989.

29. Mel’cuk, I. A., *Dependency Syntax: Theory and Practice,* State University of New York Press, Albany, New York, 1988.

30. Newell, A. and Simon, H., *Human Problem Solving,* Prentice-Hall, Englewood Cliffs, New Jersey, 1972.

31. Newell, A., *Unified Theories of Cognition,* Harvard University Press, Cambridge, Massachusetts, 1990, Chapters 7 and 8.

32. Nirenburg, S., Lesser, V., and Nyberg, E., ‘‘Controlling a language generation planner,’’ *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence,* 1989, pp. 1524-1530.

33. Polk, T. A. and Newell, A., ‘‘Modeling human syllogistic reasoning in Soar,’’ *Proceedings of the Annual Conference of the Cognitive Science Society*, August 1988, pp. 181-187.

34. Polk, T. A. and Newell, A. and Lewis, R. L., ‘‘Toward a unified theory of immediate reasoning in Soar,’’ *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, 1989, pp. 506-513.

35. Polson, P. G. and Kieras, D. E., ‘‘A quantitative model of the learning and performance of text editing knowledge,’’ *CHI ’85 Proceedings*, 1985, pp. 207-212.

36. Pritchett, B. L., ‘‘Garden path phenomena and the grammatical basis of language processing,’’ *Language,* Vol. 64, 1988, pp. 539-576.

37. Riesbeck, C. K., ‘‘An expectation-driven production system for natural language understanding,’’ in *Pattern-directed Inference Systems*, Waterman, D. A. and Hayes-Roth, R., eds., Academic Press, New York, 1978.

38. Sager, N., *Natural Language Information Processing,* Addison-Wesley, Reading, MA, 1981.

39. Sells, P., *Lectures on Contemporary Syntactic Theories: An introduction to government-binding theory, generalized phrase structure grammar, and lexical-functional grammar,* Center for the Study of Language and Information, Stanford, CA, 1987.

40. Shieber, S. M., *An Introduction to Unification-Based Approaches to Grammar,* Center for the Study of Language and Information, Stanford, CA, 1986.

41. Simon, H. A., *Models of Thought, Volume 1,* Yale University Press, New Haven, Connecticut, 1979.

42. Simon, H. A., *Models of Thought, Volume 2,* Yale University Press, New Haven, Connecticut, 1989.

43. Singley, M. K. and Anderson, J. R., *The Transfer of Cognitive Skill,* Harvard University Press, Cambridge, Massachusetts, 1989.

44. Tambe, M., Newell, A., & Rosenbloom, P. S., ‘‘The problem of expensive chunks and its solution by restricting expressiveness,’’ *Machine Learning,* 1990, pp. 299-348.

45. Thibideau, R., Just, M. A., and Carpenter, P. A., ‘‘A model of the time course and content of reading,’’ *Cognitive Science,* Vol. 6, 1982, pp. 157-203.

46. Tomita, M., *Efficient Parsing for Natural Language: A fast algorithm for practical systems,* Kluwer Academic Publishers, Boston, Massachusetts, 1986.

47. VanLehn, K., ‘‘Toward a theory of impasse-driven learning,’’ in *Learning Issues for Intelligent Tutoring Systems,* Mandle, H. and Lesgold, A., eds., Springer Verlag, 1988.

48. VanLehn, K., *Mind Bugs: The origins of procedural misconceptions,* MIT Press, Cambridge, Massachusetts, 1990.

49. Warner, J. and Glass, A. L., ‘‘Context and distance-to-disambiguation effects in ambiguity resolution: evidence from grammaticality judgements of garden path sentences,’’ *Journal of Memory and Language,* Vol. 26, 1987, pp. 714-738.

50. Winograd, T., *Language as a Cognitive Process, Volume 1: Syntax,* Addison Wesley, Reading, Massachusetts, 1983.

51. Woods, W. A., ‘‘Progress in natural language understanding: An application to lunar geology,’’ *AFIPS Conference Proceedings,* 1973, pp. 441-450.

Table of Contents

List of Figures