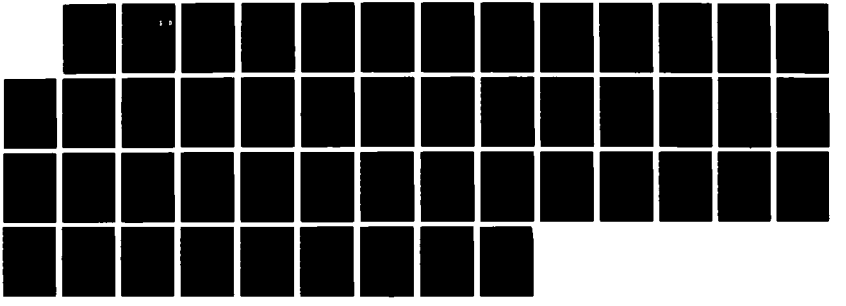
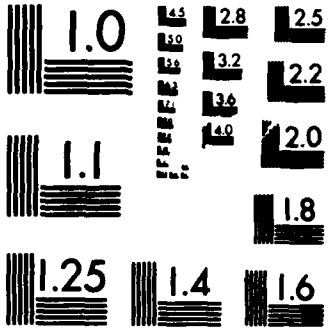
AD-A179 947 INTERACTIVE INSTRUCTIONAL SYSTEMS BASED ON MODULAR SYSTEMIC PARSERS(U) QUINTUS COMPUTER SYSTEMS INC MOUNTAIN VIEW CA E P STABLER 16 MAR 87 1/1

UNCLASSIFIED N00014-86-C-0817 F/G 12/5 NL





12

MIL-STD-847B
7 November 1983

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: unclassified	1b. RESTRICTIVE MARKINGS: none
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: unrestricted
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Quintus Computer Systems, Inc	6b. OFFICE SYMBOL (If applicable)
6c. ADDRESS (City, State and ZIP Code): 1310 Villa Street, Mountain View, CA 94041	7a. NAME OF MONITORING ORGANIZATION
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Office of Naval Research	8b. OFFICE SYMBOL (If applicable)
8c. ADDRESS (City, State and ZIP Code): 800 N. Quincy Street, Arlington, VA 22217-5000	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: N00014-86-C-0817
11. TITLE (Include Security Classification): Interactive Instructional Systems...	10. SOURCE OF FUNDING NOS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
12. PERSONAL AUTHOR(S): Edward P. Stabler, Jr	
13a. TYPE OF REPORT: Final	13b. TIME COVERED: FROM 86 Nov TO 87 Mar
14. DATE OF REPORT (Yr., Mo., Day): 87 March 16	15. PAGE COUNT: 47
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES: FIELD / GROUP / SUB. GR.	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This is the final report for "Interactive Instructional Systems Based on Modular Systemic Parsers," Phase I SBIR. This Phase I report proposes a design for a modular, principle-based parser that can formulate the various, independent functional descriptions of an utterance using independent systems of principles of the sort found in systemic and other grammars. This parsing system is embedded in a similarly modular framework for representing discourse information and conducting instructional dialogues based on work that has been done in computer-aided instruction. Phase II of this project will develop a completed prototype of the design proposed here.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL: Robert M. Keller, Director, Research	22b. TELEPHONE NUMBER (Include Area Code): 415-965-7700 / 22c. OFFICE SYMBOL

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

SECURITY CLASSIFICATION OF THIS PAGE

QUINTUS COMPUTER SYSTEMS, INC.

Final Report on
Interactive Instructional Systems
Based on Modular Systemic Parsers

U.S. Office of Naval Reseach

SBIR Contract N00014-86-C-0817

March 16, 1987

Edward P. Stabler, Jr.

Principal Investigator

QUINTUS COMPUTER SYSTEMS, INC.

Final Report on
Interactive Instructional Systems Based on Modular Systemic Parsers

U.S. Office of Naval Reseach
SBIR Contract N00014-86-C-0817

March 16, 1987

Edward P. Stabler, Jr., Principal Investigator

Table of Contents

1. Overview 1
2. Product Motivation: The Interaction Problem in IISs 2
3. MQP and IIS Product Definition 4
4. MQP Software Issues 7
   4. 1. Systemic Insights and Parsing 10
   4. 2. Formalization of the Grammar 11
   4. 3. Computational Approach 15
   4. 4. Rightmost Normal Form 20
   4. 5. Pronouns and the Discourse Model 24
   4. 6. Logical Query Optimization 25
   4. 7. Summary of MQP Design Features 25
5. Additional IIS Software Issues 26
6. Conclusions 26
   References 28
   Appendix 1. Compiling Parser Representations 31
   Appendix 2. A Listing and Session with a "Compiled" Parser 38

QUINTUS COMPUTER SYSTEMS, INC.

Final Report on
Interactive Instructional Systems Based on Modular Systemic Parsers

U.S. Office of Naval Reseach
SBIR Contract N00014-86-C-0817

March 16, 1987

Edward P. Stabler, Jr., Principal Investigator

List of Figures

3.1. IIS Architecture 3
4.1. MQP Architecture 10
4.2. A Rightmost Normal Form Parse 22

1. Overview

Phase I ONR SBIR funding has aided Quintus in specifying two products: a modular natural language query processor (MQP) designed for maximum portability and efficiency, and a package of tools for the implementation of Interactive Instructional Systems (IISs). These two products can be used separately, but they are designed to work together in facilitating the implementation of IISs with sophisticated natural language processing capabilities that will enhance the quality of the student-IIS interaction.

The market for efficient, portable natural language query processing systems and for interactive instructional systems is certainly present. Even fairly crude and expensive IISs have already demonstrated their value and are being deployed at many universities and technical training programs. The goal of this project has been to specify an approach to natural language query processing and instructional systems that will reduce the cost of such systems by maximizing portability and extensibility. These features have been enhanced in the products specified here by enforcing a rigorous modularity in design, isolating domain-dependent features clearly, and by implementing the entire system in a high-level declarative language (Quintus Prolog). Since increasing modularity can reduce efficiency, specialized optimizing techniques have been developed for compiling the modular IIS components into provably equivalent but much more efficient Prolog code which can in turn be compiled and executed efficiently on a wide range of standard, general-purpose computing machines.

Since there are obviously many differences between potential IIS domains, the IIS tools and MQP specified here will be sold with engineering support. An envisioned breakdown of the package as a product includes the following basic parts:

1. Quintus Prolog [33]
2. Interfacing tools (graphics, menu systems)
3. A complete MQP system in which domain-specific components are identified, together with thorough documentation of design and porting strategies
4. A complete IIS system that illustrates the coordination of interfacing tools, the MQP, and rule-based instructional capabilities, with thorough documentation of design and porting strategies
5. Installation service
6. Consulting service

The remainder of this report focuses on the technical description of the MQP and of the IIS, rather than on the business aspects. When we refer to the MQP and IIS we will not mean the complete list of services, but only the core technology and tools.

2. Product Motivation: The Interaction Problem in IISs

There is a tremendous demand for viable IISs that can relieve human teachers of part of the burden of instructing students without degrading the learning experience for the student. It is no surprise that most of the automated tutoring systems currently in use show real benefits in reducing teacher workload, but they are less successful in achieving the second goal of providing a satisfying and stimulating interaction for the student [1] [38] [39]. For example, in a recent report on the use of the "Lisp Tutor" system to instruct novice programmers at Carnegie-Mellon University [1], Anderson and Swarecki point out that although their system represents one of the most extensively developed approaches to intelligent computer-assisted instruction, it still is capable of only a rather severely limited range of interaction with the student. One limitation is that in worked examples and exercises, the student must write his program from left-to-right and in strictly top-down style. It is true that this is good programming style, but as psychological studies have shown [38], this style of programming presupposes knowledge about the basic capabilities of the programming system, a knowledge that a novice programmer will not have. However, interaction with the Lisp Tutor has a more serious limitation as well: the students' interaction with the instructional system is limited at all points by restrictions on the system's ability to enter into a dialogue. At many points the student is able to choose only one of a limited set of alternatives, and the system will not reliably respond to a general question the student might have about the tutor's explanations, or about how the current problem relates to the previous one. Consequently, this is among the important issues that Anderson and Swaricki identify as being raised by their experience with tutoring systems. They point out: "Improving the dialogue capability would improve instruction." The problem with providing better dialogue capabilities, though, is obvious: "To produce high-quality dialogue, however, considerable amounts of computation are required; and this conflicts with the need for rapid response."

The Lisp Tutor is one of the best efforts in IIS development, and it begins with a number of significant advantages that some other projects lack: it is a well-funded effort that is in high and growing demand; it is used by large numbers of students every year; the subject area is relatively simple and easy to represent in the computing system; and there have been numerous studies

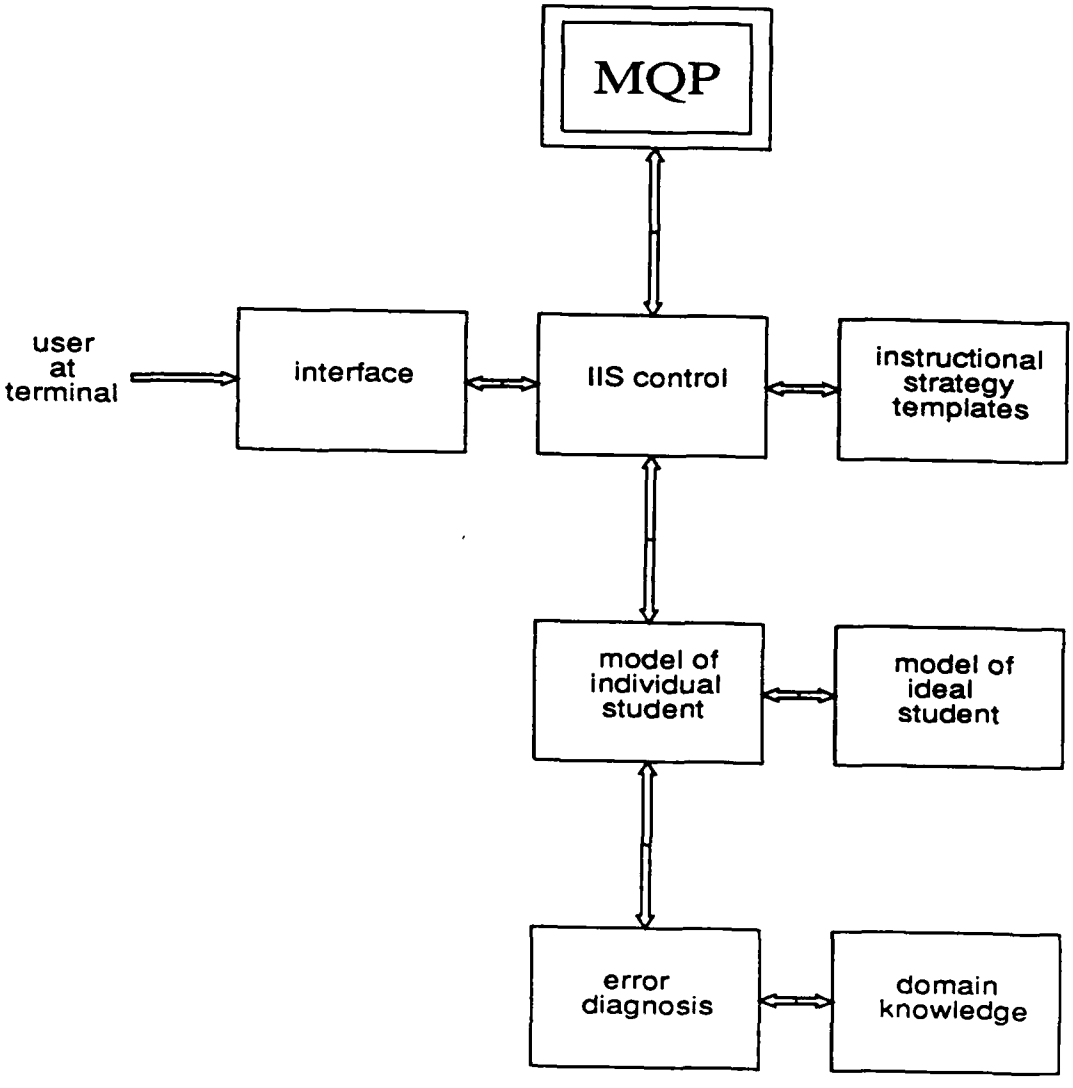


Figure 3.1. IIS Architecture

of novice programmers and their errors. So it is no surprise that the problems in the Lisp Tutor system, and in particular the problems with providing less restrictive interaction with the student, are problems that other practical IIS systems share. In their review of recent work in this field [35], Sleeman and Brown list this as one of four major shortcomings of all available IISs: "User interaction is still too restrictive, limiting the student's expressiveness and thereby limiting the ability of the tutor's diagnostic mechanisms." Less restrictive, more efficient natural language processing components would significantly improve interactive instructional system performance.

3. IIS and MQP Product Definition

The objective of this research was to design an interactive instructional system (IIS) based on a modular, natural language query processor (MQP) that makes use of the best available grammars of natural language in feasible query processing. In the MQP, modularity in design and a high-level, declarative implementation language are used to maximize extensibility and portability. The details of the IIS architecture itself may vary substantially from one domain to another, but the problems of providing an MQP that can help alleviate the "interaction problem" in IISs are similar across domains. Consequently, we have settled on the architecture illustrated in Figure 3.1. We will briefly describe each of the components depicted in this Figure.

The MQP interface. Notice that the MQP is embedded as a distinct component in the IIS. English input from the user will be directed immediately to the MQP by the IIS control component. The MQP will in turn be able to query the IIS control system for information about the state of the instructional system. The MQP will also be given independent access to the domain knowledge component. In this way, the student will be able to ask, at any point in the tutorial interaction, about material previously presented or about material that has not yet been presented.

In an IIS for teaching Prolog programming for example, the student may sometimes simply want access to reference information:

```
Query:  Explain the builtin predicate, name/2.

   ref-8-11 - name(?X,?L)

   name(X,L) is a relation between an atomic object X and a list L which
   consists of the ASCII character codes for the printed representation
   of X. Initially, either X must be instantiated to an atomic object,
   or L must be instantiated to a list of character codes (containing no
```

```
variables).  Otherwise the call simply fails.

If X is initially instantiated to an atom or number, L will get bound
to the list of character codes that make up its printed
representation.  If X is uninstantiated and L is initially
instantiated to a list of characters that corresponds to the correct
syntax of a number (either integer or float), then X will get bound
to that number; otherwise X will be instantiated to an atom
containing exactly those characters.  If neither argument is bound,
the goal fails.

Here are some examples of its use:

   | ?- name(foo,L).

   L = [102,111,111]

   | ?- name('Foo',L).

   L = [70,111,111]

   | ?- name(431,L).

   L = [52,51,49]

   | ?- name(X,[102,111,111]).

   X = foo

   | ?- name(X,[52,51,49]).

   X = 431

   | ?- name(X,"15.0e+12").

   X = 1.5e+13
```

In other cases, the MQP should be able to answer queries about the current training, where responding to these queries involves an analysis of the student's work or responding to general questions about the topic. For example, in working an exercise the student may produce a program that goes into an infinite loop on a certain input. A good debugging system will catch the infinite loop, analyze the computation to find the looping segment of the computation, and advise the student of its results. Tools for doing this in Prolog are well-known and have been implemented at Quintus Computer Systems. In such a case, the student may be prompted to ask a general question about this sort of difficulty, such as:

```
Query:  Does every procedure that calls itself cause an infinite loop?
```

Most IISs simply have no facilities for dealing with queries like this one. Although it is not feasible to expect an IIS to be able to answer every such question intelligently, responses to questions about common difficulties like this one can be prepared in advance. And as more common questions like this one are found, the IIS should be extended to provide helpful responses to these as well. For this reason, extensibility is important, and has proved to be a difficult problem for existing IISs [1] [38].

The MQP technology proposed here is carefully designed to allow for the extensibility needed for gradual improvement and adaptation of IIS query processing capabilities. A detailed account of the capabilities and internal structure of the MQP will be provided below.

Instructional strategy templates. All feasible tutorial systems, like university classes, are set up around carefully designed programs of instruction, which in this IIS are called "instructional templates." Using text and graphic presentations, these templates present the pertinent information about the domain in a systematic way, as if to an "ideal student." User responses to page prompts and exercises are used to control the rate of presentation, and sometimes even the content of the presentation: for example, extra material may be prepared in advance for students that have difficulty with certain sections, or for students that want additional, optional material on advanced topics. Some IISs provide nothing more than a set of templates, but recent work has shown how these can be usefully supplemented with capabilities for providing more specialized, individualized tutorial assistance to each student. These capabilities are provided by additional components that will be called by the IIS control system, perhaps in response to a trigger from an instructional template.

Models of the student. As noted above, some students may exhibit particular difficulty with certain material (which will be indicated by, e.g., errors in exercises), or they may exhibit particular interest in material that is optional. The IIS can often note these facts and incorporate them into a model of the individual student, which can then be used both in controlling the action of the basic instructional templates, and in directing error diagnosis routines to an appropriate level of sophistication. One approach to individual student modeling involves noting differences between the individual being tutored and an "ideal student." A system whose templates are designed for the ideal student can then be appropriately adapted for those students who are more or less sophisticated in aspects of the material being presented. This simple idea allows for substantial improvements in the quality of instruction: the students having more difficulty can be

given extra assistance in areas of difficulty, and those students doing unusually well can be given the material at an accelerated rate so they do not have to endure the tedious presentation of material they have already mastered. This capability is one of the real advantages of IIS education over even classroom instruction: individualized instruction can easily be provided, and it should be expected by IIS users.

Error diagnosis and domain knowledge. It is a trivial matter to provide the IIS user with immediate feedback on the correctness of responses to exercises, but recent work in IIS technology has shown feasible techniques for providing helpful diagnosis of the errors and corrective instruction in a range of domains. In most domains, well-developed IISs will make use of this technology. This aspect of IISs, which will obviously vary enormously from one domain to another, will be discussed in greater detail below, when we focus on a particular domain for our prototyping efforts, viz., Prolog programming.

4. MQP Software Issues

As noted above, intelligent natural language processing can make heavy demands on computational resources (memory and time), but in real, practical systems, IIS components must not slow system performance to an unacceptable level. This requirement is not peculiar to IISs. In fact, many natural language processing applications face exactly this obstacle, and so the options have been extensively researched. The work by Codd and his associates on a comfortable database query system [8] [9] has almost exactly the same goals: they tried to develop a system that can respond quickly and in a cooperative way to relatively unrestricted queries from a "casual" user (i.e., a user who cannot be expected to rigorously formulate his questions in a formal query language). Many of the strategies used in such work are applicable in the IIS domain as well.

One of Codd's strategies for improving the natural language interface is to try to make sense of user queries which are not completely analyzable by their limited English parser. This is an important idea because it is impossible at this point in natural language processing technology to design a system which can properly recognize perfectly grammatical English sentences, let alone the "almost grammatical" strings that one often gets in practical situations. Codd's approach is to assign structure to as much of the user's query as possible, and then try to make an educated guess about the query which is then paraphrased back to the user for verification.

Some of the recent work on interpreting what linguists call "telegraphic messages", i.e., abbreviated messages of the sort commonly found in professional and technical communications,

uses a similar strategy of, in effect, guessing what the missing parts of the communication are. Significant progress has been made in this area in work done for the Office of Naval Research [10] and elsewhere. This work appears to be applicable to many IIS domains as well.

Some of the strategies which initially appear most effective in optimizing the performance of natural language processing systems turn out to have unacceptable costs. An example of this is the idea of fully integrating semantic and discourse processing into the initial syntactic processing of a user's question. This strategy, the "semantic grammar" approach, has been used in some instructional systems, such as the SOPHIE system which was developed in a number of places including XEROX PARC [6]. The problem with this approach is that it sacrifices modularity and increases complexity enormously. As a prominent investigator has pointed out, not only does this approach produce an enormously complex natural language system, but the system becomes rigidly domain-specific: "...it must be written anew if the domain of discourse changed, and it would be extremely impractical to attempt to write such a grammar for anything but a limited application area" [2].

To deal with a reasonable range of student performances, IISs must be quite large and complex, and so a high degree of modularity in design is essential to reduce development and maintenance cost. Anderson and Swaricki report that their system, the Lisp Tutor, is being redesigned and rebuilt largely in an effort to obtain a higher degree of modularity. We have accordingly avoided a large, complex, non-modular, and inflexible natural language processor in our design.

The motivation behind semantic grammars, and in fact the general motivation behind reducing modularity in natural language systems, is easy to illustrate with a simple example of the need to resolve ambiguities. One source of ambiguities in natural language comes from various "attachment" possibilities for prepositional phrases. Consider the sentences "Show me the fault in the program" and "Show me the fault in the upper left hand window." In the former sentence, the more plausible interpretation is that the prepositional phrase ("in the program") modifies the *fault*, but in the latter sentence the prepositional phrase ("in the upper left hand window") most plausibly modifies the verb *show*. Notice that the appropriate decision about what a prepositional phrase modifies in a sentence like either of these can be influenced by discourse context. If I am using the Lisp Tutor with a windowing display system, the interpretation of the latter sentence will be different than if I am using a glass-manufacturing instruction system that has only teletype input and output capabilities. So if we could make use of this information about

the context of the discussion at the earliest possible point, then the parser would not have to formulate the inappropriate representation at all. In more complex cases, the savings in processing time that can come from early resolution of ambiguities can be quite dramatic.

This same kind of situation arises within the parsing component. For example, we want the system to know that it can accept sentences that have both forms *[show [the fault in the program]]* and *[show [the fault][in the program]]*, and these structural configurations could be defined with rewrite rules of some kind. But sometimes one of these structures, although allowed by the rewrite rules, is ruled out by other grammatical principles. The verb *put*, for example, requires both a direct object and a locative phrase, so "Put the fault in the window" has only one acceptable structure, *[put [the fault] [in the window]]*. We would like to apply this restriction on the use of the rewrite rules as soon as possible to rule out the alternative structure, but this threatens to remove the modularity in the representation of the rather different grammatical principles.

An approach to this problem has been developed in this Phase I effort, based on earlier work by the Principal Investigator of this project [40] [41] and others. Notice that the character of the dilemma here is very similar to the problem of finding solutions to "constraint satisfaction problems" [23]. There has been research on representing these problems in a modular, declarative notation (viz., logic), and then automatically folding the various modular components of the problem into each other to improve efficiency [4] [12]. In this way, modularity can be maintained in the system without substantial loss in efficiency because the system does an automatic optimization. These techniques can be applied to the natural language parsing problem if we can begin with an appropriately modular representation of the language. They can also be applied to integrate aspects of semantic processing and discourse modeling without removing modularity. This approach has been adopted in the design proposed here, and it has proven to be feasible even when the system is extended to handle "telegraphic" and other "almost grammatical" user queries. Preliminary treatments of these difficult inputs has been introduced into the prototypes developed in this Phase I effort, described below.

The modular design depicted in Figure 4.1 will provide for efficient processing of user queries while maximizing portability and extensibility. We will briefly discuss the most important aspects of this design, explaining in some detail one of the most difficult examples of the automatic "folding together" of grammatical principles.

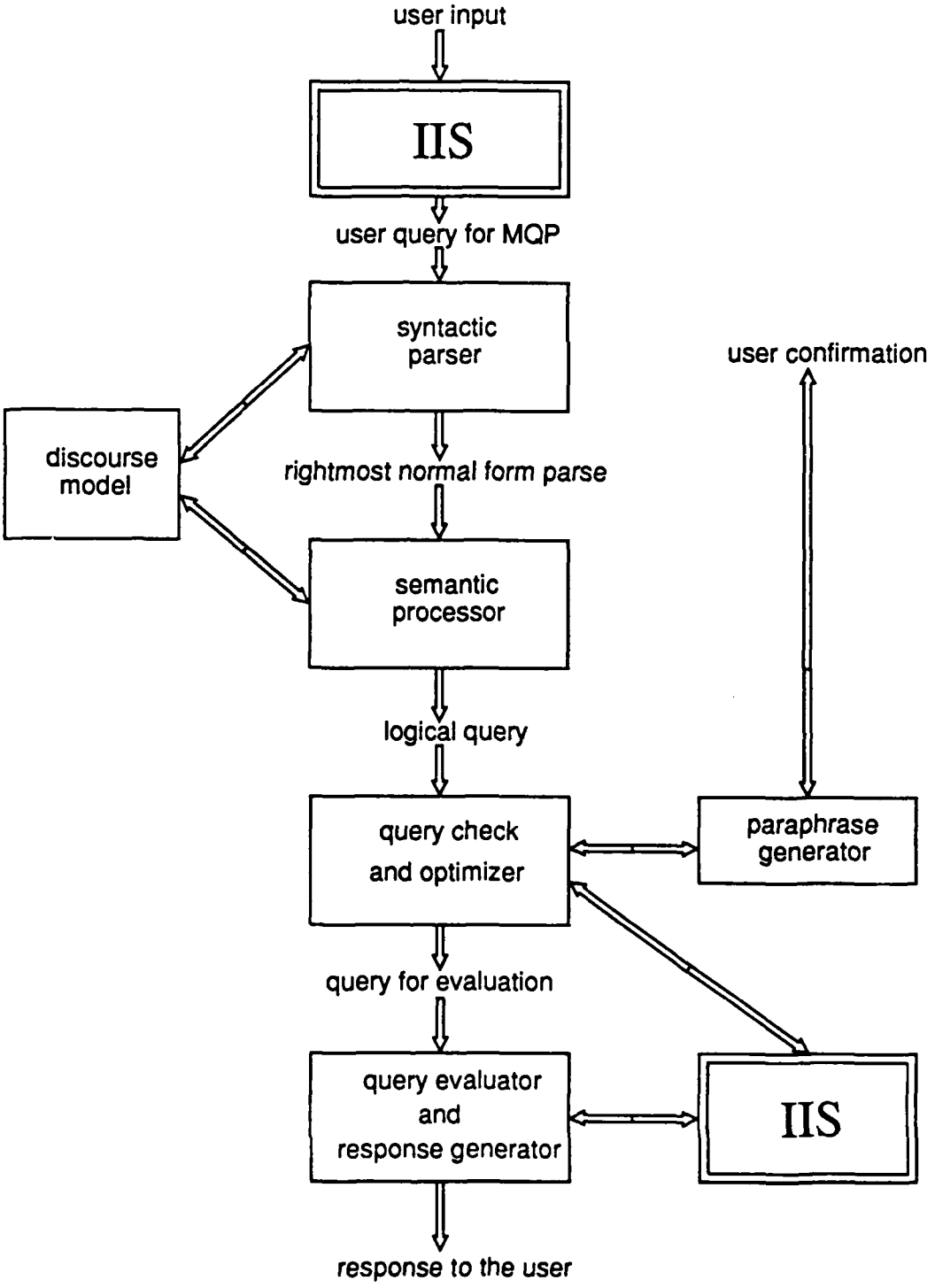


Figure 4.1. MQP Architecture

4.1. Systemic Insights and Parsing

The approach to providing an efficient natural language processing system that has just been sketched (in the previous section) requires that we have a modular representation of the language we want to deal with. And of course, if we want to make use of semantic information and discourse context in our structural analysis of the user input, this semantic and discourse information must also be represented by independent, modular sets of principles that can be applied at appropriate points (where those points are determined by the character of those principles and the optimization strategy). The emphasis of systemic grammar has always been on the semantic and pragmatic features of language, rather than on the details of the surface constituent structure [3] [14]. This emphasis is most valuable in the present project, because, while there are a number of well-developed grammatical approaches to surface constituent structure, some of which have even incorporated aspects of systemic grammar in the form of modular systems, the study of semantic and pragmatic features is relatively undeveloped.

The other feature of systemic grammar that is appropriate for an approach like the one sketched above, is its use of relatively independent sets of principles to describe different aspects of utterances and texts. Indeed, this has led previous investigators in systemic grammar to the same dilemma of modularity versus efficiency that motivates our particular design. For example, Winograd points out that "The problem of describing the realization of simultaneous choices along different dimensions is quite complex" [51], and Mann and Matthiessen have suggested that any implementation of systemic principles must have "a collection of processes that can work together" [27]. Our approach will use modular, declarative statements of the role of each of the grammatical processes, and their orchestration will be managed according to our specialized optimization strategy. In this way, modularity can be preserved while we incorporate systemic insights of recent work on discourse and pragmatic features of language [7] [3] [24] [25].

4.2. Formalizing the Grammar: Phrase structure and realization

In language understanding systems it is important to be able to deal with a fairly complete subset of the language. In this respect, language generation systems have an easier task. A language generation system need only be able to generate some sentence to convey each message, to perform each communicative act, whereas a language understanding system should, ideally, be able to deal with *every* sentence that might convey that message. Consequently, it is important to have a fairly all-encompassing grammar of the language. And on the other hand, it is

important that the grammar not accept strings that are not part of the language, since this will aggravate the already very serious ambiguity problem that is posed by natural languages.

Since getting an adequate representation of the grammar of the language is a critical desideratum on the success of this project, we have chosen to follow the approach of Winograd [51] and others: we use well studied parsing techniques, building insights from systemic grammar and other linguistic theories into a system that does not deviate too substantially from the standards in the field. Winograd uses an ATN formalism for his grammar. We will use an extended version of Definite Clause Grammars(DCGs) which have been shown to be an elegant formalism capable of representing ATN systems in a declarative, logical language [31] [32].

One significant advantage of using a logical representation of the grammar is that we can get an automatic resolution of the *realization* problem: we can use modular representations of each system of the grammar, and just require that every string satisfy the principles of every system. Unfortunately, the most straightforward approach to incorporating such principles into a logic grammar are not generally feasible, as noted in the following section on the parsing algorithm. As a result, we use an "optimization" step to enforce the grammatical principles. The principles are enforced, in the optimized representation, in one of the following ways: dependencies that are naturally stated as conditions on nodes that are sisters are "built in" to the argument structure of the grammatical categories, whereas dependencies that span more of the tree-representation are explicitly stated and their enforcement is "folded into" the tree-building process. This distinction can be illustrated by an example.

Consider a grammar which uses a phrase structure rule that says that a *vp* ("verb phrase") node can dominate a *v* ("verb") node and a *np* ("noun phrase") node. We would like to incorporate principles that tell us which verbs can occur in this kind of structure and which verbs will assign "objective case" to the *np*. These principles could be stated explicitly, but we get a much more efficient grammar by building these principles directly into the phrase structure rules. In a DCG, these restrictions can be imposed directly in the structure of the arguments to the grammatical categories. [1] The following grammar these relationships with argument structure;

[1] Actually, some of the feature restrictions cannot be handled with unification of argument structures. Extensions to standard unification techniques can be made to handle disjunctive features more efficiently, as noted in recent research on systemic parsing at the University of Southern California [17]. In most practical grammars, though, standard unification techniques will suffice.

it is shown in the "rewrite rule form that is accepted by Prolog and transformed into a "Definite Clause Grammar" parser that will run in Prolog [31]:

```
s --> np(subjective,Person,Number),vp(Person,Number).

vp(Person,Number) --> v(Person,Number).
vp(Person,Number) --> v(Person,Number,np_complement), np(objective,_,Number).

v(3,singular,np_complement) --> [likes].
v(3,singular) --> [glows].

np(subjective,3,singular) --> [he].
np(subjective,3,singular) --> [she].
np(subjective,3,plural) --> [they].
np(objective,3,singular) --> [her].
np(objective,3,singular) --> [him].
np(objective,3,plural) --> [them].
```

This grammar will accept "he likes her" and "she glows", but will not accept "her likes he" or "she glows her". Notice that this grammar also enforces agreement in person and number between the subject *np* and the *v*, even though these nodes are not sisters. This is nevertheless quite natural, since the respective positions of the subject *np* and the *v* can be specified precisely.

These sorts of relationships between precisely specified nodes in the tree contrast with relationships which are considerably less restricted. It is well known that a question that begins with a wh-word (what, who, which, when) is well-formed only if that wh-word can be seen as having been moved from another position in the sentence. For example, in the following sentence we mark the position from which the wh-word came with "Λ", the symbol for the empty terminal string:

who do you like Λ.

The "Λ" is the object of the verb. But in the following string, there is no "empty position" from which the wh-word could have come, and hence the string is not a good sentence:

who do you like john.

So a grammar must enforce this restriction on the relation between initial wh-words and empty positions in the sentence. However, this restriction is difficult to enforce, since the empty position can occur in many positions, and those positions can be arbitrarily far from the initial wh-word:

who Λ *likes john.*

who do you like Λ.

who did you say that john liked Λ.

who did you say that john liked Λ.

who did you say that john liked Λ.

who did you say that john told frank mary liked Λ.

This relation is covered by principles that must be considerably more flexible, and which must go well beyond conditions on sister nodes in the structural representation. This relation also interacts with the operation of the case and theme systems in complex ways. This is an example of one sort of condition on acceptable sentences which will be stated explicitly, rather than being built in to the phrase structure rules directly.

This last type of relation is what motivates the "HOLD list" in ATNs like the ones used by Winograd and others. This type of relation is given special treatment in most of the standard approaches to parsing. Representations of these relations in logical systems have been studied by the Principal Investigator of the project and others [40] [41] [30]. The approach proposed here, though, is unique in having the enormous advantage that the restrictions on the relation can be stated *explicitly* in a declarative language. These statements are necessarily rather intricate, but they are nevertheless far preferable to the intricate and largely *implicit* mechanisms used by some other approaches to parsing. Our approach to such relations will be discussed further in the following section.

The emphasis of systemic grammar has always been on the semantic and pragmatic features of language, rather than on the details of the surface constituent structure [3] [14]. Indeed syntactic, semantic and pragmatic features are given a quite uniform and balanced treatment in the systemic tradition. The "transitivity system" corresponds quite closely to the principles of semantic interpretation that determine the argument structure of predications, and the "information system" corresponds to a certain level of discourse analysis. This emphasis on the non-syntactic aspects of language is most valuable in the present project, because, while there are a number of well-developed grammatical approaches to surface constituent structure, some of which have already incorporated aspects of systemic grammar in the form of modular systems, the study of semantic and pragmatic features is relatively undeveloped. These semantic and pragmatic principles can be stated explicitly and independently just like the other grammatical relations, and then automatically integrated into the parsing process using the techniques described in the following section.

In sum, the formalization we have selected will be a logical representation of phrase structure rules with a variety of systemic dependencies built into them, together with explicitly stated dependencies which will be stated explicitly. The explicitly stated principles can be kept completely modular, which means that there representation can be kept closer to the form preferred by linguists. Independent systems of grammar can be represented independently and automatically integrated in the course of parsing.

4.3. Computational Approach

The dilemma of modularity versus efficiency motivates our design. Our design uses a modular representation of the language and yet has an efficient query processing engine suitable for our interactive instructional system.

Let's consider how a formalization of the restriction on placement of "empty categories" or "traces" (A) can be implemented, since this is probably the most difficult relation between nodes in a structural representation that needs to be enforced. We show how a direct logical representation of such conditions can be provided and efficiently used by left-to-right Horn clause theorem provers, like Prolog, SLD resolution with a leftmost selection rule, or Earley deduction. [2] On this approach, linguistic constraints are represented as constraints on logical derivations in the parser to ensure that a string is proven to have a well-formed structure just in case the linguistic theory entails that it does.

Consider the following grammar:

```
s --> np(F,Index), vp.
np(-wh,Index) --> name.
np(-wh,Index) --> det, n.
np(-wh,Index) --> det, n, sbar.
np(F,Index) --> trace(Index).
np(+wh,Index) --> rel_pro.
trace(Index) --> [].
sbar --> comp, s.
comp --> np(+wh,Index).
vp --> verb, np(F,Index).
```

[2] See Lloyd [18] for a presentation of SLD resolution techniques for Horn clause theorem proving, substitutions, computation rules, Prolog, etc. Earley deduction, basically an "all paths at once" generalisation of SLD resolution inspired by Earley's context-free parsing algorithm, is described in [32].

```
name --> [mary].
det --> [the].
n --> [man].
rel_pro --> [who].
verb --> [likes].
```

In this simple grammar, special subclasses of *np* are singled out by the $+wh$ and $-wh$ features, with *trace* as the only category that is in both subclasses. The category *trace* expands to the empty terminal string. Also, notice that every *np* has an uninstantiated argument, *Index*, which will play a special role in enforcement of the constraints.

According to this grammar, the set of sentences includes "mary likes the man", "the man who mary likes likes mary", "the man who likes mary likes mary", "the man who likes the man who likes mary likes mary", and so on. Notice, though, that the set also includes: "likes", "likes mary", "likes the man likes", and other strings that are not English sentences. We eliminate the acceptance of these latter strings by imposing three simple constraints on derivations.

The context free derivation trees for the grammar correspond to logical derivation or proof trees in the definite clause representation, and so it is natural to attempt to represent constraints on acceptable structural representations as constraints on logical derivations. Since Gödel's work on providing a proof predicate for arithmetic in arithmetic, it is well known that we can represent logical derivations and constraints on derivations in the same logic that we use to represent the basic axioms used in the derivation.

The nature of the constraints that we would like to impose can be indicated by the following simple examples [45]:

1. *A theta condition.* If *comp* immediately dominates an *np*, that *np* must be coindexed with a *trace* that is not dominated by *comp*.
2. *A trace-binding condition.* A *trace* must be coindexed with a preceding *np* that is immediately dominated by *comp*.
3. *A complex-NP constraint.* An *np* dominated by *sbar* cannot be coindexed with an *np* that is not dominated by that *sbar*.

These simple constraints suffice to block the acceptance of the ungrammatical strings mentioned above. Notice that these constraints are expressed in terms of structural relations (*immediately dominates, dominates, precedes*) and special relations between nodes in the structure (*coindexed*). Such constraints can be expressed as predicates on derivations in the logical representation of the rewrite grammar.

To formalize constraints like these, we first define terms that can represent derivations. We can define a function τ that will transform a logical theory expressed in Horn clauses into another theory which is "equivalent" except that derivations in the new theory build representations of themselves. The transformation is really quite simple: we add a variable as an argument to every predicate in the body of a clause, and add a term as argument to every predicate in the head of a clause, as illustrated in the following example:

$$
\begin{aligned}
S = \{ \quad & :\!-p, \\
& p\!:\!-q, r, \\
& q\!:\!-s, \\
& r, \\
& s \quad \}
\end{aligned}
$$

$$
\begin{aligned}
\tau(S) = \{ \quad & :\!-p(Proof), \\
& p(p/[Q,R])\!:\!-q(Q), r(R), \\
& q(q/[S])\!:\!-s(S), \\
& r(r), \\
& s(s) \quad \}
\end{aligned}
$$

The instance of $p(Proof)$ which we can prove in S is the one in which $Proof$ is instantiated to the proof tree: $p/[q/[s], r]$. This proof tree indicates that, in S, p (the root of the tree) can be proven by proving q and r, q can be proven by proving s, and s and r can be proven directly from atomic propositions.

We can formally define this transformation and provide proofs of its correctness. In particular, we can prove following two propositions which specify exactly the sense in which the output of the transformation is "equivalent" to the input:

Proposition 1. (*Noninterference*) $S \models \forall G$ if and only if $\tau(S) \models \forall \tau(G)\eta$, where η is a substitution restricted to variables that do not occur in G.

Proof: Since SLD-resolution is sound and complete, it suffices to prove that there is an SLD-refutation of $S \cup \{:\!-G\}$ with the correct answer substitution ϵ iff there is an SLD-refutation of $\tau(S) \cup \{\tau(:\!-G)\}$ with a correct answer substitution η which is restricted to variables introduced by τ. This is easily done with an induction on the length of the SLD proofs. ∎

Proposition 2. (*Representation Correctness*) Let G be an atom. Then $S \models \forall G$ if and only if $\tau(S) \models \forall \tau(G)\eta$ where η is a substitution $\{Proof/Tree\} \cup \varsigma$ such that *Proof* is the variable introduced into $\tau(G)$ by τ, and $(Tree)\varsigma$ corresponds to a derivation of G from S.

Proof: Again, an easy induction on length of the SLD proofs. ∎

In other words, if the original theory entails that a relation holds among some objects in the domain, then the transformed theory entails that a new relation holds among exactly the same objects together with one additional object, viz., a sound derivation of the claim that the original relation holds. And conversely.

Now that we have provided terms representing derivations in our theory, we can define predicated that will enforce conditions on those derivations. Consider again our first constraint:

1. *A theta condition.* If *comp* immediately dominates an *np*, that *np* must be coindexed with a *trace* that is not dominated by *comp*.

This first, simple *theta* condition is satisfied by any Tree if for every NP in Tree, if NP is dominated by *comp* then there is a coindexed *trace* in Tree that is not dominated by *comp*. If the arguments of each grammatical predicate uniquely specify a node in the tree, we can represent the constraint as follows: [3]

$$\forall Tree(\ theta(Tree) \leftarrow$$
$$\forall(\ parent(comp(L0, L1), np(F1, I1, L2, L3), Tree) \rightarrow$$
$$subtree(trace(I2, L4, L4), Tree) \wedge$$
$$\neg ancestor(comp(L5, L6), trace(I2, L4, L4), Tree) \wedge$$
$$coindexed(I1, I2)\ )\ )$$

The "∀" with no variables signifies the universal closure. A predication *ancestor(Node1, Node2, Tree)* is true just in case Node1 is an ancestor of Node2 in the tree Tree; in other words, Node1 dominates Node2 in Tree. The *parent* and *subtree* predicates also have the standard interpretation. The *coindexed* predicate is an equivalence relation, standard in linguistic theory [45].

The other constraints can be handled similarly. Notice that this formulation of the *theta* condition on trees is not a Horn clause: it is not a clause that can be used by Prolog. It contains

[3] The last two arguments of each grammatical predicate are introduced by the translation of the rewrite rule format into "definite clauses", as described in [31]. Prolog does this translation automatically.

universal quantifiers in its antecedent (i.e., its "body"), implication and negation. However, we have defined a transformation from this sort of representation to an equivalent "compiled" formulation that can be handled efficiently by Prolog (and by other left-to-right Horn clause theorem provers with the negation-as-failure rule). [4]

We can now formulate a logically correct representation of our parsing problem. In the most natural formulations, the constraints must apply to completed trees representing the proof that a string is an s. The most straightforward way to do this is to apply the transformation r to the grammar, and then embed the transformed grammar in another theory that contains the constraints and a predicate (*constrained_s*) which is satisfied only by completed trees that satisfy the constraints:

$$\begin{aligned} constrained\_s(L0, L, Tree) :- & \; s(L0, L, Tree), \\ & \; theta(Tree), \\ & \; trace\_bound(Tree), \\ & \; complex\_np(Tree) \end{aligned}$$

This definition of the parsing problem is correct and can be transformed into equivalent representations that can be efficiently used by left-to-right Horn clause theorem provers with negation-as-failure. The more efficient representations check partially built trees representing proofs of $s(L0, L)$ in order to cull derivations that do not satisfy the constraints at the earliest possible point. This is similar to what is done by "backtracking strategies" in other constraint satisfaction problems [23]. An example of this transformation to a very efficient representation is provided in Appendix 1.

Our strategy can really be seen as a quite direct generalization of the work of Pereira [32]. Pereira uses an "extraposition list" which is basically a concise representation of left context information and global constraints. We have just used two separate data structures to represent left context and global constraints rather than one. In our example, we also enforced more complex constraints on the derivations, and as a result our compiled approach does not run as efficiently as comparable XG grammars would, but it covers more of the language. Our approach is

[4] "Negation-as-failure" is a non-monotonic inference rule which says that we can infer $\neg\phi$ if there is no finite SLD proof of ϕ. See [18] for a discussion of this rule, and for proofs of the soundness and completeness of SLD-resolution augmented with this rule.

more easily extended to capture more sophisticated constraints on derivations, though, and our representation of those constraints is explicit and more intuitive.

Our logical representation of basic features of syntax makes the parsing problem very much like standard problems using "controlled deductions," that is, problems in which unnecessary search for proofs is eliminated. For example, it is well known that, without sacrificing completeness, standard resolution systems can terminate any derivation in which a clause is derived that is identical to an ancestor of that clause [37]. Rules of this sort can be trivially implemented with strategies of the sort described here. It is anticipated that this sort of control over deductions may be valuable in a range of problems.

In summary, the principle features of the design of the parsing algorithm are the following:

- basic phrase structure is computed using an extension of the Definite Clause Grammar framework
- "local" and relatively restricted principles of grammar are built into the phrase structure rules, and consequently enforced by the basic parser operation
- explicitly stated grammatical principles are folded into the parsing process if they will help prune the parser search space, otherwise they can be applied after the parse is complete without loss of efficiency

4.4. Rightmost Normal Form

It is well known that the ambiguity of English is one of the major difficulties for practical computer understanding systems. Most ambiguities are never noticed by fluent human speakers, but they are very difficult for a computer to resolve. For example, consider the "attachment" ambiguity in a sentence like:

Display the first rule of the procedure for append in window1.

A human speaker has no problem immediately interpreting this request to display a rule in a window, but a computer using the following simple grammar rules will find 9 different structures with 9 different interpretations:

```
vp --> v.
vp --> v, np, ppx.

np --> name_.
np --> det, opt_adj, n, ppx.

opt_adj --> [].
```

```
opt_adj --> [first].

ppx --> [].
ppx --> pp, ppx.

pp --> prep, np.

v --> [display].
name_ --> [append].
name_ --> [window1].
det --> [the].
n --> [procedure].
n --> [rule].
prep --> [in].
prep --> [of].
prep --> [for].
```

A computer does not have the common sense to realize, for example, that "in window1" does not modify "the rule" or "the procedure" but the verb "display".

When this "attachment" ambiguity is multiplied by other dimensions of ambiguity (lexical ambiguity, ambiguity in quantifier scope, etc.), sentences the same size as our example could have nearly a hundred readings. Building all of these representations and examining them is just not feasible.

Standard strategies for avoiding the ambiguity problem are available in small domains. We have chosen first, to build only one parse tree, but one from which any needed alternative structures can, in effect, be computed. This does not remove the problem of attachment ambiguities, but postpones it to the semantic analysis stage, allowing a more modular grammar.

The "rightmost normal form" constraint which allows only rightmost normal form parses to be formulated does not require any modification in the basic grammar rules. In fact, the grammar notation is not adequate to define the condition; but the condition is easily defined over derivation trees, and can then be compiled into an arbitrary grammar using the techniques illustrated in Appendix 1. This approach is similar to the approach used in the CHAT-80 system for processing queries about geography [30]. As the performance of that system shows, the enforcement of this sort of constraint allows parsing to produce very quickly a structural representation of the users query that can be subjected to efficient semantic analysis.

In the semantic processor, the ambiguity problem is handled by imposing very strict conditions on what can serve as arguments to each verb and on what can modify any particular noun or verb. These restrictions are imposed primarily on the basis of lexical specifications, i.e., on a

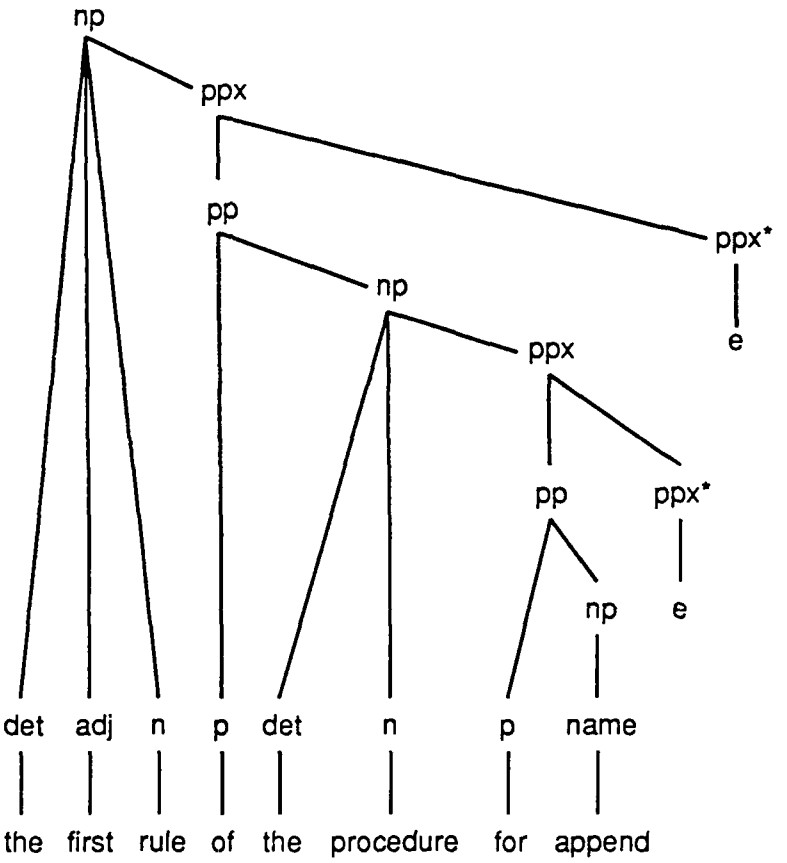


Figure 4.2. A Rightmost Normal Form Parse

word-by-word basis. We will briefly describe and illustrate the syntactic and semantic processing techniques.

As illustrated with the example above, it is computationally very expensive to build every acceptable complete parse tree for every input. Instead, we build a "rightmost normal form" which actually represents a set of parse trees. This normal form is quite close to the parse tree that is, in fact, often the preferred interpretation. However, in cases where it is not the appropriate structure, the semantic analyzer can immediately move to an alternative that is specified in the normal form. For example, the rightmost normal form for

the first rule of the procedure for append

can be represented as in Figure 4.2. In this representation, the prepositional phrases are attached as deeply in the tree as possible, but the alternative attachment points for the lower phrases easily be computed. (They are marked with asterisks.) The semantic analyzer will notice that these attachments are acceptable in this case. In processing our first example, an attempt would be made to interpret the phrase

the first rule of the procedure for append in window1.

This attempt will fail unless "in window1" can be attached to some higher node in the tree, viz., to a point where it is a modifier of the verb.

The whole task of using domain-specific information to determine the correct interpretation of an input is left to the semantic processor. This allows the domain-specific information to be isolated in one component, maximizing portability. The basic mechanism for handling the attachment ambiguities discussed above is to use "slots" which specify, on a word-by-word basis, specific restrictions on what can modify what. For example, the word "procedure" is associated with the following entry:

property(procedure, obj, Relation, rule(Proc,Relation), [slot(prep(for),obj,Relation)]).

This indicates that "procedure" is to be translated as a relation between two things, Proc and Relation, where Proc and Relation are both (abstract) objects, and where Relation is the object named by the object of a prepositional phrase beginning with "for". Clearly this translation for "procedure" is preferable, if only for reasons of efficiency, to the translation which treats "procedure" as an object that is described by the prepositional phrase beginning with "for":

procedure(X), for(X,append)

When we attempt to interpret a phrase like "the procedure for...", these restrictions on modifiers will be enforced. When a prepositional phrase cannot be interpreted as modifying the noun

phrase in which it is attached in the rightmost normal form, it is "passed up" the tree for interpretation at one of the other possible attachment sites.

4.5. Pronouns and the Discourse Model

A system that cannot resolve pronoun references cannot provide a comfortable interface for the user. The following simple queries, for example, require the interpretation of pronouns:

> What procedures call themselves?
>
> Is every computation with a goal identical to one of its ancestors nonterminating?
>
> Display the proof of $p(a)$. Which rules does it use?

The determination of the relationship between "themselves" and "procedures" in the first of these queries is dictated by syntax, but the appropriate interpretation of the pronouns in the latter two queries is not so simple. It is well known that resolving pronoun references properly in unrestricted English text requires essentially complete understanding of the text and relevant background knowledge, but fortunately there are very simple and efficient pronoun resolution algorithms that get better than 90% of the resolutions correct in technical writing [16]. Notice that the pronoun "it" in the last query shown above refers to the subject of the previous sentence. In cases like this a simple "discourse model" must be used to extend the sentence-bound syntactic processing.

Discourse analysis is conventionally treated as a problem that is quite separate from syntax and semantics, but as we noted above, this is not the approach of systemic grammar. And even in other traditions, it is becoming clear that discourse processing principles interact with other systems, and this interaction can be exploited in natural language processing. For example, the interpretation of noun phrases must take into account the information system distinction between "new" and "given" objects of discourse. In the case of descriptive noun phrases, this corresponds roughly to the syntactic distinction between definite and indefinite descriptions. An indefinite description like "a procedure" usually introduces an object to the discourse, as in "A procedure called itself. It was right recursive." The indefinite description introduces a new object which the listener is not presumed to know about yet, and this object is referred to by the later pronoun "it". A definite description like "the procedure" in the same context would indicate a reference to an object that is already introduced into the universe of discourse, and interpretive principles would be required to determine which object this was. This determination can influence even

syntactic processing, since sometimes the type of thing referred to can be used to eliminate spurious syntactic structures. Hence, in accord with the basic strategy stated in the last section, principles of discourse analysis are stated just like other principles of grammar and folded into the basic phrase structure parsing.

4.6. Logical Query Optimization

The queries formulated by the semantic component of the MQP will not in general be feasible, since the semantic processing should not have to take into account such things as the size of the relations being queried. An optimization step is essential. Domain-independent techniques for optimizing logical queries based on basic facts about the knowledge base being queried have been developed in a number of different contexts by the Principal Investigator of this project and others [46] [47] [43] [36] [29]. These can be deployed in the MQP to appropriately optimize queries according to the knowledge representation, whether it is a foreign database system, a text file, or a Prolog-internal database.

4.7. Summary of MQP Design Features

The MQP design proposed here has the potential to significantly improve student-IIS interaction by providing the portable natural language processing capabilities that will make the development of query processors for IISs feasible and cost-effective. The innovative features of this effort include the following main points:

- modular representations of the natural language component expressed in logic are very efficient when specialized optimization techniques are used
- domain-dependent features of the language processor can be isolated in specific components of the system, and tools can be developed to facilitate the tailoring of these components to new domains
- the representation of principles of discourse analysis can, to a large extent, be isolated from parsing and semantic processing, but they may be folded back into "earlier" processing by the optimization steps for substantial improvements in efficiency
- the basic MQP design has been shown feasible by earlier work and by Phase I prototyping efforts in which the novel features of the approach were tested

5. Additional IIS Software Issues

Because the emphasis of the present project is on improving the IIS-student interaction using a modular systemic parser, we have decided to base our Phase II efforts on the development of an IIS for a tutorial domain that has been studied and that appears promising. In Phase II, if funding is provided, a Prolog Tutor will be built to illustrate the capabilities of our IIS and MQP technology. This will allow us to make use of the university research on the Lisp Tutor, since the basic programming styles of Lisp and Prolog are quite similar. This choice of Prolog has other advantages as well. In the first place, some preliminary efforts on Prolog IISs have had some success [19] [20] [21] [11] [22]. And in the second place, the Prolog evaluation strategy is simple and intuitive, so effective interactive debugging methods have already been developed for it [34]. Sophisticated debugging tools of this kind have been implemented at Quintus. This previous work will allow us to focus our efforts on improving the ability of our IIS to interact with the user using our Modular Query Processor. Another advantage of this choice of domain is that Quintus Computer Systems has some of the world's leading experts in the development of Prolog systems, so there will be no shortage of Prolog programming expertise.

This choice of a tutorial domain for the development of the Phase II prototype is one in *which a sophisticated IIS will be feasible. Extensive documentation of both reference and tutorial* information has already been developed at Quintus Computer Systems, and so the representation of domain knowledge can be quite thorough and comprehensive. Furthermore, Quintus has also developed specialized, intelligent debugging tools that can be deployed in error diagnosis. With this technology, the student can be given much more helpful feedback on his performance on exercise material than current IISs provide. This most difficult aspect of IIS capabilities, reliable diagnosis of student errors, will be relatively tractable in this domain.

6. Conclusions

The principal features of the IIS and MQP designed in this Phase I project include the following:

- the improvement of user-IIS interaction will be achieved with a MQP that can handle a wide range of user queries appropriately
- the MQP will be highly modular so that development and maintenance costs will be reduced, and so that many components of the system will be domain-independent and transportable

- the MQP will be expressed in Prolog, and efficiency will be obtained without sacrificing modularity through the use of specialized optimization techniques
- IIS capabilities will include tools for sophisticated domain representation and error diagnosis, as well as for efficient access to catalogued reference material

By implementing this system in the general-purpose language Prolog, a great deal of flexibility is afforded in rapidly prototyping a system. The efficiency of Quintus Prolog ensures that a prototype is very likely to be usable in a production-level environment as well. All of the features mentioned in our Phase I proposal seem to be feasible. Quintus intends to follow up with a proposal for Phase II funding to further research and prototype the IIS and MQP products.

References

[1] Anderson, J.R. & Swarecki, E. (1986) The automated tutoring of introductory computer programming. *Communications of the ACM 29*(9):842-849.

[2] Bates, M. (1978) The theory and practice of augmented transition network grammars. In L. Bolc, ed., *Natural Language Communication with Computers, Lecture Notes in Computer Science, 63.* New York: Springer-Verlag.

[3] Benson, J.D. and Greaves, W.S. (1985) *Systemic Perspectives on Discourse, Volumes 1-2.* Norwood, New Jersey: Ablex.

[4] Bowen, K.A. (1985) Using Prolog to build expert systems. *Expert Systems and Prolog*, IEEE Videoconference, December 4, 1985.

[5] Bowen, K.A. and Kowalski, R. (1982) Amalgamating language and metalanguage in logic programming. In *Logic Programming*, edited by K.L. Clark and S.-A. Tarnlund, NY: Academic Press, 153-172.

[6] Brown, J.S., Burton, J.R. and de Kleer, J. (1982) Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II and III. In Sleeman, D. and Brown, J.S., eds. (1982).

[7] Butler, C.S. (1985) Discourse systems and structures and their place within an overall systemic model. In [3].

[8] Codd, E.F. (1974) Seven steps to rendezvous with the casual user. Technical Report RJ 1333, IBM Research Laboratory, San Jose, California.

[9] Codd, E.F., Arnold, R.S., Cadious, J.M., Chang, C.L. and Roussopoulos, N. (1978) RENDEZVOUS Version 1: An experimental English language database query formulation system for casual users of relational databases. Technical Report RJ 2144, IBM Research Laboratory, San Jose, California.

[10] Fitzpatrick, E., Bachenko, J., and Hindle, D. (1986) The status of telegraphic sublanguages. In Grishman, R. and Kittredge, R., eds., *Analyzing Language in Restricted Domains: Sublanguage Description and Processing.* Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers.

[11] Fogel, E. (1982) *A Prolog Tutor.* M.Sc. Thesis, University of British Columbia.

[12] Goebel, R., Furukawa, K. and Poole, D. (1986) Using definite clauses and integrity constraints as the basis for a theory formation approach to diagnostic reasoning. *Procs. of the 3rd Int. Conf. on Logic Programming*, edited by E. Shapiro, pp.211-222. New York: Springer-Verlag.

[13] Grosz, B.J., Jones, K.S. and Webber, B.L. (1986) *Readings in Natural Language Processing.* Los Altos, California: Morgan Kaufmann.

[14] Halliday, M.A.K. (1976) *Halliday: System and Function in Language.* Edited by G.R. Kress. London: Oxford University Press.

[15] Halliday, M.A.K. and Martin, J.R., eds. (1981) *Readings in Systemic Linguistics.* London: Batsford Academic and Educational Ltd.

[16] Hobbs, J.R. (1978) Resolving pronoun references. *Lingua, 44*: 311-338.

[17] Kasper, R. (1987) "A unification method for disjunctive feature descriptions." Presentation to the Center for the Study of Language and Information, Stanford University, March 11, 1987.

[18] Lloyd, J.W. (1984) *Foundations of Logic Programming.* New York: Springer-Verlag.

[19] Lynch, L. (1986) A thesis proposal for a computer program to teach Prolog. University of Edinburgh Department of Artificial Intelligence Discussion Paper 10.

[20] Lynch, L. (1986) Finding bugs in Prolog code. University of Edinburgh Department of Artificial Intelligence Note 308.

[21] Lynch, L. (1986) A first attempt at building an analyser for Prolog. University of Edinburgh Department of Artificial Intelligence Note 277.

[22] Maler, O., Z. Scherz & E. Shapiro (1986) A new approach for introducing Prolog to naive users. Weizmann Institute of Science Technical Report CS86-03.

[23] Mackworth, A. (1985) Constraint satisfaction. Forthcoming in the *Encyclopedia of Artificial Intelligence.*

[24] Mann, W.C. (1982) The anatomy of a systemic choice. University of Southern California, Information Sciences Institute, Report No. ISI/RR-82-104, Government Accession No. AFOSR-TR-83-0135.

[25] Mann, W.C. (1983) Inquiry semantics: a functional semantics of natural language grammar. University of Southern California, Information Sciences Institute, Report No. ISI/RS-83-8, Government Accession No. A135153

[26] Mann, W.C. (1985) An introduction to the Nigel text generation grammar. In [3].

[27] Mann, W.C. and Matthiessen, C.M.I.M (1985) Demonstration of the Nigel text generation computer program. In [3].

[28] Matthiessen, C.M.I.M. (1984) Systemic grammar in computation: the Nigel case. University of Southern California, Information Sciences Institute, Report No. ISI/RS-83-121, Government Accesion No. AD-A139351.

[29] Naish, L. (1985) Automating control for logic programs. *Journal of Logic Programming, 3*: 167-183.

[30] Pereira, F.C.N. (1982) *Logic for Natural Language Analysis,* Ph.D. thesis, University of Edinburgh, Scotland.

[31] Pereira, F.C.N. and Warren, D.H.D. (1980) Definite clause grammars for natural language analysis. *Artificial Intelligence, 13*: 231-278.

[32] Pereira, F.C.N. and Warren, D.H.D. (1983) Parsing as deduction. *Procs. of the 21st Ann. Mtg. of the Assoc. for Computational Linguistics*, 137-144.

[33] Quintus Computer Systems, Inc. (1987) *Quintus Prolog User's Guide, with Release Notes for Quintus Prolog 2.0 Beta-Test.* Quintus Computer Systems, 1310 Villa Street, Mountain View, CA, 94041.

[34] Shapiro, E. (1983) *Algorithmic Program Debugging.* Cambridge, Massachusetts: MIT Press.

[35] Sleeman, D. and Brown, J.S., eds. (1982) *Intelligent Tutoring Systems.* New York: Academic Press.

[36] Smith, D.E. and Genesereth, M.R. (1985) Ordering conjunctive queries. *Artificial Intelligence, 26*: 171-215.

[37] Smith, D., Genesereth, M. and Ginsberg, M. (1985) Controlling recursive inference. Technical Report STAN-CS-85-1063, Stanford University, Palo Alto, California.

[38] Soloway, E. (1986) Learning to program = learning to construct mechanisms and explanations. *Communications of the ACM 29*(9):850-858.

[39] Soloway, E. and Iyengar, S. (1986) *Empirical Studies of Programmers*. New York: Ablex.

[40] Stabler, E.P., Jr. (1986) Restricting logic grammars. *Procs. of the 5th Nat. Conf. on Artificial Intelligence, AAAI-86*, 1048-1052.

[41] Stabler, E.P., Jr. (1986) Parsing with government-binding constraints. *1986 Annual Meeting of the Artifical Intelligence and Robotics Program of the Canadian Institute for Advanced Research*, Banff, Alberta, 1986.

[42] Stabler, E.P., Jr. (1987) Restricting logic grammars with government-binding theory. Forthcoming in *Computational Linguistics*.

[43] Stabler, E.P., Jr. and Elcock, E.W.E. (1983) Knowledge representation in an efficient deductive inference system. *Procs. of the Logic Programming Workshop '83*.

[44] Sterling, L. and Shapiro, E. (1986) *The Art of Prolog*. Cambridge, Massachusetts: MIT Press.

[45] van Riemsdijk, Henk and Williams, Edwin (1986) *Introduction to the Theory of Grammar*. Cambridge, Massachusetts: MIT Press.

[46] Warren, D.H.D. (1981) Efficient processing of interactive relational database queries expressed in logic. *Procs. 7th Int. Conf. on Very Large Databases*.

[47] Warren, D.H.D. (1982) Issues in natural language access to databases from a logic programming perspective. *Procs. of the 20th Ann. Mtg. of the Assoc. for Computational Linguistics*:63-66.

[48] Warren, D.H.D. and Pereira, F.C.N. (1982) An efficient, easily adaptable system for interpreting natural language queries. *Amer. J. of Computational Linguistics*, 110-119.

[49] Webber, B.L. (1986) So what can we talk about now? In [13].

[50] Winograd, T. (1976) *Understanding Natural Language*. New York: Academic Press.

[51] Winograd, T. (1983) *Language as a Cognitive Process, Volume I: Syntax*. Reading, Massachusetts: Addison-Wesley.

Appendix 1. Compiling Parser Representations

In the text we provide a definition of a parsing problem in terms of rewrite rules and constraints. That definition is correct but must, for practical purposes, be transformed into a more efficient but equivalent representation. The original formulation would be terribly inefficient with any left-to-right proof strategy: we do not want to have to go through proving that a string is an *np* followed by a *vp* and then rule out the derivation because of a violation of the complex-NP constraint in the first *np*. Since the constraints really rule out lots of derivations which would otherwise be allowed by the context free component, we want to apply those constraints as soon as possible. We will describe a two-step process for getting to an efficient "compiled" representation. First we describe how to cull unsatisfactory parses as soon as possible with a specialized metainterpreter, and then we describe how to eliminate the overhead of metainterpretation. Throughout these steps, our first order formulation can be used as an intuitive specification of the problem to be solved.

The Metainterpretation Strategy. It is a simple matter to define a basic "metainterpreter" or provability predicate for Horn clause theories. The following is a standard formulation of a Prolog metainterpreter:

$$demo(true).$$
$$demo((P,Q)) :- demo(P), demo(Q)$$
$$demo(P) :- clause(P, Body), demo(Body)$$

It is easy to modify this simple provability predicate so that checks at each step to make sure that the constraints are satisfied:

$$demo(true).$$
$$demo((P,Q)) :- demo(P), demo(Q)$$
$$demo(P) :- clause(P, Body),$$
$$\qquad demo(Body),$$
$$\qquad satisfy\_constraints(P)$$

This new strategy requires (i) a different representation of the derivations, one which is available and efficient at every point in the proof, and (ii) a reformulation of the constraints themselves so that they can be applied at arbitrary points in the computation.

Formalizing derivation left-contexts. In the course of a left-to-right proof, we need to be able to inspect the parts of the proof tree that are already completed. In general, a constraint applied at any point in the proof tree should be able to test its left siblings, its ancestors, and left siblings of its ancestors, i.e., its "left context". The transformation τ which was introduced in the text of this report is not feasible for this use. Another problem with the representation of the proof provided by τ is that it makes the root node and its daughters most accessible. Since we would like to apply the constraints at the earliest possible point, we will typically be interested not in the root node, but in the left siblings and ancestors of the current node. Fortunately, there are a number of well-known tree representations that do not have these problems. A reversed, sequential representation of the proof tree will allow the left context to be built up incrementally, and it will make the current node most accessible.

We will move to a specialized representation below, but we can get a version of a simple reversed "preorder sequential representation" by simply flattening our original representations, introducing special terms that will serve as structural indicators with the obvious meaning, and reversing the result. Thus we represent $p/[q/[s],r]$ as $[\ [\ ,r,\ [\ ,s,\ ]\backslash\ ,q,\ ]\backslash\ ,p]$.

Since we want to build our trees in the course of doing a proof, we will need a slightly more complex program transformation: the left context representation when proving some subgoal should not be the same as the left context when proving the next subgoal. So we use a pair of arguments to represent the left context before the proof and the left context after the proof, respectively. We can define a transformation τ_1 that will do this. A proof begins with with a negative clause, and so such clauses will have the empty left context as their initial representation, and a variable to be instantiated with the completed proof tree. Consider our previous example, repeated again for convenience, and our transformation of it:

$$
\begin{aligned}
S = \{ \quad & :\!-p, \\
& p:\!-q, r, \\
& q:\!-s, \\
& r, \\
& s \quad \}
\end{aligned}
$$

$$\tau_1(S) = \{ \quad :-p([], Proof),$$
$$p(Proof0, [\ |Proof]):-q([\ |\backslash\ , p|Proof0], Proof1), r(Proof1, Proof),$$
$$q(Proof0, [\ |Proof]):-s([\ |\backslash\ , q|Proof0], Proof),$$
$$r(Proof, [r|Proof]),$$
$$s(Proof, [s|Proof]) \quad \}$$

At every point in a left-to-right proof in this theory, we are provided with a reversed sequential representation of the left context with the most recently completed nodes or structural indicators at the front. Notice that these left context representations do still correspond to trees, but the trees correspond to initial segments of derivations rather than to completed derivations.

Clearly there is definition of τ_1 that will provide our required features: noninterference and representational correctness.

Formalizing constraints for early application Formalizing the constraints so that they will be applicable at arbitrary points in the proof is not trivial. The basic idea is straightforward: we represent the constraints in terms of conditions on right and/or left contexts, testing the left context immediately, and providing a special mechanism that will pass constraints on right context to the right. Let's consider our three simple constraints.

Notice that our first two constraints, the theta condition and the trace-binding condition, in their original first-order formulation, say that if there is a node of a certain kind, then there must be another node of a certain kind. So we can use the following strategy: test every node at completion; if it is of the kind mentioned in the antecedent of one of these constraints, then check the left context for the other required node; if it is not in the left context, then pass the existence condition to the right; and ensure that all conditions passed to the right are satisfied before the parse is complete. The third constraint, on the other hand, sometimes introduces a requirement that must be satisfied by every node in the tree. For this type of constraint we can test every node at completion; if it is of the kind mentioned in the antecedent of this third constraint, check the left context immediately, and if the left context does not already violated the condition, pass the global constraint to the right. This means that we can have two different types of constraints passed to the right: existence requirements that can be removed as soon as possible after they have been satisfied; and conditions on every node that must be carried throughout the parse and applied as soon as possible after every point at which a violation may occur.

This approach requires that the *demo* predicate be able to carry constraints to the right. A simple approach distinguishes "local constraints" which are applied at every node from "global

constraints" which may be collected in the course of the parse and "passed to the right." We can distinguish the two different kinds of constraints in the global constraint lists by placing them in one of the two forms: *exists*(*Constraint*) and *all*(*Constraint*): the former constraints can be removed as soon as they are satisfied; the latter must apply everywhere. After satisfying each local constraint, we can try to satisfy any of the global existence constraints, and then we can impose the requirement that no unsatisfied existence constraints remain at the end of the parse.

The following *demo* predicate captures this approach:

$$demo(P) :- demo(P, [], G),$$
$$\quad global\_constraints\_satisfied(G).$$

$$demo(true, G, G).$$
$$demo((P, Q), G0, G) :- demo(P, G0, G1), demo(Q, G1, G).$$
$$demo(P, G0, G) :- clause(P, Body),$$
$$\quad demo(Body, G0, G1),$$
$$\quad satisfy\_local\_constraints(P, G1, G).$$

$$global\_constraints\_satisfied(G) :- \neg member(exists(\_), G).$$

$$satisfy\_local\_constraints(P, G0, G) :- theta(P, G0, G1),$$
$$\quad trace\_binding(P, G1, G2),$$
$$\quad complex\_np(P, G2, G3),$$
$$\quad try\_global\_constraints(P, G3, G).$$

Now let's consider how we might formulate each of the "local" linguistic constraints within this approach. Again, we will concentrate on the *theta* condition:

$$\forall Tree(\ theta(Tree) \leftarrow$$
$$\forall(\ parent(comp(L0, L1), np(F1, I1, L2, L3), Tree) \rightarrow$$
$$\quad subtree(trace(I2, L4, L4), Tree) \wedge$$
$$\quad \neg dominates(comp(L5, L6), trace(I2, L4, L4), Tree) \wedge$$
$$\quad coindexed(I1, I2)\ )\ )$$

We assume that a transformation like τ_1 has applied to add two arguments to every grammatical predicate. These arguments represent the proof before and after the predication is proven, respectively. Then a first reformulation for our strategy of testing at every node is the following:

$$theta(np(\_, I1, \_, \_, \_, LC), G0, G) :- parent(LC, comp(\_, \_, \_, \_)),$$
$$left\_or\_right\_trace(I1, LC, G0, G).$$
$$theta(P, G, G) :- \neg P = np(\_, \_, \_, \_, \_, \_).$$

$$left\_or\_right\_trace(I1, LC, G, G) :- precedes(LC, trace(I2, \_, \_, \_, \_), TraceLC),$$
$$\neg ancestor(TraceLC, comp(\_, \_, \_, \_), \_),$$
$$coindexed(I1, I2).$$
$$left\_or\_right\_trace(I1, \_, G, [exists(trace(I1))|G]).$$

The first rule for *left_or_right_trace* checks the left context for the required trace; the second rule passes the requirement to the right by adding the *exists(trace(I))* condition to the global constraint list G. This indicates that an appropriate trace must have its index bound to I before the parse is completed.

The other constraints can be handled similarly.

Specializing the left contexts. Building and testing complete representations of left context is an unnecessary computational burden if we can show that there are parts of these trees which no constraint will ever test. Ideally, we would like a provably adequate but minimal representation of the proof, relative to the grammar and the constraints that will actually be imposed. Notice that since testing the trees for satisfaction of the constraints may be much more expensive than the construction of the trees, it may be worthwhile to keep the minimal representation of the left context for use in these tests even if a complete representation must also be built for other purposes. We accordingly use a transformation τ_2 which adds three extra arguments to each grammatical predicate: the argument that would be added by τ and the pair of arguments that would be added by τ_1, except that we now subject the latter pair of arguments to specialization. The first argument builds a representation of the completed tree as the output of the parser; the latter pair of arguments holds just enough left context to be able to enforce all of the constraints.

We can reduce the representations of left context in the following ways: unneeded nodes can be removed from the tree, and unneeded parts of node labels can be removed. For our grammar

and constraints, we can remove (i) all leaves except *trace*(*Index*); (ii) all *vp* and *s* nodes, and (iii) all *np* nodes that dominate the single node *name*, or the pair of nodes *det*, *n*, or the triple of nodes *det*, *n*, *sbar*. A thorough justification of this reduction in the left context can be provided.

The metainterpretation approach has real advantages. It preserves complete modularity in the rewrite rules and the various constraints. This is valuable for getting the constraints implemented correctly before specializing the left context representations, constraint applications, etc. And this approach is much more efficient than the "naive" representation.

The Compilation Approach The overhead expense of metainterpretation is unnecessary. The role of the metainterpreter is (i) to apply the local constraints after each grammatical predication is proven, (ii) to carry the global constraint list through the proof, and (iii) to apply the global constraints upon completion of the grammatical proof. All of these tasks can be carried out without metainterpretation. We describe one very straightforward way to do this.

We can easily achieve (i) without metainterpretation. We can simply apply the local constraints as a final condition in each rule of the grammar. But we can do better than this by noting that the local constraints have nontrivial application only when we have used a rule that expands a node in a position that could satisfy the antecedent of one of the constraints. In our example, there are only three such rules: the rule that expands *trace* to the empty string; the rule that expands *np* to *trace*, and the rule that expands *np* to *rel_pro*. We can add the condition

$$satisfy\_local\_constraints(Goal, G0, G)$$

to just these three rules, where *Goal* is the head of the rule itself, *G*0 is the "previous" list of global constraints, and *G* is the new list of global constraints.

While this approach suffices, it is easy to do even better. We know which constraint has the possibility of an active role at each of these three kinds of nodes: the *trace_binding* condition applies only to *trace* nodes, and none of the other constraints apply there. At the *np* nodes, both the *theta* condition and the *complex_np* condition may be relevant. So we can condition the three rules accordingly: the rule expanding *trace* is conditioned only by *trace_binding*, and the other two rules can be conditioned only by the *theta* and *complex_np* constraints.

The remaining roles of the metainterpreter are also easy to cover. We can accomplish (ii) by using a pair of lists to carry the global constraints from one grammatical predicate to the next, in the way we have used the pairs of lists to hold the string to be parsed (as DCG's do also) and

the representation of left context. And finally, we can accomplish (iii) by introducing a predicate that will check the global constraints after the grammatical proof is complete:

$$constrained\_s(L0, L, Tree, LC0, LC, G0, G) :- s(L0, L, Tree, LC0, LC, G0, G),$$
$$satisfy\_global\_constraints(G).$$

The constraints can then remain exactly as they were for the metainterpretation strategy, and the resulting system runs much faster. A listing of the compiled representation of our example is provided in Appendix 2.

Automating the Transformation to "Compiled" Form. It is clear that substantial parts of the transformation from the natural first-order representation to "compiled" form can be automated. We have in fact developed such techniques.

Given a first order formulation of the rewrite rules (the grammar) and of the constraints, we can get to a compiled formulation as follows:

(a) reformulate the constraints so that they have the form of conditions on right and left contexts, optimizing these formulations where possible; testing with metainterpretation;

(b) specialize the representations of left context based on an analysis of the grammar and the constraints formulated by (a);

(c) using an analysis of the grammar and the constraints formulated by (a), determine which constraints should apply at which nodes;

(d) transform the grammar:

- add 5 arguments to each predicate in the grammar: an argument holding the standard parse tree, a pair of arguments to build the specialized representation of left context according to (b), and a pair of arguments to carry any global constraints which are added by local constraint application;
- condition the relevant rules according to the results of (c).

Some progress has been made in automating this "compilation" step, though some parts of the transformation will be very difficult to automate satisfactorily. We have preliminary implementations of (b), (c), and (d). The output of the transformation is guaranteed to have the basic features of the trees and left contexts built properly, and this output can always be subjected to further modification by a competent logic programmer.

Appendix 2. A Listing and Session with a "Compiled" Parser

We provide a listing of Quintus Prolog code, followed by a brief, edited session log that shows this code running after being compiled by Quintus Prolog Version 2.0 and running on a SUN 3/50.

```
% THE GRAMMAR - this is the output of the "compiler"
constrained_s(L0,L,Tree,Proof0,Proof,G0,G) :-
   s(L0,L,Tree,Proof0,Proof,G0,G),
   global_constraints_satisfied(G).

s(L0,L,s/[NP,VP],Proof0,Proof,G0,G) :-
   np(_,_,L0,L1,NP,Proof0,Proof1,G0,G1),
   vp(L1,L,VP,Proof1,Proof,G1,G).

np(-wh,Index,L0,L,np(Index)/[Name],Proof0,Proof,G0,G) :-
   name(L0,L,Name,Proof0,Proof,G0,G).
np(-wh,Index,L0,L,np(Index)/[Det,N],Proof0,Proof,G0,G) :-
   det(L0,L1,Det,Proof0,Proof1,G0,G1),
   n(L1,L,N,Proof1,Proof,G1,G).
np(-wh,Index,L0,L,np(Index)/[Det,N,Sbar],Proof0,Proof,G0,G) :-
   det(L0,L1,Det,Proof0,Proof1,G0,G1),
   n(L1,L2,N,Proof1,Proof2,G1,G2),
   sbar(L2,L,Sbar,Proof2,Proof,G2,G).
np(_,Index,L0,L,np(Index)/[Trace],Proof0,[down|Proof],G0,G) :-
   trace(Index,L0,L,Trace,[up,np(Index)|Proof0],Proof,G0,G1),
   theta(np(_,Index,_,_,_,_,Proof0,[down|Proof]),G1,G2),
   complex_np(np(_,Index,_,_,_,_,Proof0,[down|Proof]),G2,G).
np(+wh,Index,L0,L,np(Index)/[Rel_pro],Proof0,[down|Proof],G0,G) :-
   rel_pro(L0,L,Rel_pro,[up,np(Index)|Proof0],Proof,G0,G1),
   theta(np(_,Index,_,_,_,_,Proof0,[down|Proof]),G1,G2),
   complex_np(np(_,Index,_,_,_,_,Proof0,[down|Proof]),G2,G).
trace(Index,L,L,trace/[e],Proof,[trace(Index)|Proof],G0,G) :-
   trace_binding(trace(Index,_,_,_,_,Proof,_),G0,G1),
   try_global_constraints(trace(Index,_,_,_,_,Proof,_),G1,G).

sbar(L0,L,sbar/[Comp,S],Proof0,[down|Proof],G0,G) :-
   comp(L0,L1,Comp,[up,sbar|Proof0],Proof1,G0,G1),
   s(L1,L,S,Proof1,Proof,G1,G).

comp(L0,L,comp/[NP],Proof0,[down|Proof],G0,G) :-
   np(+wh,_,L0,L,NP,[up,comp|Proof0],Proof,G0,G).
```

```
vp(L0,L,vp/[Verb,NP],Proof0,Proof,G0,G) :-
   verb(L0,L1,Verb,Proof0,Proof1,G0,G1) ,
   np(_,_,L1,L,NP,Proof1,Proof,G1,G) .

name([mary|L],L,name/[mary],Proof,Proof,G,G).
det([the|L],L,det/[the],Proof,Proof,G,G).
n([man|L],L,n/[man],Proof,Proof,G,G).
rel_pro([who|L],L,rel_pro/[who],Proof,Proof,G,G).
verb([likes|L],L,verb/[likes],Proof,Proof,G,G).

% THE CONSTRAINTS

global_constraints_satisfied(G) :-
   \+member(exists(_),G).

try_global_constraints(Goal,[exists(Constraint)|Gs],G):-
   exists(Goal,Constraint),
   try_global_constraints(Goal,Gs,G).
try_global_constraints(Goal,[Constraint|Gs],[Constraint|G]) :-
   try_global_constraints(Goal,Gs,G).
try_global_constraints(_,[],[]).

% Theta: If current node is np, and current is immediately dominated by comp,
%   then current must be coindexed with a trace that is not dominated
%   by comp.
theta(Goal,G,G) :- \+Goal=np(_,_,_,_,_,_,_).
theta(np(_,_,_,_,_,_,LC,_),G,G) :-
   \+parent(LC,comp,_).
theta(np(_,I1,_,_,_,_,LC,_),G0,G) :-
   parent(LC,comp,ParentLC),
   left_or_right_trace(I1,ParentLC,G0,G).

% There must be either a preceding trace not dominated by comp, or
%  a following trace not dominated by comp. We catch these two
%  cases with the following two rules, respectively.
% A trace not dominated by comp leaves a_bound(Index) in the Global
%  constraint list.
left_or_right_trace(I1,LC,G,G) :-
   preceding_node(LC,trace(I2),TraceLC),
   \+ancestor(TraceLC,comp,_),
   coindexed(I1,I2).
% We add exists(trace(Index)) to the global constraints to signal that Index
%  needs to be bound to a trace in A position
```

```
left_or_right_trace(I1,_,G,[exists(trace(I1))|G]).

coindexed(I,I).

% We look for the exists(trace(Index)) requirement of a preceding np,
%  check to see if we have bound its index, and whether the current
%  node is in A position (i.e., not dominated by comp)
% We do this every time a new trace is created - if this requirement
%  cannot be satisfied at any of those points, it cannot be satisfied
exists(trace(Index,_,_,_,TraceLC,_),trace(I)) :-
   I==Index, \+ancestor(TraceLC,comp,_).

% trace-binding: a trace must be coindexed with a preceding np that is
%  immediately dominated by comp
trace_binding(Goal,G,G) :- \+Goal=trace(_,_,_,_,_,_).
trace_binding(trace(Index,_,_,_,LC,_),G,G) :-
   preceding_node(LC,np(Index),NpLC),  % indices unified here
   parent(NpLC,comp,_).

% complex-NP:An np dominated by sbar cannot be coindexed with an np that
%  is not dominated by sbar.
% These non-co-reference conditions apply to both the right and to the
%  left, so we add a global constraint with the Index and a term
%  that is not shared by any two distinct sbars (viz, the left context):
%     restrict_to_sbar(Index,SbarLeftContext)
complex_np(np(_,Index,_,_,_,NpLC,LC),GO,G):-
   first_ancestor(NpLC,sbar,SbarLC),% the first sbar suffices
   G=[all(restrict_to_sbar(Index,SbarLC))|GO],
   \+exists_cnp_violation(LC,G).
complex_np(np(_,_,_,_,_,_,NpLC,LC),G,G):-
   \+ancestor(NpLC,sbar,_),
   \+exists_cnp_violation(LC,G).
complex_np(Goal,G,G):-
   \+Goal=np(_,_,_,_,_,_,_,_).

% To find a violation we must find two indices which are coindexed (by
%  one of the other principles). The test "coindexed(I1,I2)" will
%  not make this distinction, because all indices are vars, and any
%  two vars can be unified in such a test. So we get the desired
%  effect by using "\+anti_index(I1,I2)" instead.
% We have a cnp violation if there is an Index1 restricted to SbarLC
%  and (i) there is no first sbar ancestor of the current NP
```

```
% yet Index1 and the current index are already coindexed, or
% (ii) there is a first sbar ancestor which is different from the
% one associated with index1 and yet Index1 and current index
% are already coindexed.
exists_cnp_violation(LC,G) :-
   member(all(restrict_to_sbar(Index1,SbarLC)),G),
   current_node(LC,np(Index2),NpLC),
   ( \+first_ancestor(NpLC,sbar,SbarLC2)
   ; first_ancestor(NpLC,sbar,SbarLC2),
     \+SbarLC=SbarLC2
   ),
   \+anti_index(Index1,Index2).

anti_index(0,1).

% THE UTILITIES

current_node([down|Rest],CurrentNode,Remainder) :- !,
   revtree([down|Rest],CurrentNode/_,Remainder).
current_node([],unlabelled,[]).

% For any node with left context LC, Parent is the imm dominating node with
% left context Remainder
parent([up,DomNode|Remainder],DomNode,Remainder) :- !.
parent([down|Rest],DomNode,Remainder) :-
   revtree([down|Rest],_,Remainder1),
   parent(Remainder1,DomNode,Remainder).
parent([],unlabelled,[]).

% For any node with left context LC, Ancestor is a dominating node with
% left context Remainder
ancestor([],Ancestor,Remainder) :- !, Ancestor=unlabelled, Remainder=[].
ancestor(LC,Ancestor,Remainder) :-
   parent(LC,Parent,Remainder1),
   ( Ancestor=Parent,
     Remainder=Remainder1
   ; ancestor(Remainder1,Ancestor,Remainder)
   ).

% For any node with left context LC, Ancestor is the first ancestor with
% left context Remainder
% (Typically used with LC and Ancestor instantiated)
first_ancestor(LC,Ancestor,Remainder) :-
   ancestor(LC,Ancestor,Remainder), !.
```

```
% For any node with left context LC, Node is a preceding node with
%  left context Remainder
preceding_node([down|Rest],Node,Remainder) :-
   left_sib(Rest,Node,Remainder).
% Ancestors (i.e., the anon var) do not count as preceding nodes
preceding_node([up,_|Rest],PrecedingNode,Remainder):-
   preceding_node(Rest,PrecedingNode,Remainder).

left_sib([up|Rest],Node,Remainder) :- !,
   domcat(Rest,Node,Remainder).
left_sib([down|Rest],Node,Remainder) :- !,
   preceding_node([down|Rest],Node1,Remainder1),
   (  Node1=Node,
      Remainder1=Remainder
   ;  left_sib(Remainder1,Node,Remainder)
   ).
left_sib([Sib|Rest],Node,Remainder) :-
   (  Sib=Node,
      Rest=Remainder
   ;  left_sib(Rest,Node,Remainder)
   ).

% revtree(RTree,Tree,[]) transforms a completed, reversed sequential RTree
%  into our standard nested Tree
revtree([down|Rest],Domcat/ListSibs,Remainder) :-
   collect_sibs(Rest,[],ListSibs,Remainder1),
   domcat(Remainder1,Domcat,Remainder).

collect_sibs([up|Rest],List,List,Rest) :- !.
collect_sibs([down|Rest],List0,List,Remainder) :- !,
   revtree([down|Rest],Tree,Remainder1),
   collect_sibs([Tree|Remainder1],List0,List,Remainder).
collect_sibs([Sib|Rest],List0,List,Remainder) :-
   collect_sibs(Rest,[Sib|List0],List,Remainder).

domcat([Domcat|Rest],Domcat,Rest).

% append and select are given the following defs at initialization
% append([],X,X).
% append([A|L],M,[A|N]) :- append(L,M,N).
% select(X,L,R) :- append(L1,[X|L2],L), append(L1,L2,R).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The following is an edited session running the code listed above, with the addition of i/o routines to prettyprint the structures formulated.

```
| ?- constrained_s([likes],[],_,[],_,[],_).

no
| ?- constrained_s([mary,likes,the,man],[],_,[],_,[],_).
s
   np(A)
      name mary
   vp
      verb likes
      np(B)
         det the
         n man

[]
0ms

yes
| ?- constrained_s([the,man,who,mary,likes,likes,mary],[],_,[],_,[],_).
s
   np(A)
      det the
      n man
      sbar
         comp
            np(B)
               rel_pro who
         s
            np(C)
               name mary
            vp
               verb likes
               np(B)
                  trace e
   vp
      verb likes
      np(D)
         name mary

[down,down,trace(_1926),up,np(_1926),down,down,up,np(_1926),up,comp,up,sbar]
17ms

yes
| ?- constrained_s([the,man,who,likes,mary,likes,mary],[],_,[],_,[],_)
s
   np(A)
      det the
```

```
      n man
      sbar
        comp
          np(B)
            rel_pro who
        s
          np(B)
            trace e
          vp
            verb likes
            np(C)
              name mary
    vp
      verb likes
      np(D)
        name mary

[down,down,trace(_1926),up,np(_1926),down,down,up,np(_1926),up,comp,up,sbar]
17ms

yes
| ?- constrained_s([the,man,who,mary,likes,mary,likes],[],_,[],_,[],_).

no
| ?- constrained_s([the,man,who,mary,likes,mary,likes,mary],[],_,[],_,[],_).

no
| ?-
```

END

5-87

DTIC