Integrating Knowledge Sources in Language Comprehension

Jill Fain Lehman and **Richard L. Lewis** and **Allen Newell**
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Multiple types of knowledge (syntax, semantics, pragmatics, etc.) contribute to establishing the meaning of an utterance. Immediate application of these knowledge sources is necessary to satisfy the real-time constraint of 200 to 300 words per minute for adult comprehension, since delaying the use of a knowledge source introduces computational inefficiencies in the form of overgeneration. 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 present 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.

Introduction

Language comprehension is the process by which knowledge is used to map an utterance to a meaning. Many sources of knowledge contribute to this process: morphology, syntax, semantics, pragmatics, and discourse conventions are those most often considered. In addition to providing knowledge that may be critical in finding the correct mapping, each source may contribute significantly to reducing the effects of local ambiguity on the search for a meaning. Reducing search is a computational necessity; without multiple knowledge sources even common constructions such as prepositional phrase attachment can lead to an exponential number of interpretations. More important, search reduction is necessary to satisfy the real-time constraint of 200 to 300 words per minute required to model adult comprehension. To guarantee the correct mapping we need all knowledge sources. To guarantee a minimal search space we must be able to bring each type of knowledge to bear as soon as it is relevant.

Immediate application of multiple knowledge sources is a key component in many psychological theories of comprehension and parsing (e.g., [Altmann and Steedman, 1988, Marslen-Wilson and Tyler, 1980, Thibadeau *et al.*, 1982]). However, constructing a system to bring its knowledge to bear at the right moment is difficult. There seems to be a trade-off between search efficiency and ease of engineering. At one end of the spectrum is a pipeline design in which knowledge sources are ordered (morphology, syntax, semantics, etc.), and each source generates a set of structures based on the previous phase's output. A system with separated knowledge sources is relatively easy to create and extend, and exhibits few surprising or inconsistent behaviors as the grammar becomes large. Yet, by delaying the application of a type of knowledge, initial stages overgenerate, producing extra interpretations and unnecessary search[1]. At the other end of the spectrum is a fully-integrated design which encodes all the knowledge in the generator. In short, all knowledge sources contribute at once to the meaning structure produced as each word is comprehended. Although overgeneration is prevented, building such a system requires anticipating and keeping track of all

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.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NSF, The Markle Foundation or the U.S. government.

[1]The work in principle-based parsing [Fong and Berwick, 1990] is an attempt to control the overgeneration in a syntactic pipeline while preserving its ease of engineering. However, since there is no single optimal ordering of modules, computing the optimal ordering for each sentence, while inexpensive, is heuristic. Consequently, overgeneration may still result. Similarly, blackboard models like Hearsay [Erman *et al.*, 1980] and READER [Thibadeau *et al.*, 1982] may reduce inefficiency by allowing for more flexible interaction than the pipeline approach, but cannot guarantee against overgeneration to the same degree full integration can. Furthermore, the same flexibility that can increase efficiency makes creating and extending the knowledge sources difficult.

the points at which each type of knowledge is relevant. With multiple knowledge sources, fully specifying the points of interaction becomes quite complex, and unexpected inconsistencies tend to arise as the grammar becomes large. In fact, no fully-integrated system has been attempted to date. Instead, most attempts at integration have combined only syntax and semantics, generally through the use of semantic grammars [Birnbaum and Selfridge, 1981], domain-specific syntactic grammars [Sager, 1981], or semantic annotations to syntactic rules [Shieber, 1986].

We believe that NL-Soar [Lehman *et al.*, 1991], the model of language comprehension we have implemented in Soar [Laird *et al.*, 1987, Lewis *et al.*, 1990, Newell, 1990], is a significant step toward a fully-integrated system. It comes closer to achieving ideal search behavior than previous approaches by bringing all sources of knowledge to bear as soon as possible. At the same time, the system maintains the modularity of a pipeline design. We achieve this combination of advantages by virtue of Soar's learning mechanism, *chunking*. Initially, comprehension proceeds by deliberate problem solving using independently represented knowledge sources. Through chunking, new knowledge is added to the system that applies all these sources in a single processing step for each word.

Overview of NL-Soar

Soar distinguishes between knowledge made available through search in problem spaces and knowledge available immediately through recognition. The computational model underlying Soar supports this view by committing to a long-term *recognition memory* (realized as a parallel production system) that yields knowledge in constant time. Problem solving proceeds recognitionally when Soar can select or apply operators in a problem space by simply drawing on this memory. Given the mapping of Soar onto human cognition [Newell, 1990], there is time for only a few (1-3) operators per word within the 200-300 millisecond per word constraint. Therefore, achieving recognitional comprehension is cast as the problem of creating a *comprehension operator* [Newell, 1990] that is directly implemented by productions in the recognition memory. A comprehension operator applied to a word must simultaneously bring to bear all the knowledge necessary to understand the word in the given context. Thus, the comprehension operator is a realization of the *immediacy of interpretation principle* [Thibadeau *et al.*, 1982]. Fully-integrated comprehension for the whole sentence proceeds via word-by-word application of comprehension operators.

Comprehension operators map an utterance to its meaning. The representation for meaning used in NL-Soar is a *model* of the situation that the utterance is about [Johnson-Laird, 1983, Newell, 1990, Polk *et al.*, 1989]. The essential property of models is that the elements in the representation map one-to-one into the situation. The right-hand side of Figure 1 shows the situation model that results from comprehending the sentence *The star above the circle is red.* To aid in building the situation model, NL-Soar also constructs an *utterance model* (Figure 1, left-hand side). This model is

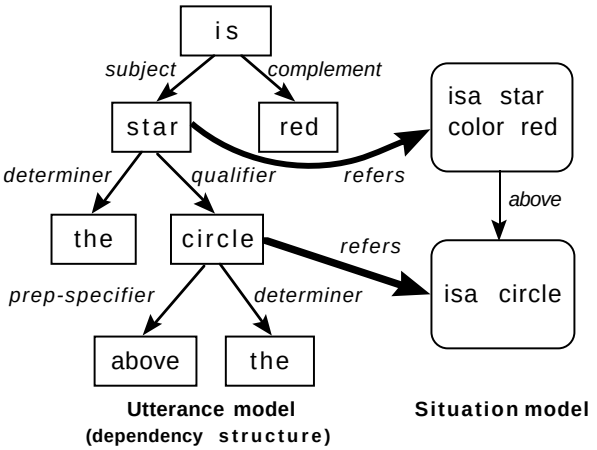


Figure 1: **The models for** *The star above the circle is red.*

a type of dependency structure [Hays, 1964, McCord, 1989, Mel'ĉuk, 1988], in which nodes correspond to words, and arcs label grammatical relations between words. Thus, the utterance model reflects the syntactic structure of the utterance, while the situation model reflects the structure of the situation being discussed. The relationship between the two models is one of *reference*; parts of the utterance model may refer to objects in the situation. Both models provide context for the comprehension process[2].

If models come from the application of comprehension operators, where do comprehension operators come from? As noted above, hand-coding fully-integrated operators for each word is an extremely difficult and complex task. In NL-Soar, however, hand-coding is unnecessary—comprehension operators arise automatically. To understand how, examine the system's problem spaces in Figure 2. The Comprehension space is where the recognitional comprehension capability is to reside. When Soar first attempts to apply the comprehend operator for a word in a new context, however, an *impasse* arises because the knowledge is not immediately available in recognition memory. Soar responds to this impasse (as it does all impasses) by creating a new subgoal to acquire the needed knowledge through search in another problem space. Knowledge in NL-Soar then chooses the Language space to deliberately implement comprehension by searching the space of partial utterance and situation models. For the operators in the Language space to construct and modify the two models, various syntactic, semantic, and pragmatic constraints on those operators must be satisfied. Ensuring that the constraints are met happens through the sequential application of operators in the Constraint space. If an impasse arises while checking semantic constraints, the Semantics space accesses the current situation model or general world knowledge to deliberately justify a semantic relation.

The result of the deliberate problem solving in the lower three spaces is to resolve the impasse for the comprehend

[2] Additional models may be needed (e.g. a discourse model) as new knowledge sources are added to the system.

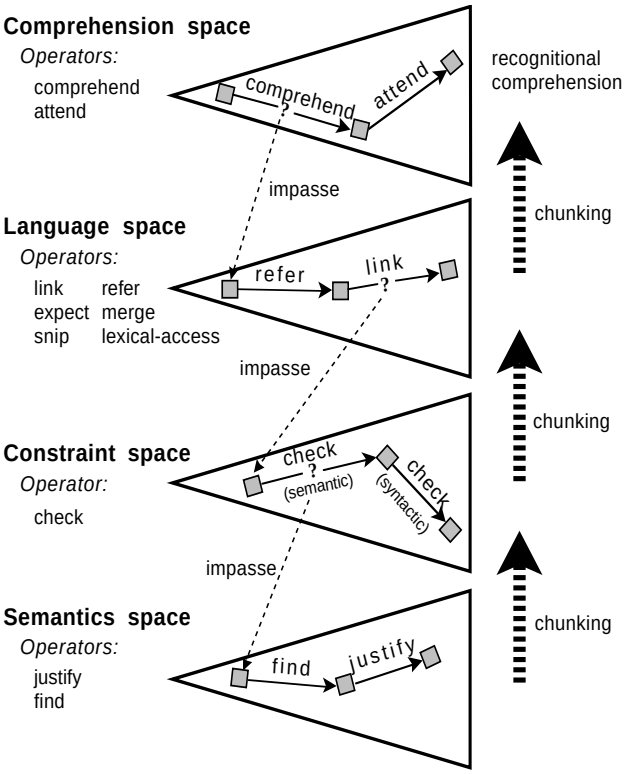
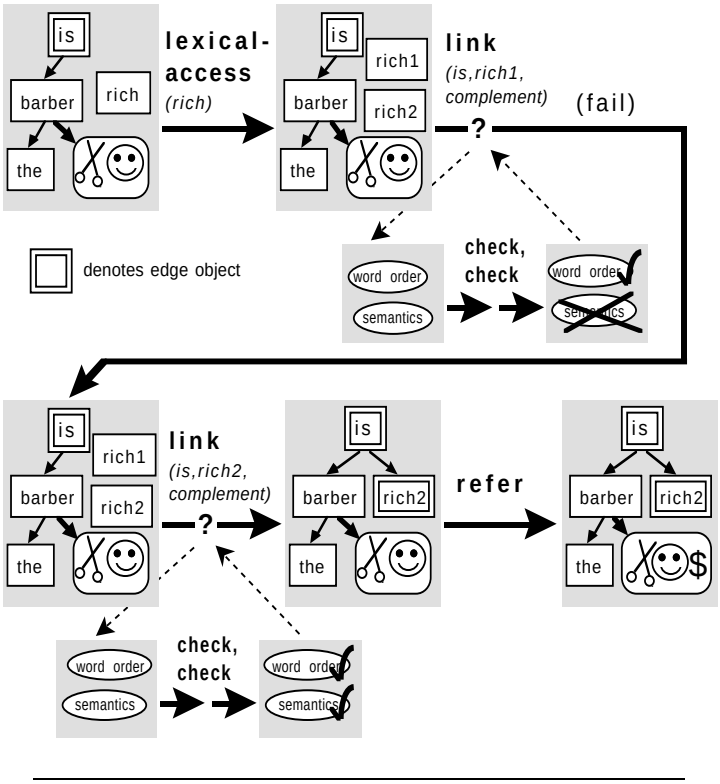


Figure 2: **The NL-Soar tower of problem spaces.**



Recognitional comprehension after chunking:

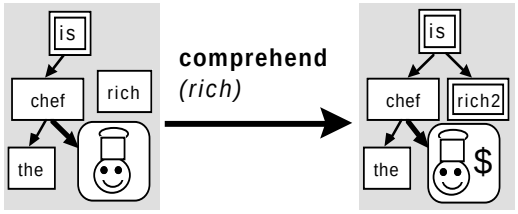


Figure 3: **Deliberate and recognitional comprehension of *rich*.**

operator, that is, to determine the appropriate changes to the two models in response to the word being comprehended. As these changes are made, Soar's chunking mechanism stores away the results of the problem solving and their pre-impasse conditions in the form of new productions (chunks) in recognition memory for the Comprehension space. These chunks form a recognitional comprehension capability, so that in future similar situations, comprehension can proceed without deliberate search. The chunks simultaneously bring to bear all the knowledge sources that were sequentially accessed in the lower problem spaces[3].

Examples of NL-Soar's behavior

The description of NL-Soar's problem spaces given above says little about the specific knowledge in the system or the particular type of processing that results. In this section, we make these ideas concrete by tracing the system's behavior with three simple examples. The characterization that emerges is that of a single-path comprehension system that combines both *bottom-up* (data-driven) and *top-down* (expectation-driven) knowledge with a limited capability for repairing misinterpretations.

[3]We have concentrated on how chunking creates new productions in the Comprehension space. However, chunking takes place whenever an impasse is resolved. Thus, knowledge is constantly moving up the tower of problem spaces in Figure 2, making processing more efficient in all spaces.

Constructing the utterance and situation models

Consider comprehension of the word *rich* in *The barber is rich.* When the system cannot recognitionally implement the comprehend operator for *rich*, an impasse arises and problem solving continues in the Language space. Figure 3 illustrates the chain of operator applications that follows the creation of the Language space's initial state. As shown in the first grey box, the state consists of a partial utterance model and a single object in the situation model representing the barber. The first operator to be applied is lexical-access which retrieves two senses of *rich*: an adjective describing food (*rich1*), and an adjective meaning wealthy (*rich2*). Next, link operators are proposed for each sense. A link operator specifies a relationship between the word being comprehended and a word on the active edge of the utterance model[4]. Thus, functional, structural, and lexical ambiguity contributes to the parallel generation of multi-

[4]The active edge is a subset of the nodes in the utterance model corresponding to non-closed constituents. When *rich* is

ple link operators. In the absence of other knowledge, the system arbitrarily chooses which link operator to try.

In this case an attempt is made to link the first sense of rich (food). Associated with each link operator is a set of constraints which must be met before the operator can be applied. When knowledge of the success or failure of these constraints is not immediately available, the constraints are checked independently in the Constraint space. The current link operator evokes only two constraints: one for word order (the verb assigning a predicate complement must come before the complement), and another requiring semantic consistency between the predicate and the subject. Semantic consistency may be justified by using pragmatic knowledge (finding an appropriate referent already in the situation model) or by simple inference in the Semantics space. Since *rich1* describes food and not people, there is neither a barber who is rich (in the food sense) in the model, nor a justifying inference available from world knowledge. Thus the semantic constraint fails, and the link operator terminates with failure as well.

Next, the link for *rich2* is tried. Again, word order passes, and this time the semantic check passes as well (although no instance of a rich barber could be found in the model, knowledge exists in Semantics that allows a referent that isa person to be rich in the sense of wealthy). Since both constraints pass for *rich2*, this link operator succeeds, making the correct addition to the utterance model. This new piece of structure then triggers the refer operator, which updates the situation model by adding a "wealthy" property to the barber object. In general, the refer operator may create new objects, properties, and relations or resolve a reference to an existing object in the situation.

These changes to the utterance and situation models complete the application of the top-level comprehend operator for *rich*. Soar's chunking mechanism now integrates all four types of knowledge—lexical, syntactic, semantic, and pragmatic—used in the lower spaces into new productions in recognition memory. Figure 4 shows an example of one of these productions. These new productions directly implement the comprehension operator for *rich* in utterances similar to the example. As shown at the bottom of Figure 3, this part of *rich*'s comprehension operator will transfer to all predicate complement utterances in which the subject has a referent in the model with the appropriate semantic features.

IF comprehending *rich*	[lexical]
and there is a verb on the edge,	[syntactic]
and the verb has an unfilled pred-complement role,	[syntactic]
and the subject of the verb isa person,	[semantic]
and the subject's referent isn't already wealthy	[pragmatic]
THEN the sense of *rich* is wealthy,	[lexical]
assign *rich* to be the pred-complement of the verb,	[syntactic]
assert the wealthy property of the subject's referent	[pragmatic]

Figure 4: **A new production learned for *rich*'s comprehension operator.**

comprehended, only *is* is in the edge set available for attachment.

NL-Soar differs from previous attempts at integration (e.g., semantic grammars, domain-specific syntactic grammars, and syntactic grammars augmented with selectional restrictions) in three important ways. One, the single comprehension operator avoids generating any of the intermediate structures of the deliberate search. Two, the integration of referent resolution permits the context (the situation model) to aid in disambiguation. Three, the system is designed to be extendible to other knowledge sources while maintaining both modularity and recognitional comprehension.

Adding expectations and top-down knowledge

Often it is impossible to link an incoming word directly into the dependency structure. Consider the word *the* in *Mary loves the barber*. It does not properly attach to either *Mary* or *loves*; it must attach to an as yet unseen noun. Rather than buffer such words, NL-Soar uses expectations to maintain a connected utterance model. Figure 5 shows the creation

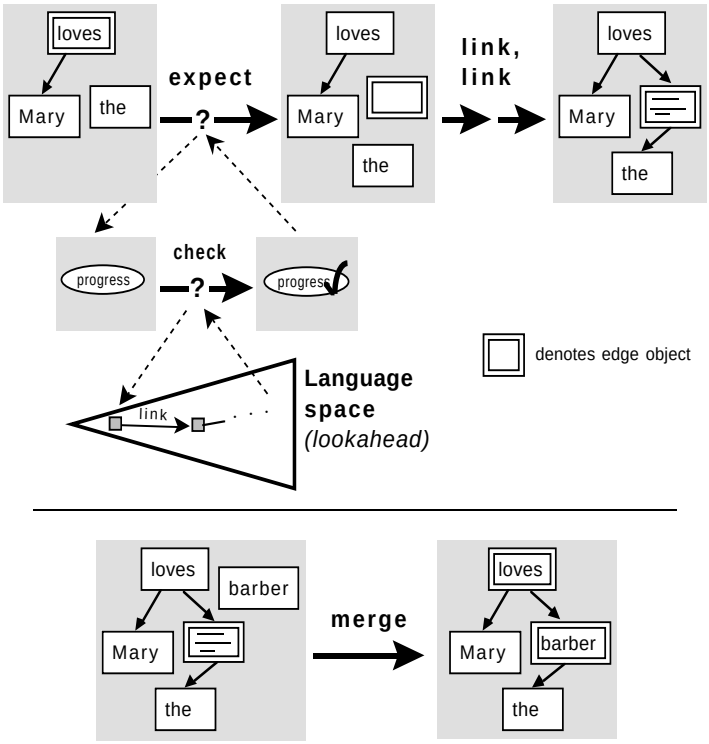


Figure 5: **Creating and merging expectations.**

of an expectation during comprehension of *the* (we omit the situation model in this and subsequent figures). Ignore for the moment the impasse on the expect operator and follow the states along the top row of the figure. First, expect produces an empty node—the expectation—that reserves a place for the future word. Next, two link operators are applied sequentially: the first assigning *the* as the specifier of the expectation, and the second assigning the expectation as the object of *loves*. As the links are made, the expectation

accretes the properties necessary to satisfy the constraints of the links. Thus, there is no special knowledge encoded in the expect operator about what to expect; instead, this knowledge is generated dynamically as a side effect of linking the expectation into the existing model. When the expected word finally arrives, the merge operator (Figure 5, bottom) merges it with the expectation, making sure that the new word is consistent with the constraints previously recorded.

How does the system know *when* to create an expectation? There are no specific conditions for proposing the expect operator—it may be attempted at any time. But it is only appropriate when it makes progress toward the successful comprehension of the incoming word as a cohesive part of the utterance to this point. This idea is cast as a *progress* constraint in the Constraint space (the impasse on the expect operator in Figure 5). To determine whether positing an expectation makes progress, NL-Soar searches in a copy of the Language space. This lookahead search reveals that *the* can be linked to the expectation, and the expectation linked to *loves*. Therefore a cohesive structure integrating *the* is possible. Chunks learned during lookahead search encode the conditions that determine when an expectation is needed.

All operators in the Language space use local, bottom-up knowledge. But like expect, all operators in the Language space also have a progress constraint. Thus, the scheme outlined above provides a general mechanism for automatically acquiring top-down knowledge in a comprehension operator via lookahead to pass the progress constraint. If an expectation is created in order to comprehend the current word, then the creation of the expectation and its links is combined with the necessary syntactic, semantic, and pragmatic constraints in the chunks that define the comprehension operator for that word. By integrating over all available knowledge sources, this conception of "top-down" subsumes and moves beyond that found in standard syntactic parsing strategies.

Adding recognitional repair

Since NL-Soar maintains a single interpretation during comprehension of a sentence [Altmann and Steedman, 1988, Frazier and Rayner, 1982, Pritchett, 1988], some mechanism is needed to recover if the system is led astray by a local ambiguity. Consider the sentence *Mary knows Sharyn loves the barber*. *Sharyn* is initially assigned the object role of *knows*. At the point of comprehending *loves*, *Sharyn* must be reassigned to be the subject of *loves* so that the embedded clause may serve as the object. When a previous interpretation proves to be incorrect, the snip operator is applied to undo part of the utterance model. An interpretation was incorrect if no link or expect operator can make progress. A snip operator is proposed for each link connected to a node in the edge set (lookahead may be evoked to determine which snip will be successful). Figure 6 shows the processing during comprehension of *loves*. First, snip cuts the object link between *knows* and *Sharyn*. Next, link operators apply, attaching *Sharyn* as the subject of *loves* and then *loves* as the clausal direct object of *knows*. The snip operator itself only disconnects a piece of the utterance; the

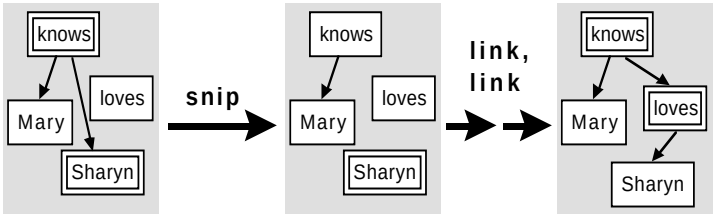


Figure 6: **Repairing a misinterpretation in** *Mary knows Sharyn loves the barber***.**

actual repair is carried out by the same link and refer operators we saw above. In contrast to an arbitrary backtracking scheme, the repair mechanism does not require maintaining a complete memory of previous choice points[5].

As in the case of creating expectations, if the comprehension of a word in the current context requires a repair, the effects of all the separate actions that went into making the repair will be captured in new productions for a single operator in the Comprehension space. In this way, limited *recognitional repair* becomes another part of NL-Soar's comprehension capability.

Summary

We have seen examples of sentences in which NL-Soar's processing proceeded strictly bottom-up (*The barber is rich*), included a top-down component (*Mary loves the barber*), or required repair (*Mary knows Sharyn loves the barber*). In each case, the operators that were applied accessed many different types of knowledge at different times. What is important to note about these examples is that the modularity of both the search control strategies and the knowledge sources is irrelevant in the Comprehension space. The comprehension operators produced by chunking are fully integrated, simply making changes to the models directly (and without search) for each word. Currently in NL-Soar, chunking is a compilation mechanism rather than a language acquisition mechanism. What changes is not the set of utterances that can be assigned meaning structures but the process by which a meaning is assigned. Chunking transforms the mapping process from a search-intensive sequential application of operators representing different but necessary types of knowledge, to the simultaneous application of all knowledge sources by a single operator.

Up to this point, the system's development has been driven by two types of demands. The first type of demand has been the satisfaction of basic psychological criteria: word-by-word, incremental comprehension under the real-time constraint. The second (and related) type of demand has been functional: how to compose meaning structures, how to prevent overgeneration, how to commit to structures before all relevant information is available, how to undo incorrect commitments. As a result of this emphasis, certain knowledge sources have been completely ignored to

[5]The limited nature of the repair means that garden path phenomena may arise. We are currently investigating NL-Soar's predictions in this area.

date (the discourse level, for example). Given the automatic way in which comprehension operators arise within NL-Soar, however, we believe the system can be extended to include additional knowledge sources without sacrificing the benefits of fully-integrated processing. In addition to the lack of some knowledge sources, both the system's syntactic and semantic coverage remain underdeveloped. The system's grammatical coverage is currently limited to about 75% of Allen's basic English grammar [Allen, 1987]. Its world model is a simple class hierarchy in the Semantics space. So far this degree of coverage has been adequate for preliminary work in three areas: a natural language interface to a robot arm [Laird *et al.*, 1990], instruction taking for simple reasoning experiments in psychology [Lewis *et al.*, 1989], and predicting garden path phenomena. To continue to extend the system's coverage in realistic and useful ways, we are exploring integrating NL-Soar with other Soar systems. For a more detailed description of the system, see [Lehman *et al.*, 1991].

Acknowledgements. The content and clarity of this paper were improved by the comments of Angela Kennedy Hickman. The integration of NL-Soar with the Robo-Soar project at the University of Michigan was done by Scott Huffman.

References

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

[Altmann and Steedman, 1988] G. Altmann and M. Steedman. Interaction with context during human sentence processing. *Cognition*, 30:191–238, 1988.

[Birnbaum and Selfridge, 1981] L. Birnbaum and M. Selfridge. Conceptual analysis of natural language. In R. C. Schank and C. K. Riesbeck, editors, *Inside Computer Understanding*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1981.

[Erman *et al.*, 1980] L. D. Erman, F. Hayes-Roth, V. R Lesser, and D. R. Reddy. The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *Computer Surveys*, 12(2), 1980.

[Fong and Berwick, 1990] S. Fong and R. Berwick. The computational implementation of principle-based parsers. In M. Tomita, editor, *Proceedings of the First International Workshop on Parsing Technologies*, pages 75–84, Pittsburgh, PA, 1990. Carnegie Mellon University.

[Frazier and Rayner, 1982] L. Frazier and K. Rayner. Making and correcting errors during sentence comprehension: eye movements in the analysis of structurally ambiguous sentences. *Cognitive Psychology*, 14:178–210, 1982.

[Hays, 1964] D. G. Hays. Dependency theory: a formalism and some observations. *Language*, 40(4):511–525, 1964.

[Johnson-Laird, 1983] P. N. Johnson-Laird. *Mental Models*. Harvard, Cambridge, MA, 1983.

[Laird *et al.*, 1987] J. E. Laird, A. Newell, and P. S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

[Laird *et al.*, 1990] J. E. Laird, E. S. Yager, M. Hucka, and C. M. Tuck. Robo-Soar: An integration of external interaction, planning, and learning using Soar. To appear in Van de Velde (ed.) *Machine Learning for Autonomous Agents*, 1990.

[Lehman *et al.*, 1991] J. F. Lehman, R. L. Lewis, and A. Newell. Natural language comprehension in Soar. Technical report, School of Computer Science, Carnegie Mellon University, 1991. Forthcoming technical report.

[Lewis *et al.*, 1989] R. L. Lewis, A. Newell, and T. A. Polk. Toward a Soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, pages 514–521, 1989.

[Lewis *et al.*, 1990] R. L. Lewis, S. B. Huffman, B. E. John, J. E. Laird, J. F. Lehman, A. Newell, P. S. Rosenbloom, T. Simon, and S. G. Tessler. Soar as a unified theory of cognition: Spring 1990. In *Twelfth Annual Conference of the Cognitive Science Society*, pages 1035–1042, 1990.

[Marslen-Wilson and Tyler, 1980] W. Marslen-Wilson and L. K. Tyler. The temporal structure of spoken language understanding. *Cognition*, 8:1–71, 1980.

[McCord, 1989] M. C. McCord. A new version of slot grammar. Technical Report RC 14506, IBM Research Division, Yorktown Heights, NY 10598, 1989.

[Mel'ĉuk, 1988] I. A. Mel'ĉuk. *Dependency Syntax: Theory and Practice*. State University of New York Press, Albany, 1988.

[Newell, 1990] A. Newell. *Unified Theories of Cognition*. Harvard University Press, Cambridge, Massachusetts, 1990.

[Polk *et al.*, 1989] T. A. Polk, A. Newell, and R. L. Lewis. Toward a unified theory of immediate reasoning in Soar. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, pages 506–513, 1989.

[Pritchett, 1988] B. L. Pritchett. Garden path phenomena and the grammatical basis of language processing. *Language*, 64:539–576, 1988.

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

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

[Thibadeau *et al.*, 1982] R. Thibadeau, M. A. Just, and P. A. Carpenter. A model of the time course and content of reading. *Cognitive Science*, 6:157–203, 1982.