A SCHEMA & CONSTRAINT-BASED REPRESENTATION TO UNDERSTANDING NATURAL LANGUAGE

by

Eliza Wing-Mun Kuttner

Technical Report 87-2

January 1987

A SCHEMA & CONSTRAINT-BASED REPRESENTATION TO UNDERSTANDING NATURAL LANGUAGE

By

ELIZA WING-MUN KUTTNER

B.Sc., University of British Columbia, 1983

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES
(DEPARTMENT OF COMPUTER SCIENCE)

We accept this thesis as conforming
to the required standard

[illegible]

[illegible]

THE UNIVERSITY OF BRITISH COLUMBIA

November 1986

Abstract

This thesis attempts to represent the syntax and semantics of English sentences using a schema and constraint-based approach. In this approach, syntactic and semantic knowledge that are represented by schemata are processed in parallel with the utilization of network consistency techniques and an augmented version of Earley's context-free parsing algorithm. A sentence's syntax and semantics are disambiguated incrementally as the interpretation proceeds left to right, word by word. Each word and recognized grammatical constituent provide additional information that helps to guide the interpretation process.

It is desirable to attempt to apply network consistency techniques and schema-knowledge representations on understanding natural language since the former has been proven to be quite efficient and the latter provides modularity in representing knowledge. In addition, this approach is appealing because it can cope with ambiguities in an efficient manner. Multiple interpretations are retained if ambiguity exists as indicated by the words processed so far. However, incorrect interpretations are eliminated as soon as their inappropriateness is discovered. Thus, backtracking search which is known to be inefficient is avoided.

[illegible]

Contents

List of Figures

List of Tables

Acknowledgement

I would like to thank my supervisor, Bill Havens, for research ideas, help, support and patience. My thanks also to the Natural Sciences and Engineering Research Council of Canada and the Department of Computer Science at U.B.C. for financial support. I would also like to thank all my friends at the Department for making my years of graduate studies an enjoyable as well as educational experience. Special thanks to Michael, Alex, Lisa and my second reader, Richard Rosenberg, for their help in correcting this thesis. Thanks to my parents, my brother and my husband for their love and encouragement without which this thesis could never have reached completion. Lastly, I would like to thank Babaji and Satish for teaching and helping me with everything.

Chapter 1

Introduction

Research in knowledge representation (KR) for understanding natural language (NL) has gained prominence over the years. In order for a natural language understanding (NLU) system to attain the level of competence of humans, it must have the same kind of knowledge of the world that a human has, and this large store of "world-knowledge" must be represented and manipulated in an efficient manner.

Many KRs have been utilized for the task of understanding NL. Although there are advantages in each representation, there are disadvantages as well. Thus, none of the representations is perfect, and the search continues for a KR that encompasses the advantages of other approaches and yet can avoid the disadvantages. This thesis conducts one such search attempt by introducing a schema and constraint-based KR approach for understanding NL. It combines the advantages of logical and procedural representations while overcoming some of their disadvantages.

Logic is precise and seems to express facts in a way that corresponds to our rational understanding of some domain [Hayes, 1977] [McCarthy, 1977], and the use of first-order logic for KR in NLU is quite popular. However, NLU systems using logic for KR are inefficient. This is due to the fact that representation and processing are separated in the logic-based approach [Barr & Feigenbaum, 1981]. A logic representation may be appealing but the processing method that determines how the stored facts are manipulated can be inefficient. For example, the use of the resolution method of inference in QA3 [Green, 1969] causes combinatorial explosion in the number of ways facts can be combined to make inferences as the number of facts in the database increases.

A procedural approach to KR offers more efficiency because the embedding of knowledge into procedures allows control over the deduction process. However, it has disadvantages as well. In a logic-based system, addition of logical facts or assertions is straight-forward, but in a procedural-based system, addition of facts can be complicated since it may cause changes in the heuristic knowledge of different procedures due to the unavoidable interaction between various facts and the nature of heuristics. Thus, modularity of knowledge in the database is lost in this approach. Winograd's SHRDLU [Winograd, 1972] is a system that adopted this KR. Winograd pointed out in [Winograd, 1980] that the representation of the speaker/hearer internal structure in SHRDLU was *ad hoc*. As a result, the preciseness of logical representation is lost.

In this thesis, we describe a schema and constraint-based approach to NLU that attempts to incorporate the efficiency of a procedural approach, and yet can attain a reasonable level of modularity of knowledge and preciseness of representation that a logical approach offers. Efficiency is hard to achieve in most NLU systems with large databases because search is involved. Our approach takes an alternative view of language as providing semantic constraints on descriptions of the linguistic world instead of driving a search for such a description. This means that we are trying to view NLU as a constraint satisfaction problem that avoids search whenever possible.

A class of network consistency algorithms has been developed by Waltz, Montanari, Mackworth and Freuder respectively to solve constraint satisfaction problems efficiently. The complexity of these algorithms has been studied [Mackworth & Freuder, 1984], and it has been shown that one type of consistency, arc consistency, is achievable in quadratic time, and path consistency in cubic time in the worst case. These algorithms have been applied to machine perception tasks successfully [Waltz, 1972] [Mackworth, 1977b] [Havens & Mackworth, 1983] [Mulder, 1985]; we apply them to NLU in this thesis. Computer vision and NLU can both be considered as recognition tasks which need to resolve uncertainties or ambiguities in the input. In computer vision, segmentation of the gray scale of a digitized picture into edges and regions is required before interpretation is performed whereas in NLU, it is the segmentation of the input sentence into words and syntactic categories. In both cases, there is a need for feedback between segmentation and interpretation. Also, knowledge of either a visual or natural language domain can be represented by specialization hierarchies [Brachman, 1982] which are knowledge structures that categorize classes of objects in the world that is being represented. These similarities between vision and NLU tasks as well as the successful application of network consistency techniques on vision tasks make us believe that it is appropriate to use a constraint-based approach for NLU.

Logic is the best tool for achieving preciseness and modularity of knowledge; however, efficiency is hard to attain at the same time. We believe that a schema and constraint-based approach offers efficiency as well as provide a reasonable degree of modularity and preciseness of representation. Several representations fall into the category of schema knowledge representations. They include frames [Minsky, 1975], scripts and plans [Schank, 1975] [Schank & Abelson, 1977]. Also, different forms of memory schemata [Bobrow & Norman, 1975] are classified as schema knowledge representations. [Rumelhart & Ortony, 1976] also discusses the representation of knowledge in memory in terms of schemata. In each case, a schema represents a self-contained collection of related knowledge. An expansion of the knowledge base (KB) is equivalent to the addition of schemata in this approach, and since the knowledge is well-organized into schemata with the interactions between them well-defined, additions to the KB are quite straight-forward.

The combination of consistency techniques and schema knowledge representations as a representational formalism has been described as schema labelling [Havens, 1985]. The methodology has been used for interpreting hand-printed Chinese characters [Bult, 1986], and for recognizing VLSI circuit designs from their mask layouts [Alon & Havens, 1985]. This thesis adopts this approach to represent the syntax and semantics of English sentences.

The system implemented takes a simple sentence as input and produces a representation of the sentence's syntax and semantics as output which is in the form of a parse tree (PT) and a network consistency graph (NCG). The PT is the syntactic structure assigned to the input sentence after it has been analyzed according to the given phrase structure grammar. It represents the syntactic aspects of the input sentence. The semantics are captured in the NCG which is a semantic network of schemata linked by labelled arcs which specify the relationships between the semantic entities of the sentence. The PT and NCG are created in parallel as the system analyses the input sentence one word at a time from left to right. Each word may provide additional syntactic or semantic constraints that help the system to disambiguate the syntax and semantics of the sentence. Syntactic constraints are applied in the PT whereas semantic constraints are applied in the NCG. At this point, network consistency techniques are utilized to maintain consistency in the NCG after the semantic constraints are applied. Schema knowledge representations are used to represent all the knowledge needed by the system in the KB which is a collection of static schemata of syntactic and semantic information. If a sentence is ambiguous, then multiple representations are produced. An augmented version of Earley's context-free parsing algorithm is used to provide control for the system [Havens, 1983]. Chapter 3 of this thesis presents this schema and constraint-based approach to NLU through a detailed description of our implementation. Chapter 2 provides a background study of the field of NL from the early history of machine translation in the 1940s to the knowledge-based systems of the 1970s. Also, descriptions of some network consistency techniques and Earley's context-free parsing algorithm are given. Chapter 4 describes some sample sentence representations produced by the system. Chapter 5 discusses the merits and drawbacks of this approach, and the possibility of expansion of the system in the future.

Chapter 2

Background

This chapter gives a brief history of natural language research showing how the research trend has progressed from machine translation and pattern matching without the use of knowledge in the 1940s to knowledge-based systems in the 1970s. A collection of KR techniques for NLU is presented and the systems that adopt these techniques are mentioned and discussed. Finally, some background on network consistency techniques and Earley's context-free parsing algorithm are given.

2.1 Early history

Soon after computers came into existence in the 1940s, researchers were interested in applying them to the study of language. Initial attempts involved only surface-level processing of text such as the compiling of word indexes and concordances. During the 1950s, the main computer application for natural language was in machine translation [Weaver, 1949] where the computer tries to assume the role of a human translator in the translation of texts from one language to another. The basic method involves dictionary-lookup, word substitution, and rearrangement of the resulting string of words to fit the target language's word order. However, the focusing on syntactic information alone in such attempts produced poor results. The problems that arose include the choosing of appropriate word equivalences when a word has several translations depending upon the context, and the rearrangement of words in the target language to produce a truly equivalent sentence. It then became apparent that high-quality translation can only be achieved if the system can "understand" the input text before it reconstructs the string of words in the target language. In addition, such language understanding involves much "world-knowledge" which is applied implicitly when humans translate from one language to another [Bar-Hillel, 1960].

Thus, the focus of AI natural language research shifted to that of natural language understanding [Winograd, 1980]. However, the early NLU systems that were developed in the 1960s still did not deal with the issue of knowledge representation; these systems used *ad hoc* data structures to store facts about a restricted domain. In addition, the syntax of language was not dealt with in any sophisticated way, and semantic knowledge was only implicit in the patterns and heuristics used for parsing. A representative set of these early systems include SAD-SAM [Lindsay, 1963], BASEBALL [Green et al., 1963], STUDENT [Bobrow, 1968], ELIZA [Weizenbaum, 1966], and SIR [Raphael, 1968].

The next and current phase of NL research that began in the 1970s is closely connected with research on the representation of knowledge. The NLU programs that belong to this category of research are called knowledge-based systems in [Barr & Feigenbaum, 1981], since these systems use a fair amount of knowledge about the domain of discourse to help understand sentences and the knowledge is stored within the system using some knowledge representation method.

2.2 Knowledge-based systems

Instead of using *ad hoc* data structures for storing facts about the domain of discussion like the early NLU systems did, all knowledge-based systems utilize some formal KR method for storing, retrieving and manipulating knowledge. The principal KR methods that have been utilized in NLU systems include logical forms, procedures, semantic networks and the various forms of schemata such as frames, scripts and plans.

Logic was one of the first KR methods used in AI. One of the early NLU systems that used logic to represent knowledge is the general-purpose question-answering system QA3 [Green, 1969]. QA3 could solve simple problems in a few different domains such as chemistry, robot movement and automatic programming. It used the resolution method of inference to perform deductions and was quite successful in solving simple problems with a certain degree of generality. However, as the number of facts increases in a database, the system's performance decreases because the resolution method caused combinatorial explosion in the number of ways facts are combined to make inferences.

More recent attempts in using logic for representing knowledge in NLU systems include the works of NL researchers such as Dahl, Pereira, Warren, and McCord. Dahl has written a NL query system to be consulted in Spanish/French [Dahl, 1981] where the grammar and the facts of the database were written in PROLOG, a programming language based on first-order logic. This system was later adapted to Portuguese consultation by H. Coelho and F.

Pereira, and subsequently to English consultation by D. Warren and F. Pereira. With the development of the PROLOG programming language, logic was used in these cases as the underlying representational formalism as well as a programming language for the translation of NL input to a NLU system.

Logic programming, since the development of the PROLOG programming language, has become quite popular. In addition, several grammar formalisms have been developed and incorporated into PROLOG, including metamorphosis grammars [Colmerauer, 1978], extraposition grammars [Pereira, 1981], and a special case of metamorphosis grammars which Pereira and Warren have given the name definite clause grammars [Pereira & Warren, 1980].

Extraposition grammars were used in a NL question-answering system called Chat80 by Pereira and Warren which was also written in PROLOG and was designed to be efficient as well as easily adaptable to a variety of applications [Warren & Pereira, 1982]. NL analysis is performed in three separate phases in this system. The translation phase translates an English sentence into logic. Next, the planning phase augments the logical form with extra control information which makes it an efficient piece of PROLOG program. The last phase is execution where the optimised PROLOG code is executed to produce an answer.

Dahl and McCord have subsequently developed, both separately and jointly, NLU systems that solve NL problems such as coordination and quantifiers [Dahl & McCord, 1983], and use slots and modifiers for syntactic and semantic interpretation [McCord, 1982]. These systems were also written in PROLOG with the use of logic grammars.

The procedural representation of knowledge is another KR method that has been employed in NLU systems. The procedural approach stresses the importance of "knowing how" to use knowledge and achieves this by representing the knowledge of the world as procedures which when executed know how to carry out specific actions according to the heuristic knowledge that is embedded in the procedures. This type of domain-specific information allows more directed deduction processes and thus provides the efficiency that declarative representations lack. However, the procedural approach has its disadvantages as well. The knowledge that is captured in procedures cannot be stated explicitly as it can be in declarative representations thus making it less accessible, and consequently making the task of adding and changing procedures quite difficult as minor changes in one procedure may have far-reaching effects on other procedures. The merits and drawbacks of the two methods are discussed in [Winograd, 1975] in response to the declarative/procedural controversy of the 1970s.

Woods' question-answering system about airline flight schedules [Woods, 1968] is an example of an early procedural system. Input questions are translated into functions which when executed over the system's database produce the correct answer. The LUNAR program [Woods et al., 1972] follows from this research and takes a similar approach. It is a NL information-retrieval system that aids geologists with their task in evaluating data on moon rock and soil composition obtained from the Apollo-11 mission. The system processes English queries in three steps. The syntactic analysis step takes an English query and produces a derivation tree with the help of an augmented transition network parser [Woods, 1970]. The derivation tree is then transformed to an expression in a formal query language that captures the semantics of the original English query in the semantic interpretation step. Finally, the query language expression is executed over the database to produce a response.

The other representative system of the procedural approach is the SHRDLU program [Winograd, 1972]. The program maintains an interactive dialogue with the user regarding its simulation of a robot arm that manipulates a set of toy blocks on a table. Syntactic, semantic and reasoning knowledge are embodied in procedures which are pieces of executable code. The program was written in LISP and MICRO-PLANNER. The latter is a version of the PLANNER language [Hewitt, 1972] which represents procedural data in the form of "theorems". PLANNER's paradigm is theorem proving, and it satisfies a goal by looking for the appropriate "theorem" to prove it. However, it allows the efficiency of the process to be increased by providing the user the flexibility to specify his own heuristics in his procedures.

The semantic network formalism for KR has its origin in Quillian's development of a psychological model of human associative memory [Quillian, 1968] and his Teachable Language Comprehender (TLC) program [Quillian, 1969] which simulates this memory model. Many systems with a network-based representation have been written since then. However, all that some of these systems have in common is simply the superficial common notation of having nodes and arcs representing some sort of a taxonomic hierarchy. [Woods, 1975] stresses the importance of understanding what the notation means, and [Brachman, 1983] discusses the many different meanings that the IS-A link of a semantic network has adopted over the years and points out those meanings which are important in expressing knowledge and which gives a semantic network its expressive power.

An early attempt at using the semantic network formalism that follows directly from Quillian's TLC work is Carbonell's work on a computer-aided instruction program called SCHOLAR [Carbonell, 1970]. This tutoring program can answer questions posed by students about South American geography; the geographical information is stored in a semantic network. Around the same time, Fillmore's work on linguistic case structure [Fillmore, 1968] was influencing the development of semantic networks. The cases of Fillmore were utilized as the underlying relationships that label the arcs in the networks in [Simmons & Slocum, 1972], [Simmons, 1973] and [Rumelhart & Norman, 1973]. Although these works all used the case frame approach in choosing arc types, Rumelhart and Norman's system was more psychologically oriented and placed more emphasis on the simulation of long-term memory and human cognitive processes, whereas Simmons' system put more emphasis on the generation of answers to questions using the semantic network. [Shapiro, 1982] is a more recent attempt in the research in sentence generation from semantic networks.

A couple of speech understanding systems, [Woods et al., 1976] and [Walker, 1978], used semantic networks for representing knowledge. In connection with Walker's system, Hendrix has developed the idea of "network partitioning" [Hendrix, 1975] [Hendrix, 1979] in which the nodes and arcs are partitioned into "net spaces" which allow for the representation of logical connectives, the delimiting of the scopes of quantified variables, the encoding of alternative and hypothetical worlds as well as the focusing of attention at particular levels of detail. Grosz has adopted the idea of network partitioning in her work on the representation and use of focus in dialogue understanding [Grosz, 1977].

Woods's 1975 paper, "What's in a link", has raised some important questions regarding the logical adequacy of the semantic network representation and has asked us to consider for the first time the meaning of the semantic network notation. In response to these issues raised by Woods, many researchers have attempted to investigate the underlying logical meaning of the network formalism and to deal with the expressive inadequacy of the notation. This includes the following works: [Schubert, 1976], [Hayes, 1977], [Schubert et al., 1979], [Levesque & Mylopoulos, 1979] and [Shapiro, 1979]. In fact, [Shapiro, 1979] follows directly from Shapiro's earlier work, [Shapiro, 1971], where a distinction was already made between the conceptual level and the structural level of the network. An excellent historical review of semantic networks may be found in [Brachman, 1979].

Related to the research on semantic networks is the work on schemata for knowledge representation. The term schema was first used by Bartlett in relation to his work on memory [Bartlett, 1932]. Schemata involve organizing knowledge into aggregate structures. A variety of representations fall into this category. Minsky's frames [Minsky, 1975] and Schank's scripts and plans [Schank, 1975] [Schank & Abelson, 1977] are among the well-known ones. [Bobrow & Norman, 1975] and [Rumelhart & Ortony, 1976] also discuss schema representations in representing knowledge in memory.

Frames are data structures that are used to represent stereotyped situations; scripts are frame-like structures that are designed to represent sequences of events that describe some stereotyped human activities, and plans differ from scripts in the sense that they participate in the explanation of the sequences of actions that lead to a goal. An important characteristic of schema-based or frame-based systems is their capability to represent both declarative and procedural knowledge [Winograd, 1975]. A frame is organized into slots and besides representing a collection of static facts, a frame can have procedures attached to its slots to drive the reasoning process of the system. Other interpretations of the idea of frames are provided in [Hayes, 1981].

The GUS system [Bobrow et al., 1977] is an experiment with frame-based NLU. It was designed as a prototype of an automated airline reservation assistant. It attempted to illustrate the expectation-driven processing of frames and facilitate in the understanding of various aspects of NL such as indirect answers and anaphoric references. The systems, SAM (Script Applier Mechanism) and PAM (Plan Applier Mechanism), were developed to demonstrate the use of scripts and plans in understanding simple stories [Schank et al., 1975] [Schank & Abelson, 1977]. By understanding a story, we mean the ability to paraphrase the story and make inferences from it. SAM does this by trying to fit the story into one or more scripts and it has been used to understand newspaper stories [Cullingford, 1978]. The PAM system [Wilensky, 1978] understands stories by determining the goals of the characters in the story and then interpreting their actions by matching them to plans that lead to those goals.

The idea of frames as structures that unify a collection of related knowledge is interesting but vague. The development of KRL (Knowledge Representation Language), a frame-based representation language, was an attempt to make precise the intuitive ideas about frames and to explore frame-based processing [Bobrow & Winograd, 1977]. The language KL-ONE [Brachman, 1978] [Brachman, 1979] [Brachman & Schmolze, 1985] has also been designed for supporting structured knowledge representations. The principle structure of KL-ONE is the "concept" whose primary component is a "role". Roles are like generalized attribute descriptions representing the relationships between entities denoted by the concept and other entities.

KL-ONE has inspired other new research efforts done on representations including KRYPTON and KL-TWO. The KRYPTON system [Brachman et al., 1983a] [Brachman et al., 1983b] overcomes some of the trouble with frames with regard to its limitations in representing either assertions or descriptions. Instead of defining the system in terms of structures for representing knowledge, KRYPTON provides a functional view of a knowledge base emaphasizing what it can be asked and told about the domain. The system has two major components, namely the terminological component and the assertional component. The former represents objects in terms of definitional knowledge using frame-like structures whereas the latter represents propositions similar to the manner of first-order predicate calculus language. Thus, KRYPTON handles both terminological and assertional knowledge by combining frame-like structural representations with a first-order theorem prover. This type of hybrid inference system that provides more than one language for expression of domain knowledge is appealing because an intelligent system usually has more than one kind of representational need [Brachman & Levesque, 1982] [Brachman et al., 1985].

KL-TWO [Vilain, 1985] is similar to KRYPTON in that it is also a hybrid representation system. However, instead of basing the representation system on a first-order theorem prover, KL-TWO is based on a device called RUP (Reasoning Utility Package) [McAllester, 1980] [McAllester, 1982]. RUP provides only a subset of the inferential power of a first-order theorem prover, but it is computationally more efficient. KL-TWO is basically composed of two components, called PENNI and NIKL. PENNI is a modified version of RUP and NIKL is a terminological reasoner which is a direct descendant of KL-ONE.

Another recent attempt in hybrid representation is the integration of frames with production rules in the KEE system [Kehler & Clemenson, 1984]. The use of a frame-based representation in this system to assist in the task of reasoning is well-described in [Fikes & Kehler, 1985].

The lattest attempt in providing a representation that incorporates schemata and yet can avoid their vagueness is schema labelling [Havens, 1985], which defines schemata precisely for recognition tasks. The theory shows how schema knowledge representations can be combined with network consistency techniques to yield a descriptively and procedurally adequate formalism.

This thesis adopts the schema labelling approach and is therefore partially schema-based. The knowledge base for our system is organized into schemata. In addition, the resulting network representation that captures the semantics of an input sentence is a network of schemata, each being a particular instance of one of the model schemata given in the knowledge base. Our schemata basically provide us with a structured representation of a collection of information, but do not have embedded procedures that participate in the reasoning process of the system as frames and scripts have.

2.3 Network consistency techniques

Some network consistency techniques are reviewed in this section to provide the background for understanding our constraint-based approach to NLU that adopts these techniques.

Network consistency techniques were developed in the attempt to solve constraint satisfaction problems (CSP). [Mackworth & Freuder, 1984] defines a CSP as follows: Given a set of n variables, each with a particular domain and a set of constraining relations which involve a subset of the variables, find all possible n-tuples such that each n-tuple is an instantiation of the n variables in their particular domains satisfying the relations. The variable domains can be continuous or discrete, and the relations can be unary, binary or generally n-ary. For discrete domains that consist of a finite set of values, backtracking may be used to solve a CSP. However, backtracking is inefficient and exhibits problems such as thrashing [Bobrow & Raphael, 1974]. As a result, network consistency algorithms were developed.

Waltz's filtering algorithm [Waltz, 1972] was the first of a set of network consistency algorithms. His algorithm was designed to analyse line drawings of toy blocks. In particular, the algorithm was a procedure for filtering out impossible labels for the lines in the drawing. Lines in the drawing meet at points called junctions, and there are a set of possible labellings for each type of junction. The junctions in the drawing are the set of variables for this CSP. Each variable has a domain of allowable labelled junctions. The problem is to determine which junction interpretations provide a globally consistent interpretation. Consistency is forced by the constraint that each line in the drawing must be assigned one and only one label along its entire length.

Waltz's filtering algorithm works as follows. The filtering procedure goes through the junctions in any order. For each junction, the procedure looks at its neighbours and sees whether the constraint is satisfied. The constraint is satisfied if the line shared by two junctions has the same label at both ends. If this constraint is not satisfied, that labelling for the junction is eliminated. When such a deletion occurs, all neighbouring junctions whose interpretations are constrained by the deleted junction interpretation are revisited. This and other network consistency algorithms eliminate all local inconsistencies that cannot participate in any global solutions, but they do not necessarily solve the CSP. In the best possible case, the CSP can be solved to yield one interpretation for each junction if there are sufficient constraints to propagate such a solution. However, if the algorithm does not leave us with a unique labelling for each junction when it terminates, a backtracking search can be used to enumerate the remaining possible labellings. In any case, the algorithm is at least an efficient preprocessor in that it reduces the size of the domains of the variables in only a single pass through the junctions. Waltz's filtering algorithm only deals with binary constraints.

Mackworth [Mackworth, 1977a] presented three types of network consistency algorithms, namely node, arc and path consistency algorithms for eliminating local inconsistencies that involve 1, 2 or 3 variables respectively. An obvious generalization of his algorithm would deal with arbitrary n-ary constraints. Waltz's filtering algorithm is in fact a special case of AC-2, the second arc consistency algorithm presented by Mackworth, whereas Montanari's algorithm [Montanari, 1974] is a version of a path consistency algorithm.

Freuder's network consistency algorithm [Freuder, 1978] called k-consistency removes all inconsistencies involving all subsets of size k out of the n variables. His algorithm is a generalization of Mackworth's algorithms. Node, arc and path consistency corresponds to Freuder's 1-, 2- and 3-consistency for k=1, 2 or 3 variables respectively. Freuder's algorithm differs from the others in the sense that it may determine all solutions to a CSP if we have k=n, thus eliminating the need to have further tree search to determine the solutions.

[Mackworth & Freuder, 1984] is a study of the complexity of these network consistency algorithms. It is shown that node, arc and path consistency can all be achieved in polynomial time. In particular, arc consistency is achievable in time linear in the number of binary constraints using AC-3, the third arc consistency algorithm, and the worst case complexity for arc and path consistency is $O(n^2)$ and $O(n^3)$ respectively. In comparison to depth-first backtracking whose worst case complexity is exponential time, these network consistency algorithms show a significant improvement.

A network consistency algorithm known as HAC (Hierarchical Arc Consistency) has recently been developed [Mackworth et al., 1985] to exploit structured domains of the variables in CSPs. This algorithm is useful when the variable domains can be organized hierarchically such that all the possible labels in each domain need not be represented explicitly but can be represented implicitly by one or more abstract labels. Hierarchical arc consistency can also be achieved in linear time like AC-3. Under the specific condition where the domains are appropriately structured, HAC performs better than AC-3, but in the worst case where there is no solution, HAC may be twice as slow as AC-3. The HAC algorithm has been used in the Mapsee3 system [Mulder, 1985] for interpreting hand-drawn sketch maps.

Our approach to NLU is partially constraint-based. We try to view the task of NLU as a CSP so that we can exploit these network consistency techniques to achieve efficiency in analysing NL. Backtracking search is so commonly used in NLU systems to discover the validity of a sentence and as we have mentioned before, network consistency techniques are superior to backtracking. When viewing NLU or any other recognition tasks as a CSP, we need to identify the variables of the problem, their domains and the constraints that are applied on them. We have chosen the semantic entities of a sentence as the variables for the CSP; the domain for each of these entities is the set of meanings that are attached to it, this is similar to all the facts of the database that are applicable to this entity; the constraints on these entities are the semantic constraints that express the allowable relationships or interactions between these entities. A hierarchical version of arc consistency, AC-2 [Mackworth, 1977a] in particular, is used by our system to arrive at an interpretation for each semantic entity. Although arc consistency does not guarantee a solution, the algorithm can reduce all local inconsistencies such that the resulting network incorporates all the possible interpretations of the sentence. Backtracking search may then be used to assemble actual global interpretations. However, the use of network consistency techniques for pre-processing has restricted backtracking search to a minimal.

2.4 Earley's context-free parsing algorithm

Many parsing algorithms have been developed in the past. This includes the left-parsable (LL) parsing algorithms, the operator precedence parsing algorithms, the predictive parsing algorithms and the right-parsable (LR) parsing algorithms. Although these algorithms are very efficient in that they run in linear space and time, they can only handle a small subset of context-free grammars. However, these efficient algorithms are adequate in handling all the syntactic features of programming languages. Natural languages include more difficult phenomena than programming languages. For example, the grammar may be ambiguous and usually all parses instead of just one are of interest. Thus, at least a general context-free parsing algorithm is needed for processing natural languages.

Backtracking algorithms may be used but they require exponential time. Tabular methods, on the other hand, are asymptotically much faster than backtracking algorithms. Earley's context-free parsing algorithm [Earley, 1970] and the Cocke-Younger-Kasami algorithm [Aho & Ullman, 1972] are two of the most well-known ones that run in polynomial time. They each take space n^2 and time n^3 where n is the length of the input string. Earley's algorithm has the advantage that it only requires time n^2 whenever the grammar is unambiguous and can even operate in time n for most grammars for programming languages whereas Cocke-Younger-Kasami's algorithm requires the grammar to be in Chomsky normal form. Also, it has been shown that Earley's algorithm is applicable to schema-based systems [Havens, 1983]. As a result, we adopted Earley's algorithm for our system.

We will now describe the algorithm informally. A formal description may be found in [Earley, 1970]. The use of Earley's algorithm in the context of our NLU system is explained in detail in chapter 3.

Earley's algorithm is a recognizer. This means that it takes as input a string and either accepts or rejects it depending on whether or not the string is a sentence of the specified grammar. The grammar is a context-free grammar (CFG) and is formally defined as a 4-tuple

$$G = (N, T, P, S)$$

where N is a finite set of non-terminal symbols, T is a finite set of terminal symbols, P is a finite set of production rules, and S is a distinguished symbol of N called the start symbol. Each production in P is of the form

$$A \rightarrow \alpha$$

where A is a symbol in N and α is a string $\sigma_1 ... \sigma_n$ where σ_i is in $(N \cup T)$. Let $w = x_1 x_2 ... x_n$ be the input string with every x_i in T, $1 \leq i \leq n$.

Earley's algorithm scans the input string from left to right and as each symbol x_i is scanned, a set S_i of states is constructed. A state has the form

$$[A \rightarrow \alpha \cdot \beta, j]$$

which represents the condition of the recognition process in recognizing the non-terminal A starting at x_{j+1} in w using the production $A \rightarrow \alpha\beta$ where α and β are in $(N \cup T)^*$. The dot between α and β is a metasymbol not in N or T, and it delimits the portion, α, of the production's right hand side which has been recognized so far from the portion, β, that still needs to be recognized.

Initially, since none of the symbols in the input string has been recognised, the algorithm begins by creating the state set S_0 containing just the state $[\phi \rightarrow \cdot S \dashv, 0]$. ϕ is a new non-terminal that is not in N, $\dashv$ is a new terminal symbol not in T ,and S is the root of the grammar. $\dashv$ represents the null character at the end of an input string to act as a terminator. As the algorithm processes each new symbol x_{i+1}, a new state set S_{i+1} is generated. If the entire input string is processed and the state set S_{n+1} contains the single state $[\phi \rightarrow S \dashv \cdot, 0]$, then the input string is an accepted sentence of the given grammar. On the other hand, if at any point in the processing S_{i+1} remains empty after S_i has been processed, then w is rejected as a valid sentence of the grammar G.

A state set S_i is operated on in the following manner. The states in the set are processed in order and depending on the form of the state, one of three operations is performed on it. These three operations are known as predictor, scanner, and completer. They may add more states to S_i and may also put states in the next state set S_{i+1}. The algorithm moves on to process the states of S_{i+1} after all the states in S_i have been processed.

The predictor operation is applicable to a state when there is a nonterminal to the right of the dot. This operation causes a new state to be added to S_i for each alternative production of that non-terminal. Thus, in the beginning, the state $[\phi \rightarrow \cdot S \dashv, 0]$ indicates that the predictor operation can be applied and states of the form $[S \rightarrow \cdot\alpha, 0]$ are placed in S_i for each production $S \rightarrow \alpha$ in P.

The scanner operation is applicable to a state when there is a terminal to the right of the dot. This operation compares that terminal symbol with x_{i+1}, and if they match, it adds the state to S_{i+1} with the dot moved to the right of the terminal symbol to indicate that it has been scanned. Thus, if the state is $[A \rightarrow \alpha \cdot c\beta, k]$ and c is a terminal symbol that matches the input symbol x_{i+1}, then the state $[A \rightarrow \alpha c \cdot \beta, k]$ is placed in the next state set.

The completer operation is applicable to a state such as $[A \rightarrow \alpha c \cdot, k]$ when the dot is at the end of the production. This operation retrieves all states from the previous state set S_k that have in their productions the non-terminal symbol A to the right of the dot. The dot is then moved past A in these states and these updated states are added to the current state set S_i. Thus, if the state under processing is $[A \rightarrow \alpha\cdot, k]$, then the completer operation will retrieve all states from the state set S_k that have the form $[B \rightarrow \alpha \cdot A\beta, j]$, $1 \leq j \leq n$, $j < i$, propagates them as $[B \rightarrow \alpha A \cdot \beta, j]$, and places them into S_i.

Conceptually, the predictor provides all possible extensions of a partially completed parse. The scanner checks potential extensions against the input. The completer updates the confirmed extensions to enable the parsing process to continue.

Our system adapts the idea of Earley's parsing algorithm to processing sentences. In fact, the algorithm's three main operations of predicting, scanning and completing form the basis of control in our system. However, the productions that are used in our system have embedded syntactic and semantic constraints to enable the analysis of English sentences whose grammar is not context-free. The algorithm has been altered to accomodate this and also to produce all the possible parse trees or derivations of the input sentence instead of just acting as a recognizer.

Chapter 3

A Schema & Constraint-Based Approach to NLU

The schema and constraint-based approach to NLU is explained in this chapter through the description of a NLU system which follows this approach. See [Havens, 1985] for an introduction to combined schema and constraint-based recognition.

3.1 Overview of the system

An overview of the system is shown in Figure 3.1. Each component of the system is described briefly in this section. The main purpose of the system is to explore the use of schema knowledge representations in combination with network consistency techniques in the task of NLU. Thus, the system is not designed as a NL query system that provides answers to questions on a database, but rather as a representational system that produces a syntactic and a semantic representation for a given input sentence.

The system accepts simple declarative or interrogative English sentences as input. It utilizes the information provided by the knowledge base (KB) to analyse the sentence, producing as

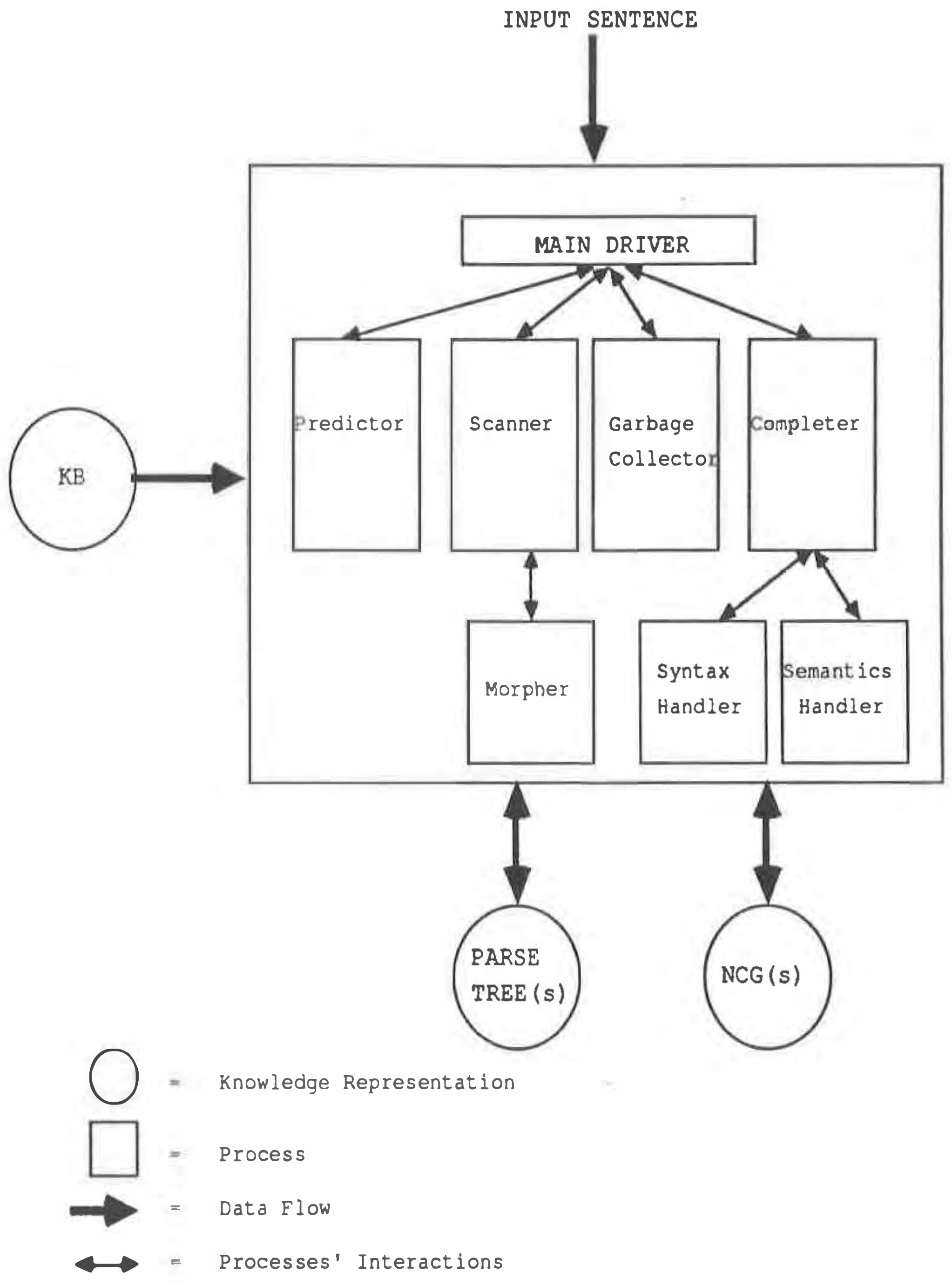

Figure 3.1: Overview of the system

output a parse tree (PT) and a network consistency graph (NCG) which represent the syntax and semantics of the input sentence respectively. If there are multiple interpretations for a sentence, then multiple PTs and NCG s are produced in parallel with shared structures whenever possible. Focus of the system is on representing the meaning of input sentences.

A PT represents the syntactic structure of an input sentence and is built by the system as it analyses the sentence according to the grammar provided in the KB. This structure is a tree with each node representing a syntactic category, and each leaf node also represents a word in the input sentence thus identifying the word's syntactic category and its relation with respect to the entire structure. The formation of this structure is performed in parallel with the construction of the NCG which represents the semantic structure of the input sentence. As each input word is analysed and appropriately identified with a PT node, one or more semantic nodes may be created in the NCG which correspond to the semantics of the input word. Links are established at this point between the PT node that represents the current input word and the corresponding semantic nodes in the NCG. This allows the ongoing parsing process to further guide the construction process of the NCG.

The NCG is a semantic network that is built incrementally in parallel with the PT. The semantics of each input word is represented by one or more nodes in the NCG. The arcs that link these semantic nodes represent the relationships that exist between the semantic components of the sentence. These semantic components are entities from the database and the relationships between them correspond closely with the cases of Fillmore [Fillmore, 1968]. As each new node is added to the NCG, additional semantic constraints are introduced. These constraints may alter the existing interpretation that each node contributes towards the global interpretation of the sentence. In order to maintain a consistent global interpretation with all constraints satisfied, network consistency techniques are applied at this point.

The KB provides the knowledge about the task domain. It also contains both syntactic and semantic knowledge that guide the system in its construction of the PT and the NCG. The information within the KB is represented using schemata.

The main driver of the system controls the interpretation of each sentence by manipulating a combination of processes. These processes are the predictor, the scanner, the completer and a garbage collector. They direct the analysis performed by the system and are derived from Earley's context-free parsing algorithm.

The predictor computes the phrase structure grammar rules that may be involved in the derivation of the PT and predicts the syntactic nodes that may participate in one or more of the final PT structures. The scanner scans the input sentence one word at a time to provide the necessary information that leads to the acceptance or rejection of the predictions made by the predictor. The completer then narrows the predictions of the multiple interpretations to the appropriate ones that conform to the input. The garbage collector removes the impossible interpretations as soon as they are discovered. This involves the removal of syntactic and semantic nodes from the PT and the NCG.

There are other subordinate processes defined in the system. The scanner has a subprocess called the morpher which performs suffix analysis on each word scanned. The completer has two subprocesses, namely the syntax and the semantics handlers. The syntax handler applies syntactic constraints on the nodes in the partially built PT whereas the semantic handler applies semantic constraints on the nodes in the partially built NCG. It also participates in the construction of the NCG. Network consistency techniques are utilized by the semantic handler to resolve the semantic constraints that arise as each word is scanned. If either the syntactic or the semantic constraints are not resolved, then the completer is notified to terminate the current interpretation.

In summary, the main driver, with its processes and subprocesses, performs the analysis of input sentences while the KB provides the knowledge that enables the interpretations to take place. The PT and NCG are the final products of this interpretation. Multiple PTs and NCGs will be created if there is more than one interpretation for an input sentence.

3.2 Knowledge base (KB)

The knowledge that the system needs in order to analyse English sentences resides in the KB. The task domain is composers and their music. The KB is a static collection of ***model schemata***. Each of these schemata is a prototype which represents either syntactic or semantic knowledge. During the interpretation process, ***derived schemata*** are generated from these prototypes forming the syntactic and semantic nodes of the PT and the NCG.

There are two types of model schemata in the KB. The ***syntactic schemata*** represent syntactic knowledge which specifies how sentences are structured. This is its knowledge of English grammar. The other type of model schemata is the ***semantic schemata*** which represent the semantics of words. These two types of model schemata are described in the following sections.

3.2.1 Syntactic schemata

Each syntactic schema represents a syntactic category and is referred to by the abbreviated name of the syntactic category which it represents. Table 3.1 shows the abbreviated names of the syntactic categories that are handled by the system. Figure 3.2 gives the BNF syntax of a syntactic schema.

s	–sentence	pp(s)	–prepositional phrase(s)
np	–noun phrase	qword	–question words
subnp	–noun group	auxv	–auxiliary verb
vp	–verb phrase	v	–verb
comps	–subject-complements	det	–determiner
mods	–general modifiers	adj	–adjective
nmods	–noun modifiers	npr	–proper noun
adjs	–adjectives	n	–common noun
nposs	–possessive nouns	prep	–preposition

Table 3.1: Syntactic categories' abbreviations

Each syntactic schema contains a set of composition rules which define the various ways the syntactic category can be composed. If the syntactic category denotes a terminal symbol in the grammar, then the syntactic schema will have the rule (***dot *term**) to indicate that the syntactic category is composed of an input word. Otherwise, a composition rule is composed of the symbol ***dot**, grammar constituents, plus one or more predicates. The symbol ***dot** at the beginning of each rule serves a special purpose in Earley's parsing algorithm that is described in section 2.4. It is used to delimit the portion of the rule that has been recognized and constraints that have been applied and resolved successfully from the portion that still needs to be found

```
<syntactic schema> ::= (<composition rule1>^+)|(<composition rule2>)
<composition rule1> ::= (*dot <grammar constituent>^+ | <predicate>^*)
<composition rule2> ::= (*dot *term)
<grammar constituent> ::= s|np|ngroup|vp|comps|mods|nmods|adjs|nposs|pp|pps|qword|
                          auxv|v|det|adj|npr|n|prep
<predicate> ::= (<syn predname> <syn args>^*)|(<sem predname> <sem args>^*)
<syn predname> ::= rwordis|hasfeature|gparhasno|gparhas|isposs|isnotposs|nvagree|vvagree|
                   dnagree
<sem predname> ::= build|setptr|sp|checknum
<syn args> ::= <wlist> | <syn feature> |(<grammar constituent>)| <grammar constituent>^2
               (<test>)^{0,1}
<sem args> ::= <grammar constituent><word>^{0,1} | <grammar constituent>^2
               <arc reln>^{0,1} |(schild <grammar constituent>)<grammar constituent>
               <arc reln>
<wlist> ::= a list of English words
<syn feature> ::= (trans)|(intrans)|(cop)
<test> ::= (iftrans)|(ifnosubj)
<word> ::= an English word
<arc reln> ::= <condn list> | <case> | <function>
<condn list> ::= ((<word><case choice>)^+)
<case> ::= agent|obj|mod|temp|locn|event
<function> ::= depends|rdepends
<case choice> ::= <case> |(<case>^+)
```

Figure 3.2: BNF grammar for a syntactic schema

from the input sentence and from the constraints that still need to be applied. The predicates are either syntactic or semantic constraints that should be applied at the appropriate time in the recognition process. They provide context-sensitive information to the system in deciding the appropriate parse. They also provide the necessary instructions on when and how to build and link nodes and maintain consistency in the NCG. The semantics of these predicates are explained in section 3.4. The complete set of syntactic schemata can be found in the appendix.

Each composition rule of a syntactic category denotes a possible composition or parse or interpretation for that syntactic category. Multiple interpretations exist when multiple rules of a syntactic schema are simultaneously active. For a particular interpretation to be valid, all constraints in that rule must be satisfied and the input must conform with the rule's compositional elements.

The syntactic schemata are models of grammatical rules. The schemata in the PT that capture the grammar of a sentence are derived schemata of these model syntactic schemata in the KB. They are created during the analysis of a sentence and are dependent both on the models and on the input. The *dot symbol is always at the beginning of a rule of a model syntactic schemata, but each derived schema's rule will probably have its *dot in different positions in the rule reflecting the status of that schema with respect to the condition of the parsing process. The creation and manipulation of these derived schemata is explained in more detail in the following sections.

As an example, one rule in the set of composition rules for the sentence syntactic schema named 's' is:

(*dot np vp (nvagree np vp) (sp vp np agent) (sp np vp agent) (checknum vp))

This rule says that an input sentence is a valid sentence if it is composed of a noun phrase (np) followed by a verb phrase (vp). This is the rule for the composition of a declarative sentence and there are other rules that specify alternate ways that a sentence can be formed. They can be found in the s syntactic schema given in the appendix.

There are four predicates in this sample rule:

```
(nvagree np vp)
(sp vp np agent)
(sp np vp agent)
(checknum vp)
```

(nvagree np vp) is a syntactic constraint. It says that to satisfy the constraint there must exist a number agreement between the main noun in np and the main verb in vp. For instance, if the main noun is in singular form, and the main verb is in third person singular form, as in 'Bach lives', then the constraint is satisfied. Interpretation of a sentence based on this rule is allowed to continue only if the constraint is satisfied. All syntactic constraints serve a similar purpose, which is to check certain nodes in the partially constructed PT for a particular property or for some syntactic agreement between nodes that must exist before a sentence can be considered valid and that the interpretation process can continue.

The remaining constraints in the example are semantic constraints, and are applied on the semantic nodes in the NCG(s). The first two constraints are called specialization constraints. They create arcs between the semantic node that represents np and the semantic node that represents **vp** in the NCG. The relationship between the two is marked 'agent'. This means that **np** fills the agent case of the schema relations of **vp**. Network consistency must exist when such links are made between semantic nodes. The technique of network consistency was introduced in section 2.3, and section 3.4.2 explains how it is performed on a NCG in detail. If the NCG is still consistent after a link is made between two of its nodes, then the semantic constraint is satisfied.

The (**checknum vp**) semantic constraint does not create new links in the NCG. Instead, it tries to apply the number constraints provided by determiners on the semantic nodes of nouns. The specialization constraint and the number constraint are the two basic types of semantic constraints. The specialization constraint is usually less straight forward in its specification of the relationship that should exist between nodes such as the 'agent' relationship given in the above example. In most cases, a list of possible relationships and the conditions under which they are applicable are specified, and it is the semantic handler's job to decide which relationship is appropriate for a given situation.

3.2.2 Semantic schemata

For each English word, there's a schema in the KB that represents it. The schema contains all the syntactic and semantic knowledge of the word that is needed in order that the system may function. Figure 3.3 gives the syntax for such a semantic schema. A brief explanation of each of the properties that a semantic schema may have is given in Table 3.2.

Words of different syntactic categories have different types of syntactic and semantic knowl-

<semantic schema> ::= <word><syn cat><inflns><syn feature>$^{\{0,1\}}$<labelset>
<case list>$^{\{0,1\}}$<schema relations>$^{\{0,1\}}$<sem rule>$^{\{0,1\}}$
<word> ::= a word
<syn cat> ::= n|npr|pn|adj|qword|det|prep|v|auxv
<inflns> ::= s|es|s-d|s-ed|es-ed|er-est|r-st|irr|*| <infln entry>
<syn feature> ::= (trans)|(intrans)|(cop)
<labelset> ::= (<label>$^{+}$)
<case list> ::= (<case>$^{+}$)
<schema relations> ::= (<sreln>$^{+}$)
<sem rule> ::= (equal-num <num>)|(morethan <num>)
<infln entry> ::= (<rootwd><possessive>$^{\{0,1\}}$<number>$^{\{0,1\}}$<tense>*<person>$^{\{0,1\}}$)
<label> ::= <word> | <fabricated label>
<case> ::= agent|obj|mod|temp|locn|event|label
<sreln> ::= (<fabricated label><label>$^{+}$)
<num> ::= 0|1|2|3|4|5|6|7|8|9
<rootwd> ::= the non-inflectional form of an English word
<possessive> ::= (poss)
<number> ::= (number pl)
<tense> ::= (infin)|(tns present)|(tns past)|(pastpart)|(prespart)
<person> ::= (pncode <code>)
<fabricated label> ::= <word><num>$^{+}$
<code> ::= 1sg|3sg|13sg|x13sg

Figure 3.3: BNF grammar for a semantic schema

word: an English word or an identifier representing an English description.

syntactic category: the syntactic class that the word belongs to.

inflectional knowledge: the possible variety of terminations of a word to express the relations of number, person and tense.

syntactic features: exists only for verbs, classifying them as transitive, intransitive or copula verbs.

label set: A collection of labels to capture the meaning of a word.

case list: exists only for verbs and prepositions, specifying the cases that may be filled.

schema relations: exists only for verbs and prepositions, specifying the objects that may fill the given cases, thus showing the relationships between the objects.

semantic rule: exists only for determiners, a semantic predicate that specifies how to quantify nouns.

Table 3.2: Properties of a semantic schema

edge in their schemata. Thus, not every semantic schema has all the properties listed in Table 3.2. Each of these properties and the type of words to which it pertains is explained further below.

Basically, syntactic knowledge includes the syntactic category that a word belongs to and its syntactic features. All words have syntactic knowledge in their semantic schemata. Words that are of the syntactic categories of nouns and verbs also have knowledge of their inflections included in the schemata. Inflection is a type of syntactic feature that through variance in word termination expresses the relations of number, person and tense. For example, the plural form of the noun 'composer' can be formed with the suffix 's' appended to the word. Thus, 's' is the inflectional knowledge within the schema for the word 'composer'.

Certain verbs are considered as irregular and have no inflectional knowledge that can guide the morpher in deriving the tense and person of an inflected form of the verb. Thus, the knowledge of person and tense are provided explicitly for each present, past and past participle form of the verb in the KB. Verbs also have the syntactic features that categorize them as transitive, intransitive or copula verbs.

Syntactic knowledge of words is needed because the satisfaction of syntactic constraints depends on them. For example, a syntactic constraint such as **(hasfeature (trans))** is satisfied if the schema for the verb has **(trans)** as a syntactic feature meaning that the verb is transitive.

Similarly, the satisfaction of semantic constraints depends on the semantic knowledge within the semantic schemata. Words of the syntactic category of determiner are basically quantifiers in our system. Their semantics is represented by predicates, such as **(equal-num 1)** for the determiner 'one', which is basically a semantic constraint which when executed captures the semantics of the word. Thus, we are assuming that the semantic significance of determiners is in providing number constraints on nouns.

Words of other syntactic categories have their semantics represented by label sets. A label set is a collection of labels which are meant to capture the meaning of a word. For example, the word 'composer' which is a noun has the label set:

(Bach Handel Haydn Mozart Beethoven Chopin Berlioz Tchaikovsky Verdi)

We are assuming that a noun represents a class of objects and we call each of these objects a label. A label may be another noun or a pseudo-noun which also has a label set whose labels also have label sets and so on. As a result, a hierarchy of knowledge organized into classes and subclasses is formed for each noun. This is known as the specialization hierarchy

[Brachman, 1982]. Pseudo-nouns are descriptions for a collection of nouns. They allow groups of nouns or labels to be categorized under one name. For example, 'voc-music' is a pseudo-noun that represents the collection of objects that fall under the category of vocal music. The specialization hierarchy for the word 'music' in Figure 3.4 shows how pseudo-nouns can group nouns together and be used as labels. The pseudo-nouns are in italics in the figure and the descriptions that they represent are given in parentheses.

As with nouns, the semantic knowledge of adjectives and question words are represented by label sets within the semantic schemata. For an adjective, its label set is the class of all objects that may be modified by the adjective. For example, the adjective 'famous' has in its schema the label set (music composer) in our KB because all of the musical compositions and composers are considered famous and can be modified by this adjective. Similarly with a question word, its label set represents a class of objects. For example, the question word 'who' represents the class of all people and since all the people in our database are composers, its label set is simply (composer).

Prepositions and verbs also have label sets for representing semantics of words. However, there is a substantial difference between their label sets and those of nouns, adjectives and question words. Verbs denote linguistic events, and each event involves the interaction of objects. For example, the verb 'compose' represents all the composing events and each such event may involve a composer, a composition and the time and place that the compositon was written. Thus, verbs' label sets are not simply classes of objects as in nouns but are classes of events. We represent each event by what we call a fabricated label so that the event with

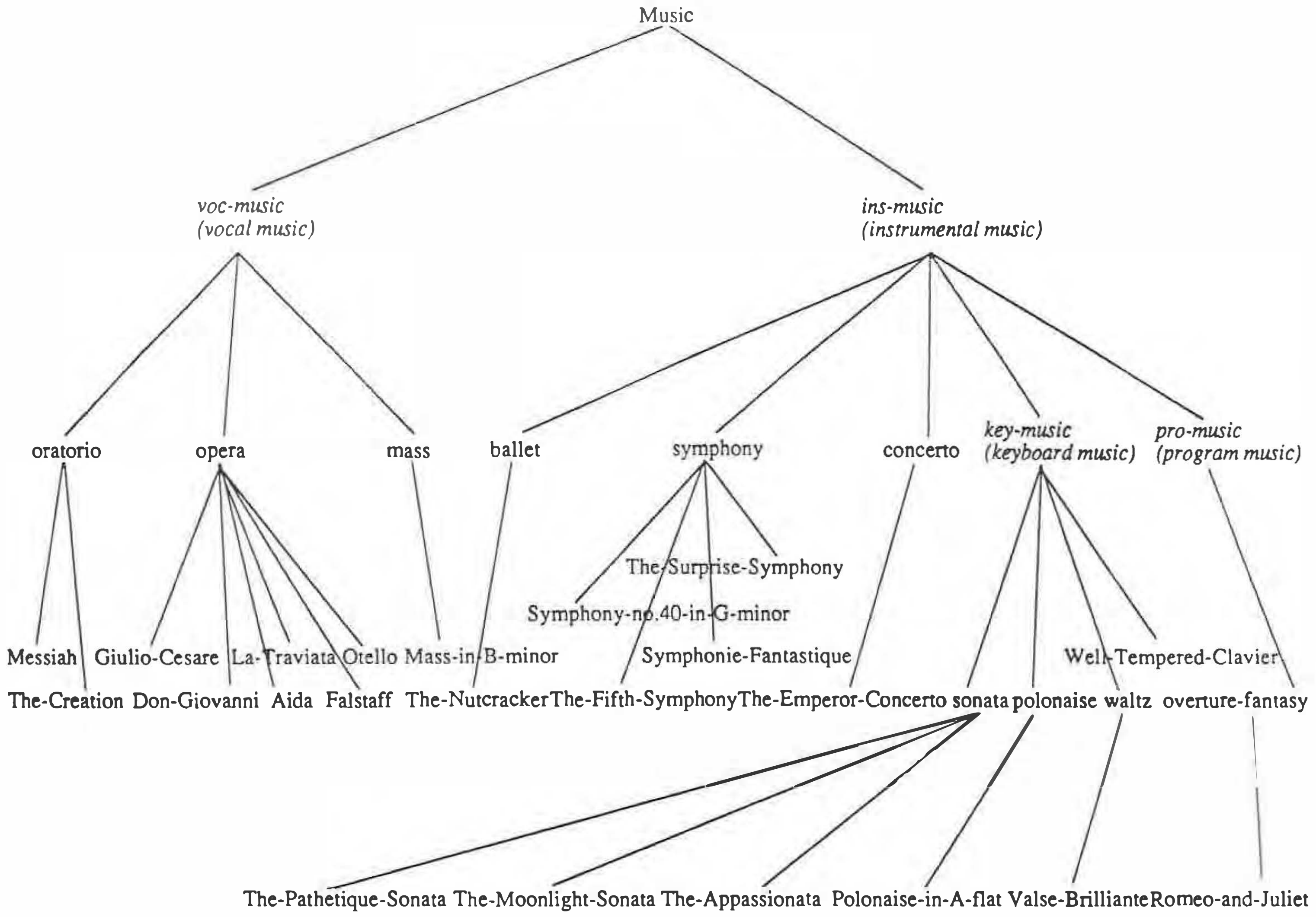

Figure 3.4: Specialization Hierarchy for the noun 'music'

all the inter-related objects can be referred to by a single name. For example, if there are altogether three composing events for the verb 'compose', then its label set will be **(compose1 compose2 compose3)** where each element in the list is a fabricated label denoting an event. In addition, we also need to include what we call a case list and schema relations within the semantic schemata of verbs. The case list within a semantic schema expresses the cases of the verb that may be filled. This is an adaptation of Fillmore's cases [Fillmore, 1968]. For each fabricated label, there is a corresponding schema relation that lists the objects that fill the cases of the verb for a particular incidence of that verb. The set of schema relations is a set of tuples which represents all the events that are denoted by the verb in the KB. Thus, for the verb 'compose', the case list is (label agent obj mod) and each line in Table 3.3 denotes a schema relation for each of the three events in the label set.

label	agent	obj	mod
compose1	Bach	Mass-in-B-minor	in1
compose2	Bach	Well-Tempered-Clavier	
compose3	Handel	Messiah	in2

Table 3.3: Sample schema relation for the verb 'compose'

The first schema relation in the table says that we have an event named 'compose1' which represents the event of composing where 'Bach' is the agent of the act of composing and the composition 'Mass-in-B-minor' is the object of the act. Modifiers on the act are summarized by the fabricated label 'in1'. The other two schema relations can be interpreted in a similar fashion. To summarize, in the case that there are only three composing events, then the semantic schema

word:	compose
syntactic category:	v
inflectional knowledge:	s-d
syntactic features:	(trans)
label set:	(compose1 compose2 compose3)
case list:	(label agent obj mod)
schema relations:	(compose1 Bach Mass-in-B-minor in1)
	(compose2 Bach Well-Tempered-Clavier)
	(compose3 Handel Messiah in2)

Figure 3.5: Sample semantic schema for the verb 'compose'

word:	in
syntactic category:	prep
inflectional knowledge:	*
label set:	(in1 in2 in3)
case list:	(label event temp locn)
schema relations:	(in1 compose1 1733 Leipzig)
	(in2 compose3 1742 Dublin)
	(in3 born1 1685 Eisenach)

Figure 3.6: Sample semantic schema for the preposition 'in'

for the verb 'compose' in the KB contains the information given in Figure 3.5.

In order to understand the fabricated label 'in1' that appeared in a schema relation in Figure 3.5, we now turn to the semantic schemata of prepositions. Somewhat like verbs, prepositions do not simply represent classes of objects but represent the relationships between objects. Thus, within the semantic schemata of prepositions, we again have label sets of fabricated labels, case lists and schema relations as in verb schemata. A simplified version of the semantic schema of the preposition 'in' is given in Figure 3.6 to illustrate this.

The 'in1' in the schema relation of the fabricated label 'compose1' in the previous example is itself a fabricated label which expresses the relationship of the event of composing with the time and location of the event in one name. For example, the schema relation of 'in1' informs us that the event represented by 'compose1' has happened in the year 1733 in the city of Leipzig.

3.3 System control

Earley's context-free parsing algorithm [Earley, 1970] and Havens' schema-based recognition method [Havens, 1983] have been adapted to provide the necessary control for the interpretation process in our system. Recognition is performed in a bottom-up fashion coupled with top-down control. This integrated bottom-up and top-down recognition method allows the construction of the PT and the NCG in parallel. In addition, all possible interpretations can be explored in parallel while each schema retains control of the recognition process. The result is the creation of multiple PTs and NCGs with shared networks for multiple interpretations. We shall describe control in terms of three processes after Earley's three parsing functions in the following sections. A brief description of the actions involved in the interpretation process is given below.

The main driver of the system is responsible for controlling the direction of interpretation of an input sentence. Direction is either bottom-up or top-down depending on the status of the derived syntactic schemata in the state sets of the system. The status of the derived syntactic schemata are in turn dependent on the input sentence and the KB. A derived syntactic schema is an instantiation of a model syntactic schema in the KB. Its creation signifies the possibility of its corresponding model syntactic schema in capturing the syntax of a part of or the entire input sentence. Derived syntactic schemata reside in state sets which eventually turn into the PT representation.

Initially, the main driver of the system creates a state [1] called the ***derived root schema*** that contains the rule (*dot s *term) [2] and places this schema in the state set called st0. This schema is responsible for the chain reaction spawning of syntactic schemata derived in the future. The list (*dot s *term) represents the status of the derived root schema. *dot is a symbol that delimits the recognized portion of the rule from the unrecognized portion. Initially, *dot is at the far left of the rule stating that s which is the root of the grammar has not been recognized yet. The user may choose another nonterminal symbol to be the root of the grammar. For example, if one wants to see the representation for a noun phrase instead of an entire sentence, then one may choose np to be the root of the grammar and (*dot np *term) will be the derived root schema instead. *term is a terminal symbol whose presence causes the scanner process to scan a word from the input sentence and in this case a null character should be scanned to signify the end of the input sentence. It is necessary to recognize *term after s as stated in the above rule to ensure that the end of the input string is reached after s has been recognized.

The main driver contains a loop that processes state sets one at a time until a new version of the derived root schema with the rule (s *term *dot) appears in a state set indicating that the input sentence has been interpreted successfully or until an empty state set is encountered indicating failure in the interpretation process. A state set is processed if all the derived syntactic schemata that it contains have been operated on in order; a derived syntactic schema

[1] Earley's terminology. [Aho & Ullman, 1972] calls a state an item, and calls a state set a parse list. We call the states in a state set derived syntactic schemata since each of them contains a collection of information based on the model syntactic schemata from the KB instead of being just a simple rule of the form $[A \rightarrow \alpha \cdot \beta]$.

[2] Our LISP representation of the right hand side of the rule $[\phi \rightarrow \cdot S \dashv]$ described in section 2.4.

may be operated on by either the predictor, the scanner or the completer processes depending on its status. These three processes may add more derived syntactic schemata to the current state set under processing or to the next state set to be processed. The actions of each process is either top-down or bottom-up processing. Derived syntactic schemata that do not contribute to the final interpretation of the input sentence are deleted from the state set that contains them by the garbage collector as they are found to be inappropriate during the interpretation process. Thus, all the derived syntactic schemata in the series of state sets at the end of the interpretation process form a parse tree (PT) that provides a syntactic structure for the analysed input sentence. The three processes are described in the following sections.

3.3.1 Predictor

When the main driver inspects the status of a derived syntactic schema within the current state set being processed and notices that there is a nonterminal symbol to the right of *dot in the rule within the schema, the predictor process is invoked. The predictor creates a new derived syntactic schema for each composition rule within the model syntactic schema indicated by the nonterminal symbol, and places them in the current state set for future processing. In the beginning, st0 is the current state set containing the derived root schema with the rule (*dot s *term). Since s is a nonterminal symbol, the predictor is invoked. It retrieves the model syntactic schema for the syntactic category s from the KB, and for each composition rule within the s model syntactic schema, a derived syntactic schema is created.

One of the derived syntactic schemata of the s syntactic category created at this point is shown in Figure 3.7.

instance:	s0-0
class:	s
state:	st0
rule:	(*dot np vp (nvagree np vp) (sp vp np agent) (sp np vp agent) (checknum vp))
parent:	root0-0
potchild:	nil
child:	nil
word:	nil
sem:	nil

Figure 3.7: Sample derived syntactic schema

The **instance** slot stores the name of a derived syntactic schema which is 's0-0' in this example. Names are constructed following a special rule that allows it to reflect information of which syntactic class the derived syntactic schema belongs to, its position in the list of derived syntactic schemata of the same syntactic class, and which state set it resides in. The purpose of the naming convention is basically for ease of identification during the creation of the PT.

This schema is derived from the s syntactic category and is contained in the state set st0 as specified by the **class** slot and the **state** slot respectively. The **rule** slot contains the composition rule that represents the status of this schema in the recognition process.

The **parent** slot records the name of the derived syntactic schema which causes the prediction of this schema. This is necessary so that the completer process knows which derived syntactic schema to complete to when this schema's rule has reached the completion stage. Completion of the rule reflects that the nonterminal symbol s has been captured in the input sentence by the composition rule of the s syntactic category represented in this schema.

The **potchild** slot records all the derived syntactic schemata that will be predicted from this schema. For example, this and other derived syntactic schemata spawn from 'root0-0' are all potential children of 'root0-0' representing all the possible parses with s as the root. Later, when this schema is processed, the nonterminal symbol **np** after ***dot** in the rule will cause all the possible compositions of **np** to be predicted as derived syntactic schemata, each of these will be placed in the **potchild slot** of this 's0-0' schema. The purpose of having this slot is to facilitate garbage collection. This schema can be garbage collected when all its potential children in the **potchild slot** have been deleted from the slot. The entry of a derived syntactic schema in the **potchild slot** is removed if that schema is garbage collected due to its rule not being able to capture the input thus destroying its potential in being a child of the current schema in the final PT. Alternatively, the entry of the derived syntactic schema in the **potchild slot** can be removed if that schema has completed its rule and has made a copy of the current schema, thus showing that it no longer needs the original version of the current schema. The copy of the current schema is that schema's new parent; the rule in this copy is updated to reflect the successful completion of that schema in recognizing the nonterminal symbol after ***dot**. It is necessary to make a copy of the current schema and update the rule in the copy instead of just updating the rule in the original schema because the original one may have more than one potential children, some of which may complete to it later expecting the rule not to have changed in the mean time.

The **child slot** is used to record those potential children that have successfully completed their rules. This information enables the system to print out the PT representation at the end of the interpretation process thus allowing Earley's algorithm to be a parser instead of simply a recognizer in the formal sense. The **word slot** is used to record the word from the input sentence that is scanned during the processing of the current schema such that it may be printed with the schema that it is associated with in the PT output. Only those derived syntactic schemata that have a terminal symbol in their rules can have the **word slot** filled. The **sem slot** provides a link to the NCG by storing the names of those semantic nodes in the NCG that represent the semantics of the current derived syntactic schema.

Lastly, there is one important point that needs to be mentioned with regard to the actions of the predictor. Before the creation of any new derived syntactic schemata, the predictor checks the current state set to see whether those schemata have already been predicted previously and are waiting to be processed. If so, the predictor will not create duplicate derived syntactic schemata, but will instead retrieve those derived syntactic schemata from the current state set and update its **parent slot** to reflect the fact that there exists another schema which also predicts them. This means that subparses may be shared by multiple PTs thus yielding space efficiency.

3.3.2 Scanner

The scanner process is invoked when there is a terminal symbol, ***term**, to the right of ***dot** in the rule of the derived syntactic schema under processing. The scanner then fetches the next word from the input sentence, and passes it to the morpher subprocess so that the root form of the input word can be determined. It then checks whether the resulting root word belongs to the syntactic category specified under the **class slot** of the current derived syntactic schema. If it does, it partially confirms the validity of the series of derived syntactic schemata that lead to the prediction of the current derived syntactic schema. It is only a partial confirmation because there may still exists constraints that need to be satisfied before these derived syntactic schemata just mentioned can participate in the final PT representation.

A derived syntactic schema that has been scanned successfully will be updated and then be placed on the next state set. The updating operation involves altering the name of the schema to comply with the naming convention, and most importantly to update the rule in the schema to reflect its new status. Before scanning, the rule in the derived syntactic schema is (...*dot *term), and after updating, it will be (...*term *dot) which means that all symbols to the left of *dot has been recognized.

A special case that the scanner process has to deal with is when the scanned word is a null character signifying the end of the input sentence. If the derived syntactic schema is the root schema with the rule (s *dot *term), then the scanning is successful, and normal updating as mentioned above is performed. Otherwise, an incorrect prediction has occured which expects more input and yet input has been exhausted. As a result, the garbage collector is invoked to dispose of the current schema as well as any other related derived syntactic schemata that are involved in the incorrect prediction. The garbage collector is also invoked when an unsuccessful scan has occurred; that is when the scanned input word does not belong to the syntactic class specified by the current derived syntactic schema or when it cannot be analysed successfully by the morpher.

3.3.3 Completer

The completer process is invoked whenever the derived syntactic schema under processing has *dot at the end of its rule. It usually means that a successful scan has occurred previously when this derived syntactic schema was last processed, and now its parents should be informed that the prediction is correct. So, the first operation of the completer is to retrieve all the parents of the current schema as listed in its **parent slot**. If required, consistency checks are performed on each parent. If the results are satisfactory, then the parent schema is updated and placed at the end of the queue in the current state set to await further processing. This involves either more prediction to see if the current predicted parse can be carried to its completion or completion has actually been reached. In the latter case, a trace back the prediction path is needed to confirm the success of the prediction.

When a parent is retrieved, a copy is made so that the status of the original copy will not be affected. The original copy is needed by other incompleted potential children and thus should not have its status changed. A new name is given to the copy and its rule is updated to reflect the completion of a derived syntactic schema predicted from it.

For example, if the rule in this parent copy is **(np *dot vp (nvagree np vp) (sp vp np agent) (sp np vp agent) (checknum vp))**, then the updated rule will have ***dot** to the right of **vp**. This indicates that a derived syntactic schema of the syntactic category **vp** has been correctly predicted and is now completing to its parents. Consistency checks are required for this parent since four constraint predicates, **(nvagree np vp)(sp vp np agent)(sp np vp agent) (checknum vp)**, appear after the ***dot** in its rule. The syntactic handler will be invoked to handle the syntactic constraints, and the semantic handler the semantic constraints. ***dot** will be advanced to the right of each constraint after it has been satisfied. When all constraints are satisfied, the completed derived syntactic schema will be recorded under the

child slot of its parent. The **potchild** slot of this parent which is the copy is set to nil since no predictions has been made from it yet. The derived syntactic schemata representing the recognized portion of its rule are listed under its **child slot**. The original copy of this parent will have the completed derived syntactic schema removed from its **potchild slot** since it is now a completed child of the copy.

The changes made to Earley's parser are incorported in the operations of the completer as mentioned above. Whereas Earley's completer just retrieves the parents and updates their rules, our completer also has subprocesses to apply syntactic and semantic constraints in order to be able to incoporate context-sensitive information in the grammar rules in the appropriate places.

Although a completed derived syntactic schema may be shared by multiple parents, its corresponding semantic nodes in the NCG cannot be shared as well since each parent may be involved in a different interpretation. Thus, copies of semantic nodes must be made to ensure that all the semantic interpretations are distinct and do not interact. However, space efficiency is desirable; so some semantic nodes are still shared by different NCGs as long as the interpretations can be kept separated. This handling of multiple NCGs is described in section 3.4.2 on the semantics handler.

The description of the three system control processes has shown that the parsing procedure is nondeterministic. The predictor process predicts all the possible parses simultaneously in a top-down fashion. These multiple parses are incorporated in the state sets. The derived syntactic schemata which reside in the state sets are created as a result of the prediction process. They are retained unless they are found to be incorrectly predicted via the bottom-up action of the scanner and the completer. The derived syntactic schemata which cannot participate in the final PT representation will be deleted from the appropriate state sets.

The scanner detects those incorrectly predicted derived syntactic schemata by checking the prediction against the words in the input sentence. On the other hand, the completer detects the incorrectly predicted derived syntactic schemata by calling the syntax and semantics handlers to apply the necessary constraints. Those derived schemata, both syntactic and semantic ones, that do not satisfy the constraints are then deleted.

3.4 Constraint satisfaction

This section explains how syntactic and semantic constraints are resolved. These constraints are embedded in the composition rules of a model syntactic schema. They are resolved through the actions of the syntax handler and the semantics handler respectively.

3.4.1 Syntax handler

The syntax handler is invoked by the completer whenever the derived syntactic schema under processing has syntactic constraints that need to be satisfied. Syntactic constraints are applied to the syntactic nodes of the PT. These syntactic nodes are simply the derived syntactic schemata in the state sets from where the final one or more PTs are formed. The main function of these syntactic constraints is to provide context-sensitive information to the system so that an incorrect parse may be found as early as possible. The information that these constraints provide is essential for the parsing of English sentences since they do not conform to a simple context-free grammar.

The syntactic constraints may be divided into two categories. Syntactic constraints of the first category are basically conditions that a particular derived syntactic schema must satisfy or properties that the derived syntactic schema must possess. This type of syntactic constraint assists the system in deciding which composition rule to follow at the appropriate stage during the interpretation process. The second category of syntactic constraints usually involves two particular derived syntactic schemata. The constraint specifies the number agreement that must exist between these two schemata. This type of syntactic constraint assists the system in deciding which composition rule to discard.

Both constraint types are represented as a list such as (**hasfeature** (**trans**)) where the head of the list is the name of the constraint, and the tail of the list gives the arguments to the constraint. The syntax handler's job is to call on the appropriate LISP function to perform the named constraint. It must also pass the required arguments to the LISP function. This usually involves a search for the appropriate derived syntactic schemata to which the constraint is applied. For example, the constraint (**nvagree np vp**) can only be applied after the syntax handler has located the main noun and the main verb of the sentence. The main noun, for instance, can be found in the **word slot** of a derived syntactic schema of class **n** which is listed in the **child slot** of another derived syntactic schema of class **np**. Sometimes a constraint may have an argument that specifies the condition under which it is appropriate to apply the constraint. This type of embedded constraints must be handled by the syntax handler as well. A list of the syntactic constraints, and a brief explanation of each one is given in Table 3.4.

- Category I syntactic constraints:

  rwordis: The current derived syntactic schema must be associated with an input word whose root form is specified in the argument list of this constraint.

  hasfeature: The argument of this constraint specifies the particular syntactic feature that the word in the word slot of the current derived syntactic schema must have.

  gparhasno: The argument of this constraint specifies a syntactic class that the rule of the grandparent of the current derived syntactic schema must not have.

  gparhas: The opposite of the above constraint.

  isposs: The word in the word slot of the current derived syntactic schema must be in possessive form.

  isnotposs: The opposite of the above constraint.

- Category II syntactic constraints:

  nvagree: Number agreement must exist between the main noun and the main verb.

  vvagree: Number agreement must exist between the auxiliary verb and the main verb.

  dnagree: Number agreement must exist between the determiner and the main noun.

Table 3.4: Table of syntactic constraints

3.4.2 Semantics handler

The semantics handler is invoked whenever the completer process encounters semantic predicates in the rule of the derived syntactic schema under processing. Semantic predicates may be constraints that need to be applied on the semantic nodes in the NCG, or they may be commands that specify how the NCG is built and linked to its corresponding derived syntactic schemata. The semantic handler also manages the creation of mutiple NCGs to capture multiple semantic interpretations of an input sentence.

Each semantic predicate is in the form of a list similar to a syntactic constraint with the name of the predicate given in the head of the list, and the arguments of the predicate given in the tail. If the semantic predicate is a constraint, then the semantic handler has to perform a search operation similar to that of the syntax handler in finding the appropriate semantic nodes to which to apply the constraint. There are four semantic predicates, given in Table 3.5. **Build** and **setptr** are commands for the semantics handler whereas **sp** and **checknum** are semantic constraints.

build:	Specify what type of semantic node to build and which derived syntactic schema it should correspond with.
setptr:	Specify additional links between syntactic and semantic nodes, and when to create copies of semantic nodes to allow multiple NCGs to exist separately.
sp:	A semantic constraint that causes the execution of arc consistency between semantic nodes. As a result, links between semantic nodes are established and made consistent.
checknum:	A semantic constraint on the number of labels in the label set of a semantic node as applied by a determiner.

Table 3.5: Table of semantic predicates

The **checknum constraint** is the simplest of the four. It simple checks the cardinality of the label set of a semantic node to see if it conforms with what the determiner indicates it to be. The operations of the semantics handler with regard to each of the other predicates are explained below.

The build predicate

The **build predicate** is applicable after an input word has been scanned. It builds one or more semantic nodes to represent the additional semantic entities that the new scanned word has introduced. A semantic node is a derived semantic schema which is an instantiation of a model semantic schema in the KB. The form of a semantic node for the scanned word, 'composer', is given in Figure 3.8.

snode:	sem6
rword:	composer
lset:	(composer)
comps:	nil
rcomps:	nil
filledc:	nil
rfilledc:	nil
syn:	(subnp4-3)

Figure 3.8: Sample derived semantic schema

The name of the semantic node is given in the **snode slot**. Names are numbered in the sequence in which they are created, and do not have any special significance as names of syntactic nodes do. The **rword slot** holds the root form of the scanned input word that has triggered the building of this node. It provides access to the model semantic schema associated with this derived semantic schema so that the case list and schema relations of the model one can be retrieved when needed. The **lset slot** initially contains the label set of the associated model semantic schema. The label set is refined as the interpretation process proceeds and only consistent labels that satisfy all constraints applied to them are retained. The label set represents the interpretation of the semantic node. The **comps slot** holds a list of semantic nodes which are the components of this node. A semantic node is a component of another if it applies a constraint on the other semantic node. Thus, a change in the label set of a semantic node will affect all the semantic nodes that have it as a component. The **rcomp slot** holds a list of semantic nodes which have this semantic node as a component. These two slots allow the semantic handler to have access to all the semantic nodes that are linked to this node. The **filledc** and **rfilledc slots** provide case information for the nodes listed in the **comps** and **rcomps** slots respectively. Figure 3.9 illustrates this through the example of a semantic node which denotes a verb.

snode:	sem5
rword:	write
lset:	(compose10 compose11 compose12)
comps:	(sem4 sem0)
rcomps:	nil
filledc:	((sem4 (obj)) (sem0 (agent)))
rfilledc:	nil
syn:	(vp0-4)

Figure 3.9: Sample derived semantic schema for a verb

In Figure 3.9, the component, 'sem4', fills the objective case of the verb write, and 'sem0' fills the agent case. These two components have refined the labels in the **lset slot** to only three composing events from the original list of all composing events when the node was first built. Lastly, the **syn slot** specifies the derived syntactic schema or schemata with which this semantic node is associated. The argument to the build predicate may specify the syntactic class of the derived syntactic schema with which this semantic node should associate. If unspecified, it defaults to the derived syntactic schema currently being processed by the completer which contains the build predicate in its rule.

The setptr predicate

The **setptr predicate** is applied when the list of semantic nodes of a completed derived syntactic schema should be incorporated into its parent's list. There are three special cases in which the **setptr predicate** applies:

1. A child has only one parent and the parent does not have other potential children.
2. A child has more than one parent to which to complete.
3. A child is completing to a parent which has other potential children.

The first case indicates that there is only one interpretation or parse active at the time and the other two cases indicate the existence of multiple parses or subparses. When multiple interpretations are involved, the current NCG would need to be split to capture each interpretation. This involves the copying of semantic nodes and creating a new NCG.

In the first case of only one interpretation existing, the **setptr predicate** can simply be executed by updating the parent's **sem slot** to include the semantic nodes in the **sem slot** of its child. Also, the **syn slot** of all the semantic nodes must be updated to indicate a new link to another derived syntactic schema. These additional links are necessary so that the system control which is provided via the syntactic parsing process can be carried over to control the formation of the semantic representations, the NCGs.

This simple case of having only one active predicted parse or subparse seldom occurs. Usually, one or both of the remaining cases occur during a completion process. In the second case, each parent of the child represents a different parse or subparse. Even though the child schema can be shared in the syntactic description, its corresponding semantic nodes cannot be shared since each parse represents a different semantic interpretation. Thus, appropriate actions must be taken to ensure that each parent has its own corresponding NCG. Similarly, in the third case, each potential child represents a different parse or subparse. As one of these potential children completes to its parent, a copy of this parent is made, as explained in section 3.3.3. Since each copy is involved in a different interpretation, its set of semantic nodes must be kept separated from that of other copies.

Copies of semantic nodes must be made in the two cases just mentioned to keep the semantic interpretations separate. However, each semantic node is linked to other semantic nodes in a NCG. If a copy of a semantic node is needed, all the linked semantic nodes that cannot be shared between NCGs must also be copied. The result is a splitting of a single NCG description into multiple NCG descriptions. Figure 3.10 illustrates how this is done. For the sake of readability, we use a hypothetical example to demonstrate the actions involve in splitting a NCG description. Here, semantic nodes have alphabetic names and only the **comps** and **rcomps** slots are shown. These slots contain the information on the links in a NCG and are the only slots that are affected in the splitting; thus we include them in the diagram to illustrate how they get updated.

The example assumes that node **A** is a semantic node that needs to be copied. Thus, node **A1** is created as a copy. However, the fact that node **A** is connected to other nodes in the

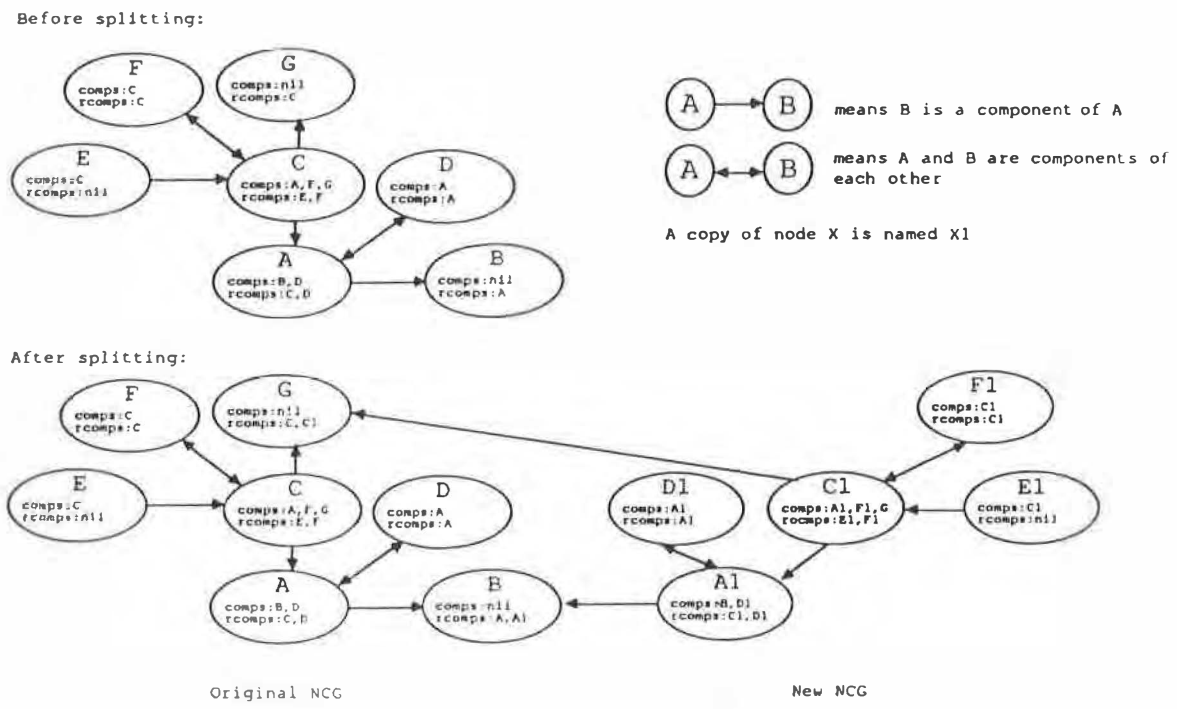

Figure 3.10: Splitting a hypothetical NCG description

network means that other nodes that cannot be shared by two interpretations must be copied and linked appropriately as well. Basically, if a node is copied, then all the nodes listed in its **rcomps slot** need to be copied as well, but the nodes listed in its **comps slot** need not be copied and can be shared. For example, node **A**'s **comps slot** has nodes **B** and **D**, and its **rcomp slot** has nodes **C** and **D**. Thus, copies are made for **C** and **D** where none is made for **B**. Copies are necessary for **C** and **D** because they have node **A** as a component which apply constraints on them, and if **A**'s bf lset slot is changed, **C** and **D**'s **lset slots** must be updated accordingly. Therefore, if node **A** has a copy **A1** which is supposed to represent a separate interpretation, then **C** and **D** should not still be linked to **A1** since they should not let their **lset slots** be affected by **A1**'s interpretation as well. As a result, copies of **C** and **D** are made. To summarize, the action of copying one semantic node usually leads to a propagation of the copying action until two interpretations can be represented by two separate NCGs that do not interact although they may share certain semantic nodes. An efficiency of representation is achieved with this sharing of node while keeping the interpretations separate.

The sp constraint

The sp constraint is applied whenever a link has to be established between two or more semantic nodes in the NCG. The arguments of the constraint give guidelines to the semantic handler in finding the semantic nodes that should be linked, and the relationship between them. Linking two nodes means one node is made a component of the other. The semantic handler then records the information concerning each link in the appropriate **comps**, **rcomps**, **filledc** and **rfilledc slots** of the two semantic nodes involved. In addition, it runs arc consistency to ensure that any inconsistencies introduced by the additional links are removed. Inconsistencies that cannot be resolved result in the constraint not being satisfied.

Consistency checking is basically a filtering mechanism for maintaining consistent label sets, which may also refine the label sets gradually to reflect a global interpretation if enough constraints exist. The network consistency algorithm that is adopted by our system is an implicit version of hierarchical arc consistency (HAC) [Mackworth et al., 1985]. In HAC, constraints are represented by relation matrices of leaf labels. These are explicitly established and preprocessed at the outset. In our system, the constraints are represented implicitly in the schema relations.

Network consistency in our system works as follows. For a link inserted between each pair of semantic nodes, one node is made the component of the other, say for example that node **B** is made a component of node **A**. Each label in the label set of **A** is checked to see whether the addition of **B** as a component has made a label inconsistent. If so, that label is deleted from **A**'s label set, and every neighbour of **A** in the network that has **A** as a component must also be checked for consistency. Thus, consistency checks propagate throughout the network to ensure that all labels in the label sets of all nodes are consistent with the inconsistent ones removed in the process.

A label of node **A** is considered consistent if it or at least one of its descendants in its specialization hierarchy of labels is compatible with at least one label in the label set of node **B**. Compatibility basically means equality. Two labels are compatible if they are equal in name, or if one of them is compatible with a member of the schema relation of the other label under the specified case. For example, the label 'Verdi' is compatible with the label 'compose21' whose schema relation is (**compose21 Verdi Otello**) if the specified case is 'agent' since 'Verdi' is a member of its schema relation and it resides in the agent case slot. If a fabricated label resides in that slot instead of 'Verdi', then the semantic handler will have to fetch the fabricated label's schema relation to check for compatability and a recursive process is involved until an atomic label can be reached to allow an atomic name comparison.

If a label is incompatible, its descendants at the next lower lever in its specialization hierarchy will be fetched to replace it in the label set. If at least one of its descendants is consistent with node B's label set, consistency is achieved for that original label. The checking for consistency in the hierarchy of a label and its descendants is essentially HAC.

Constraints are predominantly given by the schema relations of verbs and prepositions. However, noun modifiers also apply constraints on nouns. Adjectives and possessive nouns are the two types of modifiers that the system can handle. They refine the label set of the noun that they are modifying to only those labels to which the noun modifiers can be applied.

3.5 Morpher

The morpher is a subprocess to the scanner. It performs suffix analysis on an input word. Words that are not root words themselves but are formed in a regular way from their roots are analysed by the morpher to have the root form determined. These words need not be entered into the KB explicitly. However, words that are formed in an irregular way must be placed in the KB.

The morpher is given a table of possible suffixes and the syntactic features associated with each. It can determine the root form of the input word, the part of speech, and other syntactic features of the input word so that a semantic schema can be created for it dynamically. If a word cannot be analysed, the user will be asked to either respell it, give a definition for it, or abort the interpretation process. The definition must be given in the format of a semantic schema.

3.6 Garbage collector

Over the course of the interpretation process, many derived schemata, both syntactic and semantic ones, are created. However, only some will actually be part of the final PT and NCG representations. Therefore, for reasons of space efficiency, a garbage collector is provided in our system to reclaim the space used by derived schemata once they are found to be incorrectly predicted and do not play a part in the global interpretation.

The garbage collector is invoked whenever one of the three main processes of the system discover that a derived schema is inappropriate. The garbage collection process works differently depending on which process is calling it. The following is a description of the actions of the garbage collector in each situation.

After scanning, a derived syntactic schema may be found to have incorrectly predicted the input word. Thus, this derived syntactic schema is garbage collected. In addition, all the derived syntactic schemata that are related to this prediction, and are not involved in an alternate prediction at the same time are garbage collected. This involves tracing the prediction path to all the ancestors of this current derived syntactic schema and garbage collecting those which cannot participate in another parse. For each ancestor that can be garbage collected, its completed children may also be garbage collected if they are not shared by other derived syntactic schemata. As a result, the garbage collector traverses up and down the potential PT looking for derived syntactic schemata that can be disposed. At the same time, derived syntactic schemata that are related to the garbage collected ones must have the appropriate slots in their schemata updated to delete their connections to the garbage collected schemata.

A similar process is performed when a derived syntactic schema cannot complete successfully to its parent because some constraints are not satisfied. In this case, the garbage collector must also collect the related semantic nodes as well. Again, all connected semantic nodes that do not participate in another NCG description can also be garbage collected. Those that cannot be garbage collected must have their **comps**, **rcomps**, **filledc** and **rfilledc** slots updated to reflect the discontinued links.

The garbage collector can also be invoked when the predictor process is called to make further predictions when input has already been exhausted. In this case, the derived syntactic schema under processing cannot possibly be part of any final interpretations and it is garbage collected accordingly.

Chapter 4

Discussion of examples

The NLU system is written in FRANZ LISP and runs on a VAX 11/780 under the UNIX operating system. The grammar and semantic knowledge base is given in the appendix. The system can handle simple declarative and interrogative sentences. Approximately twenty complete sentences and ten sentence fragments were tested successfully. Four of these and their corresponding PT and NCG representations are illustrated in this chapter. A brief dicussion is given for each example.

The examples show how constraint propagation works and how the label sets of semantic nodes are refined incrementally to give a global consistent interpretation. The first two examples illustrate how noun modifiers are represented and manipulated in the system. Both of these examples are sentence fragments. The third example shows how determiners and prepositional phrases are handled. The last example demonstrates how multiple representations are generated for a sentence which has multiple interpretations. The last two examples are complete sentences. A diagram showing the PT and NCG representations in a network form is given for each example to facilitate understanding of the system's behaviour.

The contents of the syntactic and semantic nodes in the PT and the NCG are not shown in full in the diagrams to increase readability. Only information essential to the understanding of the diagrams is included. Leaf nodes of the PT indicate each input word that is scanned. The **rword** and **lset slots** of semantic nodes are listed beside the nodes in the diagrams. Also, the schema relations of the abstract labels in the label set of semantic nodes are listed even though they are not explicitly represented in each semantic node. They only reside in the model semantic schemata in the KB, but are listed in the diagrams to aid the reader in interpreting the fabricated labels.

Figure 4.1 gives the representation of the sentence fragment 'famous keyboard music'. The PT is shown on the left and the NCG on the right in the diagram. This example shows how adjectives affect the label set of a noun that they modify. In this noun phrase, both adjectives, 'famous' and 'keyboard', provide constraints on the noun, 'music'. As the parsing process scans the words from left to right, the semantic nodes, 'sem3', 'sem4' and 'sem5', are created along with their corresponding syntactic nodes. The label set of the noun 'music' is originally (**voc-music ins-music**). When 'subnp3-3' has established both 'mods2-2' and 'n3-3' as its children, the **sp constraint** in its composition rule is applied. It specifies the addition of links between the semantic node representing the noun and the semantic nodes representing the adjectives. Consistency checks are performed when the links are established. This results in the label set of 'sem5' to be refined to (**key-music**) to represent music which is famous and of type keyboard. At a first glance, neither 'voc-music' nor 'ins-music' in the original label set of 'sem5' matches the label 'key-music' in the label set of 'sem3'. However, the labels (**oratorio opera mass**)

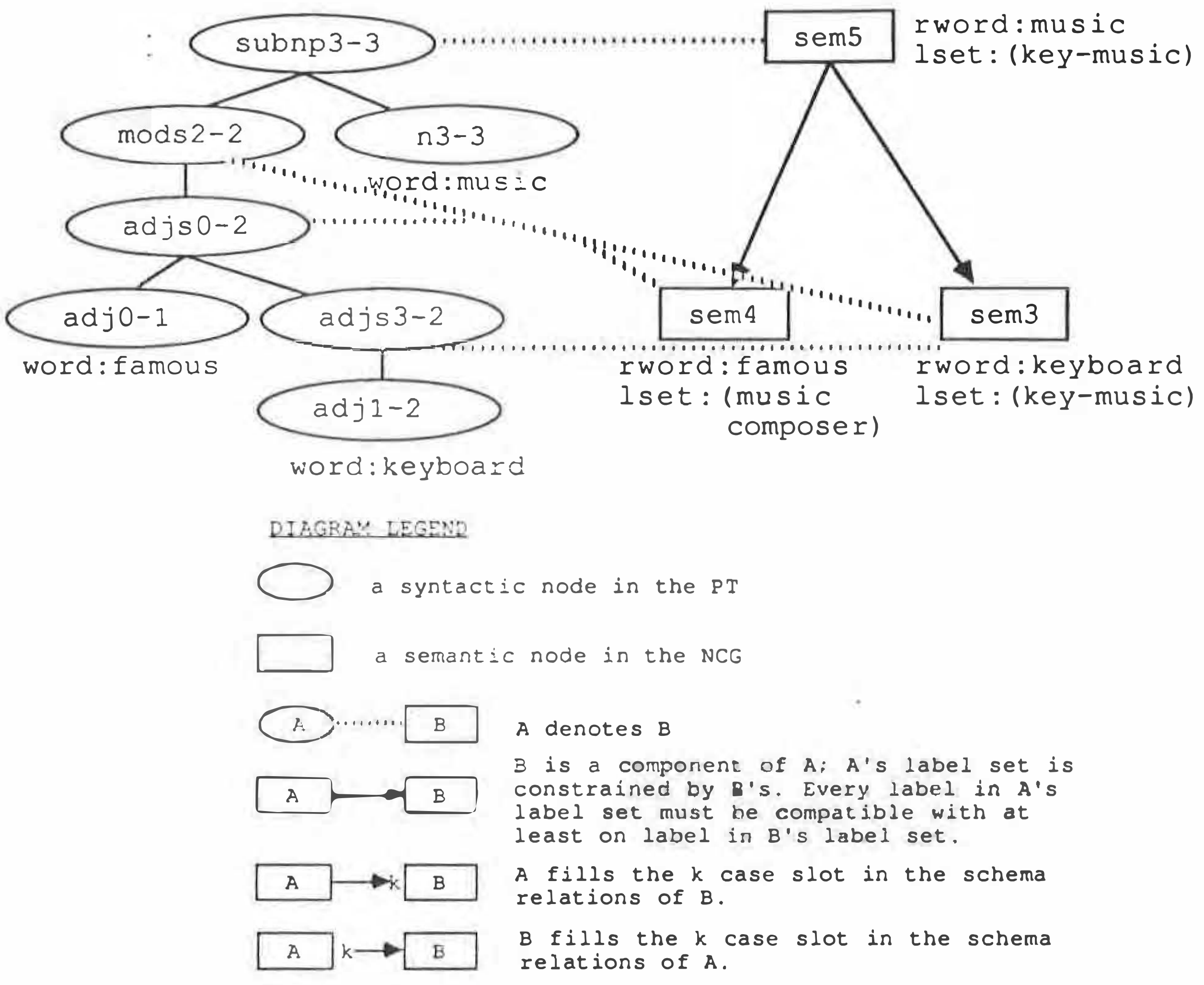
subnp3-3
sem5
rword:music
lset:(key-music)
mods2-2
n3-3
word:music
adjs0-2
adj0-1
adjs3-2
sem4
sem3
word:famous
rword:famous
lset:(music composer)
rword:keyboard
lset:(key-music)
adj1-2
word:keyboard
DIAGRAM LEGEND
a syntactic node in the PT
a semantic node in the NCG
A
B
A denotes B
B is a component of A; A's label set is constrained by B's. Every label in A's label set must be compatible with at least on label in B's label set.
k
A fills the k case slot in the schema relations of B.
B fills the k case slot in the schema relations of A.

Figure 4.1: Representation for 'famous keyboard music'

and (**ballet symphony concerto key-music pro-music**) exist under the hierarchy of the two abstract labels, 'voc-music' and 'ins-music', and 'key-music' is a member of one of these lists. The consistency propagation process then causes these descendant labels to be retrieved and to replace (**voc-music ins-music**). All except the label 'key-music' are then deleted from the label set of 'sem5' because of incompatibility.

The second example, Figure 4.2, shows how possessive nouns differ from adjectives in the way they apply constriants on the main noun. A comparison of Figure 4.1 and Figure 4.2 illustrates this. In Figure 4.1, each adjective applies constraints on the main noun independently. In Figure 4.2, a chain relationship is established where each possessive noun only apply constraints on the immediately following noun. In the system, the noun phrases, 'father of Mozart' and 'Mozart's father', are equivalent in meaning. Although the syntactic structures generated for these two noun phrases are different, the semantic representations are the same. Thus, the NCG in Figure 4.2 represents both the noun phrases, 'Mozart's father's birthday' and 'birthday of the father of Mozart'.

At the beginning of the recognition process, multiple syntactic nodes of the syntactic class, **subnp**, are active. Each node is predicted to correspond with a particular composition rule that resides in the model syntactic schema of **subnp**. When 'Mozart's' is scanned from the input, only the syntactic nodes that are derived from composition rules with the grammar constituent, **mods**, are retained. The semantic nodes, 'sem0' and 'sem1', are built at this point. An **sp constraint** is applied to create a link between the two nodes to represent all the objects that are 'of Mozart'. For example, 'father of Mozart', 'music of Mozart' and 'birthday of Mozart'.

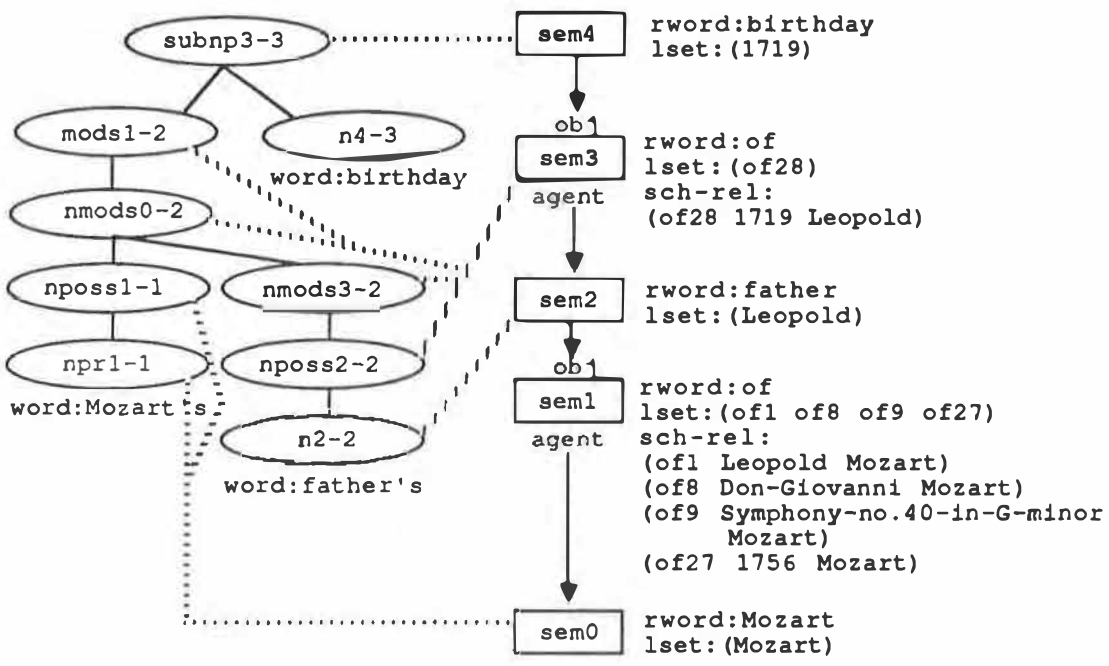

Figure 4.2: Representation for 'Mozart's father's birthday'

The label set of 'sem1' is refined from the list of all 'of' relationships to those with 'Mozart' as the agent in the schema relations. Subsequent scanning of the input word 'father's' results in the building of semantic nodes 'sem2' and 'sem3'. A similar link is established between these two nodes to reflect in the label set of 'sem3' all the 'of' relationships pertaining to all the fathers in the KB. As the parsing process discovers that there are no more possessive nouns, the link between 'sem2' and 'sem1' is made. Consistency checks then refine the label set of 'sem2' to just the father of Mozart, Leopold. Lastly, the input word 'birthday' is scanned and 'sem4' is built. When 'mods1-2' and 'n4-3' have both completed to 'subnp3-3' successfully, the link between 'sem4' and 'sem3' is made. The label set of 'sem4' is then refined from the list of birthdays of every person in the KB to only the birthday of Mozart's father.

The third example is a complete sentence with a prepositional phrase and a determiner. Figure 4.3 shows the representation for this sentence, 'which three sonatas were written by Beethoven'. Multiple prepositional phrases also apply independent constraints on a verb or noun as in the case of multiple adjectives. Both the PT and the NCG are built incrementally as in previous examples. The input words, 'which' and 'sonatas', cause the semantic nodes 'sem0' and 'sem1' to be built after the words have been scanned. The determiner, 'three', does not cause a semantic node to be constructed. Instead, it applies a number constraint on the cardinality of the label set of 'sem1' after the entire NCG has been built. In this case, there are exactly three sonatas in the label set; therefore, the constraint is satisfied. The input word, 'which', represents initially all the objects intensionally via its label set. Thus, the label set of 'sem0' is originally (`composer music father date place`). It is refined to the list of all sonatas

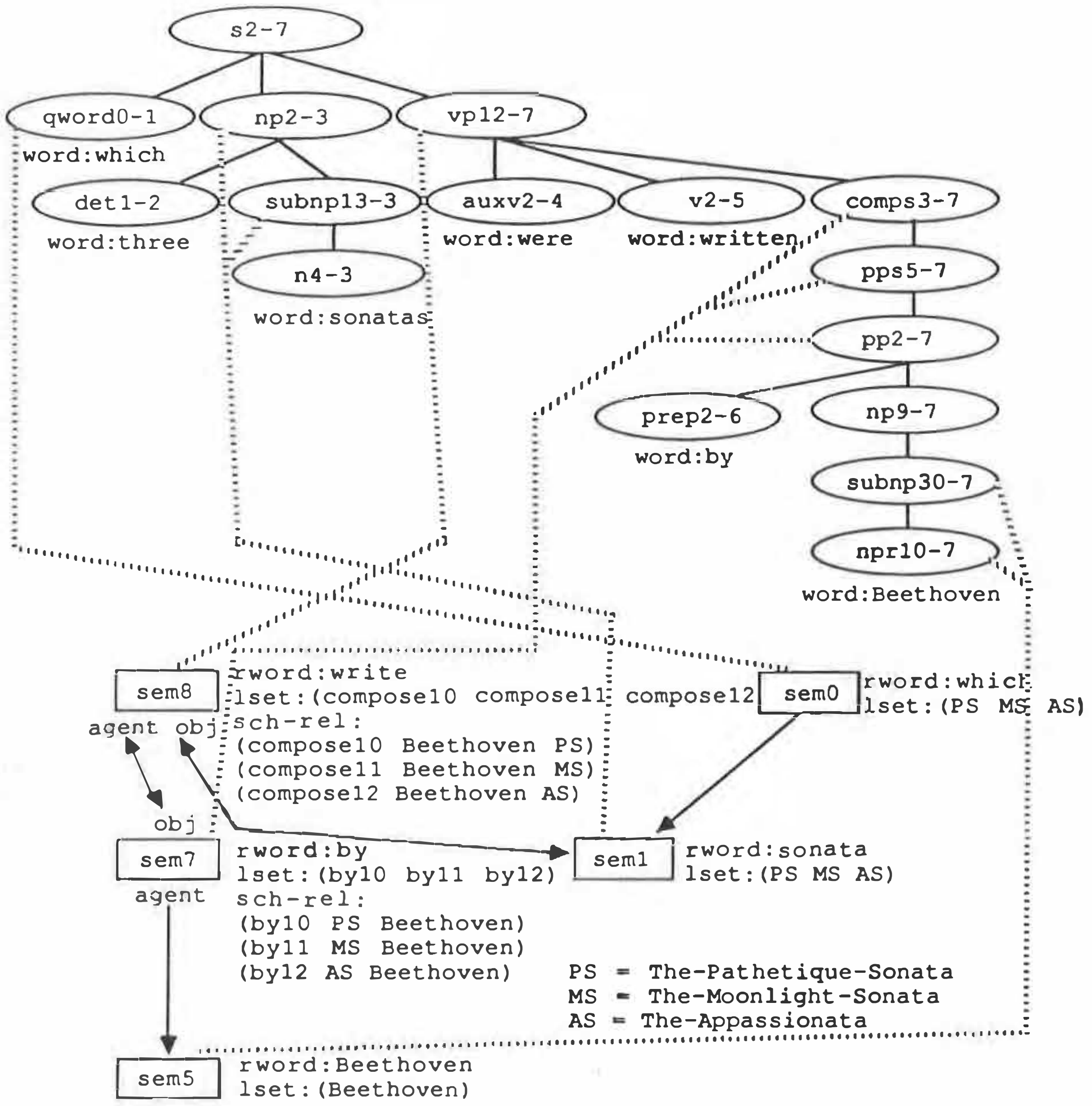

Figure 4.3: Representation for 'which three sonatas were written by Beethoven'

when the input word 'sonatas' is scanned and a link is made between 'sem0' and 'sem1'. The node, 'sem8', representing the verb 'write' is built after the words 'were written' are scanned. Its label set contains all writing events initially which are represented by schema relations. It is refined to those writing events that have Beethoven as the agent after the prepositional phrase has been scanned and a link is made between 'sem8' and 'sem7'. The node, 'sem7', has its label set refined to those fabricated labels whose schema relations have Beethoven as an agent when the link between 'sem7' and 'sem5' is made. When all the grammar constituents of 's2-7', the sentence derived syntactic schema, have completed to it, the link between 'sem8' and 'sem1' is made. The label set of 'sem1' is then refined to only those sonatas written by Beethoven. Since 'sem1' is a component of 'sem0', the constraint propagation process causes the label set of 'sem0' which now contains all the sonatas in the KB to be further refined to the same sonatas that are listed in the label set of 'sem1'. This actually provides the answer to the question.

The last example, illustrated in Figure 4.4 and 4.5, is also a complete sentence. The sentence is 'who composed the Messiah in Dublin'. It is chosen to demonstrate how multiple PT and NCG representations are generated for a sentence which has multiple parses or interpretations. In this case, the ambiguity lies in the attachment of the prepositional phrase to either the noun or the verb. Two diagrams are drawn separately for this example for clarity reasons although some nodes in the two PTs and NCGs are actually shared. The two interpretations yield different syntactic as well as semantic structures. In both diagrams, a box is drawn around the semantic nodes representing the prepositional phrase to help identify the two possible attachments of those nodes to the rest of the NCG. The semantic nodes, 'sem13' and 'sem12', have (temp locn) shown as a case slot to be filled by 'sem10' in the diagram. This actually means that 'sem10' should fill either the temporal or the locative case of the preposition 'in' within the two different possible NCGs of Figures 4.4 and 4.5 respectively. 'sem12' also has a case slot shown as **event(obj)** which is actually a combination of two cases. This means that 'sem14' should fill the 'obj' slot of the fabricated label 'compose1' which itself fills the **event slot** of 'in2' in 'sem12'.

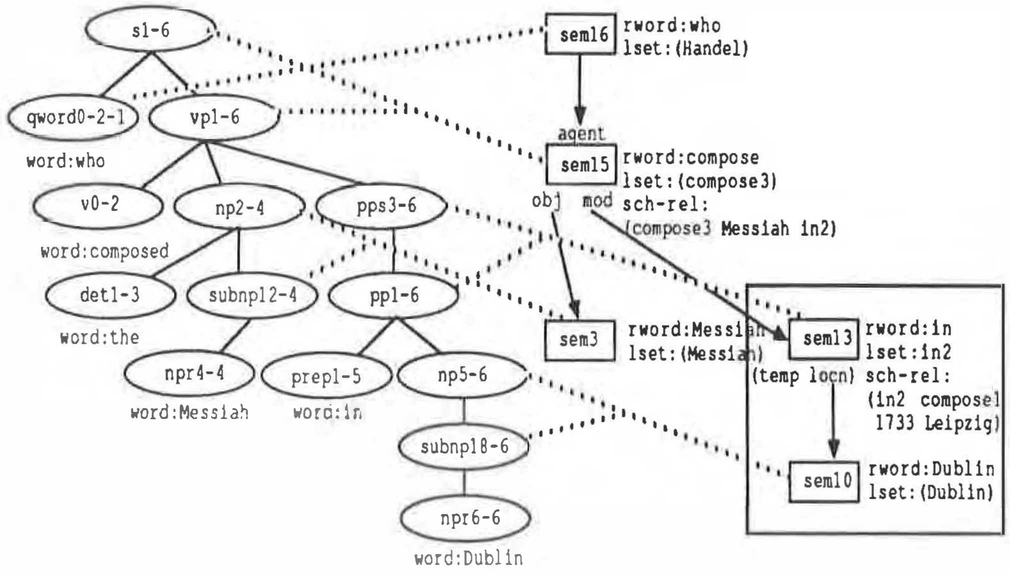

Figure 4.4: A representation for 'who composed the Messiah in Dublin'

At the beginning of the parsing process, there are multiple active syntactic nodes for the syntactic category s. After the word 'who' has been scanned from the input, only two sets of predictions that involve an interrogative sentence are retained. The semantic node, 'sem16', is then built and has the label set, (**composer**), which represents all the people in the KB

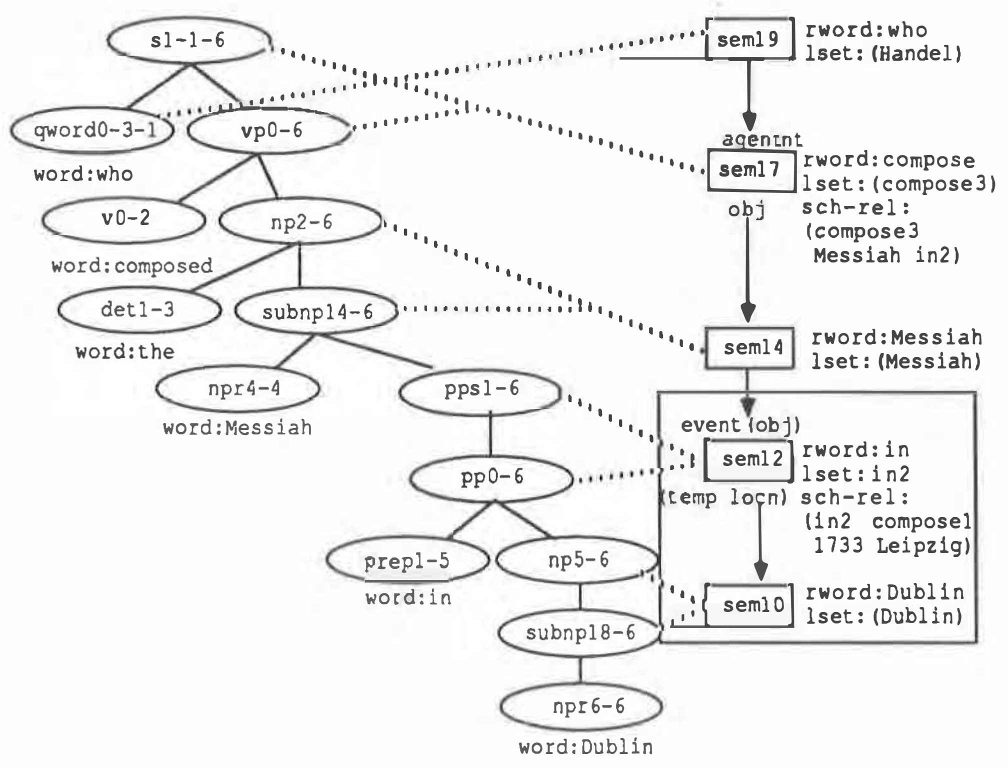

Figure 4.5: A second representation for 'who composed the Messiah in Dublin'

intensionally. One set of predictions for an interrogative sentence predict a **np** to follow the question word, and the other set predicts a **vp** to follow. When the word 'composed' is scanned, the syntactic nodes involved with the incorrect prediction of a **np** are garbage collected. The correct prediction of a **vp** thus causes 'sem15' to be built to represent the verb, 'compose', with a label set of all composing events. Multiple **vp** syntactic nodes are active at this point to indicate the different possible verb phrases that may exist. 'vp0-6' and 'vp1-6' in the two diagrams respectively are two of these possible predicted syntactic nodes for representing a **vp**. Next, 'Messiah' is scanned from the input and a semantic node representing it is built.

At this point, there is still only one NCG, with three unconnected semantic nodes representing the words, 'who', 'compose' and 'Messiah', respectively. Since a **np** may or may not contain a prepositional phrase, there are two branches of predictions involved. One branch completes the **np** to its **vp** parent, and the other is involved in further predictions of prepositional phrases attached to a noun phrase. The **np** that completes has two parents, 'vp0-6' and 'vp1-6'; therefore, multiple parses are possible and the NCG representation is split at this point. Copies of semantic nodes are made to allow for the splitting. The link between the semantic node representing 'Messiah' and the semantic node representing 'compose' is made in both NCGs. As the interpretation process continues and the prepositional phrase is scanned, semantic nodes are built to represent the **pp**. The completion of the **pp** semantic structure would cause a copy of it to be made since it has two parents, one is a **np** and the other is a **vp**. The link between 'sem15' and 'sem13' in Figure 4.4 is made at this point. The link between 'sem14' and 'sem12' for the other NCG in Figure 4.5 is also made at the same time. Input has now been exhausted and both **vp** syntactic nodes complete to their parents. Since two parses exist, the syntactic node for **s** is copied, each having a different **vp** syntactic node as its child. The semantic node for the question word 'who' is also copied at this point. One copy is linked to 'sem15' in one NCG, and the other is linked to 'sem17' in the other NCG. In both cases, the label set of the semantic node representing 'who' is refined from the label set, **(composer)**, which represents all composers intensionally to just **(Handel)**. This yields the answer to the question. Generally, multiple parses do not have to yield the same answer.

Chapter 5

Conclusions and Future Directions

In this chapter, the merits of the schema and constraint-based approach to understanding English sentences are reviewed. In addition, possible extensions to the system are mentioned.

Traditional approaches to understanding English sentences have followed the linear paradigm that involves three phases in the analysis of a sentence. First, the input sentence is parsed by a syntactic parser to form a parse tree, then a semantic analyser takes the parse tree as input and produces a database query in a prespecified formal language. Lastly, this query is analysed by an evaluator which finds the answer from the given database. This approach is still adopted by many researchers in logic programming as described in section 2.2.. The major drawback of this approach is in its inability to deal with ambiguities efficiently. If there are multiple possible parses for a sentence, then one must be selected to allow the processing to continue. In the event that this parse is incorrect as discovered later by the semantic analyser or the database evaluator, it is necessary for the system to back up and find another parse. A reliance on automatic backtracking makes this approach highly inefficient because of the thrashing behaviour of backtracking [Mackworth, 1977a]. Similarly, if the ambiguity exists in words having multiple senses, then one word sense must be chosen for each ambiguous word, and again backtracking is utilized if the wrong word senses have been chosen first.

Our approach handles both structural and word sense ambiguities much more gracefully and efficiently. All possible parses are explored in parallel and thus no backtracking is ever needed. Multiple word senses are all incorporated in the label set of a semantic schema. The inappropriate senses that do not satisfy all the existing constraints are removed through the application of network consistency. This approach remains non-committal to any of the ambiguous parse tree structures and word senses until enough evidence is found to decide on the correct interpretations. Although all parses and word senses are processed in parallel, Earley's algorithm is still efficient. It is the sharing of structures in both the PT and the NCG that makes this parallel approach feasible. In addition, label sets represent objects of the world intensionally, thus making this approach efficient.

This thesis indicates that this non-committal approach of taking into account all parses and all word senses until input evidence says otherwise is more plausible than the traditional selective processing and backtracking approach. The traditional approach involves a three-pass process whereas the schema and constraint-based approach is a one-pass, non-deterministic process where syntactic and semantic interpretations are carried out in parallel. In fact, this one-pass versus three-pass processing is part of a long standing argument in the linguistics field about whether semantics affect parsing. [Flores d'Arcais & Schreuder, 1983] give a brief history on this issue and is summarised below.

In the past, computing a structural description for a sentence was considered to be vital to the understanding of a sentence, and syntax was the main concern. Semantics were considered by linguists to be computed using a syntactic representation. This forms the basis of the three-pass processing approach. However, there has been an increasing interest in the relationships of grammatical representations to meaning structures, and models of sentence processing where semantics affect parsing became popular. In spite of this, linguists themselves are not unanimous as to whether semantics processing should be separated or intermingled with syntactic processing. Also, it is still an issue in psycholinguistics as to whether syntactic analysis of sentences constitutes a separate and autonomous stage in their perception. As Fodor says, "linguistic form recognition can't be context-driven because context doesn't determine form", and he knows of no convincing psychological evidence "that syntactic parsing is ever guided by the subject's appreciation of semantic context or of 'real world' background" [Fodor, 1983]. It is evident; however, from the point of view of efficiency in sentence processing, that semantic information be brought in as soon as possible in order to limit search.

Thus, this thesis takes a middle ground approach which splits the difference between strictly autonomous parsers and contextually driven ones. Our system builds the syntactic and semantic representations in parallel under the guidance of composition rules which contain grammatical constituents as well as syntactic and semantic predicates. The interpretation process is never driven by the semantic predicates, but if the semantic constraints are not satisfied, a possible parse under prediction by the grammar rules can be aborted. Therefore, although semantic information is never actually used to predict syntactic structures, it is introduced as soon as possible to help the system to narrow down its search for the correct parse. Fodor mentions in

[Fodor, 1983] that all the results on context effects in parsing that he knows of are compatible with this approach, and he believes that an approach of this flavour will prove ultimately to be valid.

A one-pass, non-deterministic approach is appropriate for resolving sentence ambiguities. However, predicting and carrying all possible parses and interpretations simultaneously can be inefficient in terms of space. There is a space and time tradeoff involved. This approach tries to limit hypotheses on syntactic and semantic representations by eliminating and garbage collecting the inappropriate predictions as soon as possible. The application of syntactic and semantic constraints with the use of network consistency ensures that incorrect hypotheses are eliminated as soon as input evidence precludes their existence. This process is known as specialization in Havens' theory of schema labelling [Havens, 1985]. Havens' theory also stresses the importance of combining the knowledge of composition with the specialization process. Our system follows this approach.

The knowledge of composition in our system exists in the schema knowledge base where the composition rules guide the interpretation process. Specialization is seen in the application of network consistency in refining the hypotheses to produce the final interpretation. The use of consistency techniques without knowledge of composition is inefficient as noted earlier. In a similar manner, a schema approach with knowledge of composition but without specialization is also inefficient. A schema approach may provide hypotheses to the composition of a sentence; however, it needs to search for the correct hypothesis. Backtracking may be utilized but it is highly inefficient. On the other hand, network consistency techniques are shown to be more efficient. Thus, combining the schema approach with network consistency techniques is a natural thing to do. In particular, we have chosen arc consistency for the specialization process because it is achievable in time linear in the number of constraints [Mackworth & Freuder, 1984]. In addition, the fact that our domain can be structured hierarchically has allowed us to utilize an implicit form of hierarchical arc consistency [Mackworth et al., 1985] which can be more efficient than arc consistency. Also, all the possible labels in each domain need not be represented explicitly but can be represented implicitly by one or more abstract labels. This intensional representation of label sets yields efficiency. The organization of these labels in a hierarchy has also given the system deductive power which is present in any semantic network formalism.

Logic is known to be a descriptively adequate formalism in that it is precise and is good for specification. However, logic programming based on the use of PROLOG usually relies on the automatic backtracking mechanism of PROLOG for control, and does not provide the structure for efficient computation. We have already mentioned the advantage of our approach in terms of efficiency. In addition, the composition rules in our syntactic schemata are precise and transparent as well. Although procedural linguistic knowledge is embedded in those rules to provide some control, it is not obscure as in the case of ATN (Augmented Transition Network). Also, our formalism lends itself rather well to the relaxation of some syntactic constraints. Any syntactic constraints can be removed from the composition rules without affecting the actions of the system if such relaxation is deemed desirable. This is possible since the syntactic constraints are not incorporated within procedures in the system. Relaxation of syntactic constraints allows the system to be more flexible, and to tolerate some grammatical errors in a sentence as humans often do.

Our adoption of Earley's context-free parsing algorithm is quite efficient in that subtrees that belong to more than one parse trees may be shared without redundancy in representation. However, this sharing only occurs if the subtree to be shared is predicted by the common lines of analysis in the same state set. Therefore, redundancy in the creation of similar subtree structures may still occur if a subtree created in one state set is predicted again in a later state set. An improvement can be made by adopting a similar context-free parsing algorithm by Tomita [Tomita, 1985] which claims to be more efficient in its representation of all parse trees in what he calls a shared-packed forest where the above problem does not arise.

Our system which only has a simple grammar, deals with a limited number of grammatical constructs, and works on a small domain. Thus, there is much room for development. The modularity of schemata allows ease of expansion as syntactic schemata can easily be added to provide a larger grammar with its added constraints. The difficulties lie in determining the semantic significance of more complex grammatical constructs such as adverbs, and how they should interact with other semantic entities in the sentence. Relationships between grammatical constructs in the semantic sense must be determined before the system can know how to build a NCG to represent them. It is beyond the scope of this thesis to determine the semantics of all grammatical constructs as linguists are still researching this topic. However, it suffices to say that improvements can be made to the current system with our existing knowledge of semantics. For example, determiners can be dealt with in a more thorough manner with the addition of scoping on the quantifiers and the utilization of fuzzy logic and sets. The system can also be extended to provide answers to questions instead of just outputting the syntactic and semantic representations. This involves selecting the consistent labels from the semantic nodes of the final NCG representation and producing a coherent answer from them with the addition of a sentence generation component.

Perhaps a most important enhancement to the system would be to allow additional facts to be added which would constitute a form of learning. This system will reject a sentence that is not supported by the given facts in the database. All or some of the label sets of the semantic nodes in the NCG will become empty to reflect this situation. However, the system may be altered to add the appropriate labels to the empty label sets for certain semantic entities such that rejection of the sentence would not occur. Once the representation is built, the knowledge base can be updated accordingly to reflect the new additions. For example, we present the sentence 'John is a composer' to the system where such a fact does not currently exist in the database. During the interpretation process of the system, a semantic node representing 'composer' will be built and it will have an abstract label representing all the composers in the database in its label set. When the system tries to link the semantic node representing 'John' to the one representing 'composer', consistency checking is performed. We will find that 'John' is not in the list of composers represented by the abstract label under the semantic node of 'composer' thus causing the abstract label to be deleted. At this point, the system should notify the user of this fact and ask whether the user desires to have 'John' be added to the list of composers. If the answer is in the affirmative, the label 'John' should be inserted in the now empty label set of the semantic node representing 'composer', and later be added to the label set of the word schema for 'composer' in the knowledge base to make it a permanent fact. In this case, the system knows about the verb 'is' and thus is capable of establishing the representational structures to allow for the addition of a given fact. Thus, it is essential that the system knows about the relationships between semantic entities before addition of facts can occur. Therefore, if a new fact contains a new verb not known to the system, it would not be possible for the system to perform such automatic additions to the knowledge base.

The approach introduced by this thesis could eventually be adapted to the understanding of stories and conversations as well. In these cases, more abstract schemata will exist in the knowledge base besides the simple word schemata. For example, schemata such as scripts and plans that represent situations will be needed. However, this approach would not require backtracking to search for the correct script to apply just as it would not use backtracking to search for the appropriate word sense when a word is ambiguous. Instead, all possible scripts would be processed in parallel and the system would refine on the appropriate ones when enough evidence is given in the input to decide on the correct ones. For example, let us consider the sentence 'John and Mary were walking down the aisle.' A standard approach is to select a script to apply to this situation [Schank & Abelson, 1977]. Assume that the system has chosen the 'marriage script'. However, if the next sentence is 'John took a can of apple juice off the shelf', then the system needs to backtrack and select the 'supermarket script' instead. In our approach, the system would not commit itself to either of the two scripts mentioned above after seeing the first sentence but would keep both of them around since they are both consistent with the information given in the sentence. When it sees the second sentence, enough constraints then exist to refine the interpretation to the 'supermarket script' and only at this point would the script be applied.

In summary, the schema and constraint-based approach to understanding natural language as introduced in this thesis has some merit. It can handle ambiguity more effectively than the traditional linear approach without utilising backtracking. It utilises semantics to aid parsing to increase efficiency and yet does not let semantics determine syntactic form. Thus, the approach is in middle ground concerning the issue of whether semantics affect parsing. Although the interpretation process of all parses is performed in parallel, inefficiency is avoided because intensional label sets are used and the constraint propagation process plus the knowledge of composition help to eliminate incorrect interpretations as early as possible. In addition, garbage collection is incorporated to give space efficiency. Lastly, extensions to the system are possible and it has the potential to develop into a story or conversation understanding system without sacrificing efficiency.

Bibliography

[Aho & Ullman, 1972] Aho, A. V. and Ullman, J. D. 1972. *The Theory of Parsing, Translation and Compiling*. Englewood Cliffs, N. J.: Prentice-Hall.

[Alon & Havens, 1985] Alon, A. and Havens, W. 1985. Recognizing VLSI circuits from mask artwork by schema labelling. Tech. Report 85-1, Dept. of Computer Science, Univ. of British Columbia, Vancouver, Canada.

[Bar-Hillel, 1960] Bar-Hillel, Y. 1960. The present status of automatic translation of languages. In F. L. Alt (Ed.), *Advances in Computers* (Vol. 1). New York: Academic Press, 91-163.

[Barr & Feigenbaum, 1981] Barr, A. and Feigenbaum, E. A. 1981. *The Handbook of Artificial Intelligence* (Vol. 1). Los Altos, CA.: William Kaufman, Inc.

[Bartlett, 1932] Bartlett, F. C. 1932. *Remembering*. Cambridge, England: Cambridge University Press.

[Bobrow, 1968] Bobrow, D. G. 1968. Natural language input for a computer problem-solving system. In M. Minsky (Ed.), *Semantic Information Processing*. Cambridge, Mass.: MIT Press, 146-226.

[Bobrow et al., 1977] Bobrow, D. G., Kaplan, R. M., Kay, M., Norman, D. A., Thompson, H. and Winograd, T. 1977. GUS, a frame-driven dialogue system. *Artificial Intelligence,* 8:155-173.

[Bobrow & Norman, 1975] Bobrow, D. G. and Norman, D. A. 1975. Some principles of memory schemata. In D. G. Bobrow and D. A. Collins (Eds.), *Representation and Understanding*. New York: Academic Press, 131-149.

[Bobrow & Raphael, 1974]	Bobrow, D. G. and Raphael, B. 1974. New programming languages for artificial intelligence research. *Computing Survey,* 6(3): 153-174.
[Bobrow & Winograd, 1977]	Bobrow, D. G. and Winograd, T. 1977. An overview of KRL, a knowledge representation language. *Cognitive Science,* 1:3-46.
[Brachman, 1978]	Brachman, R. J. 1978. A structural paradigm for representing knowledge. Report No. 3605, Bolt, Beranek and Newman, Inc., Cambridge, Mass.
[Brachman, 1979]	Brachman, R. J. 1979. On the epistemological status of semantic networks. In N. V. Findler (Ed.), *Associative Networks.* New York: Academic Press, 3-50.
[Brachman, 1982]	Brachman, R. J. 1982. What IS-A is and isn't. *Proc. Canadian Society for Computational Studies of Intelligence.* May 1982, 212-221.
[Brachman, 1983]	Brachman, R. J. 1983. What IS-A is and isn't: an analysis of taxonomic links in semantic networks. *IEEE Computer,* 16(10): 30-36.
[Brachman et al., 1983a]	Brachman, R. J., Fikes, R. E. and Levesque, H. J. 1983a. Krypton: integrating terminology and assertion. *AAAI-83,* 31-35.
[Brachman et al., 1983b]	Brachman, R. J., Fikes, R. E. and Levesque, H. J. 1983b. Krypton: a functional approach to knowledge representation. *IEEE Computer,* 16(10): 67-73.
[Brachman et al., 1985]	Brachman, R. J., Gilbert, V. P. and Levesque, H. J. 1985. An essential hybrid reasoning system: knowledge and symbol level accounts of KRYPTON. *IJCAI-85,* 532-539.
[Brachman & Levesque, 1982]	Brachman, R. J. and Levesque, H. J. 1982. Competence in knowledge representation. *AAAI-82,* 189-192.
[Brachman & Schmolze, 1985]	Brachman, R. J. and Schmolze, J. G. An overview of the KL-ONE knowledge representation system. *Cognitive Science,* 9(2): 171-216.
[Bult, 1986]	Bult, T. P. 1986. Schema labelling applied to hand-printed Chinese character recognition. M.Sc. Thesis, Dept. of

	Computer Science, Univ. of British Columbia, Vancouver, Canada (in preparation).
[Carbonell, 1970]	Carbonell, J. R. 1970. AI in CAI: an artificial intelligence approach to computer-assisted instruction. ***IEEE Transactions on Man-Machine Systems,*** MMS-11: 190-202.
[Colmerauer, 1978]	Colmerauer, A. 1978. Metamorphosis grammars. In L. Bolc (Ed.), ***Natural Language Communication with Computers.*** Berlin: Springer-Verlag, 133-189.
[Cullingford, 1978]	Cullingford, R. E. 1978. Script application: computer understanding of newspaper stories. Research Report 116, Dept. of Computer Science, Yale University.
[Dahl, 1981]	Dahl, V. 1981. Translating Spanish into logic through logic. *AJCL,* 7(3): 149-164.
[Dahl, 1983]	Dahl, V. 1983. Logic programming as a representation of knowledge. *IEEE Computer,* 16(10): 61-65.
[Dahl & McCord, 1983]	Dahl, V. and McCord, M. C. 1983. Treating coordination in logic grammars. *AJCL,* 9(2): 69-91.
[Earley, 1970]	Earley, J. 1970. An efficient context-free parsing algorithm. *Communications of ACM,* 6(8): 94-102.
[Fikes & Kehler, 1985]	Fikes, R. and Kehler T. 1985. The role of frame-based representation in reasoning. *Communications of the ACM,* 28(9): 904-920.
[Fillmore, 1968]	Fillmore, C. 1968. The case for case. In E. Bach and R. T. Harms (Eds.), ***Universals in Linguistic Theory.*** Chicago: Holt, Rinehart and Winston, 1-90.
[Flores d'Arcais & Schreuder, 1983]	Flores d'Arcais, G. B. and Schreuder, R. 1983. The process of language understanding: a few issues in contemporary psycholinguistics. In G. B. Flores d'Arcais and R. J. Jarvella (Eds.), ***The Process of Language Understanding.*** New York: Wiley and Sons, 1-41.
[Fodor, 1983]	1983. Fodor, J. A. 1983. ***The Modularity of Mind.*** Cambridge, Massachusetts: MIT Press.
[Freuder, 1978]	Freuder, E. C. 1978. Synthesizing constraint expressions. *Communications of ACM,* 21(11): 958-966.

[Green et al., 1963] Green, B. F., Jr., Wolf, A. K., Chomsky, C. and Laughery, K. 1963. BASEBALL: An automatic question answerer. In E. A. Feigenbaum and J. Feldman (Eds.), *Computers and Thought*. New York: McGraw-Hill, 207-216.

[Green, 1969] Green, C. C. 1969. The application of theorem-proving to question-answering systems. *IJCAI-69,* 219-237.

[Grosz, 1977] Grosz, B. J. 1977. The representation and use of focus in a system for understanding dialogue. *IJCAI-77,* 67-76.

[Havens, 1983] Havens, W. 1983. Recognition mechanisms for schema-based knowledge representations. *Int. Journal of Computers and Mathematics 9, no.1.* Pergamon Press: 185-199.

[Havens, 1985] Havens, W. 1985. A theory of schema labelling. *Computational Intelligence,* 1(3):127-139.

[Havens & Mackworth, 1983] Havens, W. and Mackworth, A. K. 1983. Representing knowledge of the visual world. *IEEE Computer,* 16(10): 90-98.

[Hayes, 1977] Hayes, Pat J. 1977. In defence of logic. *IJCAI-77,* 559-565.

[Hayes, 1981] Hayes, Pat J. 1981. The logic of frames. In B. L. Webber and N. J. Nilsson, *Readings in Artificial Intelligence*. Palo Alto: Tioga Pub. Co., 451-458.

[Hayes, 1977] Hayes, Philip J. 1977. On semantic nets, frames and associations. *IJCAI-77,* 99-107.

[Hendrix, 1975] Hendrix, G. G. 1975. Expanding the utility of semantic networks through partitioning. *IJCAI-75,* 115-121.

[Hendrix, 1979] Hendrix, G. G. 1979. Encoding knowledge in partitioned networks. In N. V. Findler (Ed.), *Associative Networks*. New York: Academic Press, 51-92.

[Hewitt, 1972] Hewitt, C. 1972. Description and theoretical analysis (using schemata) of PLANNER, a language for proving theorems and manipulating models in a robot. Report No. TR-258, AI Lab., M.I.T., Cambridge, MA.

[Kehler & Clemenson, 1984] Kehler, T. P. and Clemenson, G. D. 1984. An application development system for expert systems. *System Software,* 3(1): 212-224.

[Levesque & Mylopoulos, 1979]	Levesque, H. and Mylopoulos, J. 1979. A procedural semantics for semantic networks. In N. V. Findler (Ed.), *Associative Networks*. New York: Academic Press, 93-120.
[Lindsay, 1963]	Lindsay, R. K. 1963. Inferential memory as the basis of machines which understand natural language. In E. A. Feigenbaum and J. Feldman (Eds.), *Computers and Thought*. New York: McGraw-Hill, 217-233.
[Mackworth, 1977a]	Mackworth, A. K. 1977a. Consistency in networks of relations. *Artificial Intelligence,* 8: 99-118.
[Mackworth, 1977b]	Mackworth, A. K. 1977b. On reading sketch maps. *IJCAI-77,* 598-606.
[Mackworth & Freuder, 1984]	Mackworth, A. K. and Freuder, E. C. 1984. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence,* 25: 5-74.
[Mackworth & Havens, 1981]	Mackworth, A. K. and Havens, W. 1981. Structuring domain knowledge for visual perception. *IJCAI-81,* 625-627.
[Mackworth et al., 1985]	Mackworth, A. K., Mulder, J. A. and Havens, W. S. 1985. Hierarchical arc consistency: exploiting structured domains in constraint satisfaction problems. Tech. Report 85-7, Dept. of Computer Science, Univ. of British Columbia, Vancouver, Canada.
[McAllester, 1980]	McAllester, D. A. 1980. An outlook on truth maintenance. AI Memo 551, AI Lab., M.I.T., Cambridge, MA.
[McAllester, 1982]	McAllester, D. A. 1982. Reasoning utility package user's manual. AI Memo 667, AI Lab., M.I.T., Cambridge, MA.
[McCarthy, 1977]	McCarthy, J. 1977. Epistemological problems of Artificial Intelligence. *IJCAI-77,* 1038-1044.
[McCord, 1982]	McCord, M. C. 1982. Using slots and modifiers in logic grammars for natural language. *Artificial Intelligence,* 18(3): 327-367.
[Minsky, 1975]	Minsky, M. 1975. A framework for representing knowledge. In P. Winston (Ed.), *The Psychology of Computer Vision*. New York: McGraw-Hill, 211-277.

[Montanari, 1974] Montanari, U. 1974. Network of constraints: fundamental properties and applications in picture processing. *Information Science,* 7: 95-132.

[Mulder, 1985] Mulder, J. 1985. Using discrimination graphs to represent visual knowledge. Tech. Report 85-14, Dept. of Computer Science, Univ. of British Columbia, Vancouver, Canada.

[Pereira, 1981] Pereira, F. 1981. Extraposition grammars. *AJCL,* 7:243-256.

[Pereira & Warren, 1980] Pereira, F. and Warren, D. 1980. Definite clause grammars for language analysis – a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence,* 13(3): 231-278.

[Quillian, 1968] Quillian, R. 1968. Semantic memory. In M. Minsky (Ed.), *Semantic Information Processing.* MIT Press, 227-270.

[Quillian, 1969] Quillian, R. 1969. The teachable language comprehender: a simulation program and the theory of language. *Communications of the ACM,* 12: 459-476.

[Raphael, 1968] Raphael, B. 1968. SIR: A computer program for semantic information retrieval. In M. Minsky (Ed.), *Semantic Information Processing.* Cambridge, Mass.: MIT Press, 33-145.

[Rumelhart & Norman, 1973] Rumelhart, D. E. and Norman, D. 1973. Active semantic networks as a model of human memory. *IJCAI-73,* 450-457.

[Rumelhart & Ortony, 1976] Rumelhart, D. E. and Ortony, A. 1976. The representation of knowledge in memory. TR-55, Centre for Human Info. Processing, Dept. of Psych., Univ. of Calif. at San Diego, LaJolla, CA.

[Schank, 1975] Schank, R. C. 1975. The structure of episodes in memory. In D. G. Bobrow and D. A. Collins (Eds.), *Representation and Understanding.* New York: Academic Press, 237-272.

[Schank & Abelson, 1977] Schank, R. C. and Abelson, R. P. 1977. *Scripts, Plans, Goals, and Understanding.* Hillsdale, N. J.: Lawrence Erlbaum Associates.

[Schank et al., 1975]	Schank, R., and Yale AI Project. 1975. SAM – a story understander. Research Report 43, Dept. of Computer Science, Yale University.
[Schubert, 1976]	Schubert, L. K. 1976. Extending the expressive power of semantic networks. *Artificial Intelligence,* 7(2): 163-198.
[Schubert et al., 1979]	Schubaert, L. K., Goebel, R. G. and Cercone, N. J. 1979. The structure and organization of a semantic net for comprehension and inference. In N. V. Findler (Ed.), *Associative Networks*. New York: Academic Press, 121-175.
[Shapiro, 1971]	Shapiro, S. C. 1971. A net structure for semantic information storage, deduction and retrieval. *IJCAI-71,* 512-523.
[Shapiro, 1979]	Shapiro, S. C. 1979. The SNePs semantic network processing system. In N. V. Findler (Ed.), *Associative Networks.* New York: Academic Press, 179-203.
[Shapiro, 1982]	Shapiro, S. C. 1982. Generalized augmented transition network grammars for generation from semantic networks. *AJCL,* 8(1): 12-25.
[Simmons, 1973]	Simmons, R. 1973. Semantic networks: their computation and use for understanding English sentences. In Schank R. C. and Colby K. M. (Eds.), *Computer Models of Thought and Language*. San Francisco: Freeman.
[Simmons & Slocum, 1972]	Simmons, R. and Slocum, J. 1972. Generating English discourses from semantic networks. *Communications of the ACM,* 15(10): 891-905.
[Tomita, 1985]	Tomita, M. 1985. An efficient context-free parsing algorithm for natural languages. *IJCAI-85,* 756-763.
[Vilain, 1985]	Vilain, M. 1985. The restricted language architecture of a hybrid representation system. *IJCAI-85,* 547-551.
[Walker, 1978]	Walker, D. E. (Ed.) 1978. *Understanding Spoken Language*. New York: North Holland.
[Waltz, 1972]	Waltz, D. E. 1972. Generating semantic descriptions of scenes with shadows. Tech. Report MAC AI-TR-271, MIT, Cambridge, MA.

[Warren & Pereira, 1982] Warren, D. and Pereira, F. 1982. An efficient easily adaptable system for interpreting natural language queries. *AJCL,* 8(3-4): 110-119.

[Weaver, 1949] Weaver, W. 1949. Translation. In W. N. Locke and A. D. Booth (Eds.), ***Machine Translation of Languages.*** New York: Technology Press of MIT and Wiley and Sons (1955), 15-23.

[Weizenbaum, 1966] Weizenbaum, J. 1966. ELIZA – A computer program for the study of natural language communication between man and machine. ***Communications of the ACM,*** 9: 36-45.

[Wilensky, 1978] Wilensky, R. 1978. Understanding goal-based stories. Research Report No. 140., Dept. of Computer Science, Yale University.

[Winograd, 1972] Winograd, T. 1972. ***Understanding Natural Language.*** New York: Academic Press.

[Winograd, 1975] Winograd, T. 1975. Frame representations and the declarative/procedural controversy. In D. G. Bobrow and A. Collins (Eds.), ***Representation and Understanding.*** New York: Academic Press, 185-210.

[Winograd, 1980] Winograd, T. 1980. What does it mean to understand language? ***Cognitive Science,*** 4:209-241.

[Winograd, 1983] Winograd, T. 1983. ***Language as a Cognitive Process Vol. 1: Syntax.*** Reading, Mass.: Addison-Wesley.

[Woods, 1968] Woods, W. A. 1968. Procedural semantics for a question-answering machine. ***Fall Joint Computer Conference,*** 33: 457-471.

[Woods, 1970] Woods, W. A. 1970. Transition network grammars for natural language analysis. ***Commuinication of the ACM,*** 13(10): 591-606.

[Woods, 1975] Woods, W. A. 1975. What's in a link: Foundations for semantic networks. In D. G. Bobrow and A. Collins (Eds.), ***Representation and Understanding.*** New York: Academic Press, 35-82.

[Woods et al., 1972] Woods, W. A., Kaplan, R., and Nash-Webber, B. 1972. The lunar sciences natural language information system: Final report. BBN Report No. 2378, Bolt, Beranek and Newman, Inc., Cambridge, Mass.

[Woods et al., 1976] Woods, W. A., et al. 1976. Speech understanding systems: Final report. BBN Report No. 3438, Bolt, Beranek and Newman, Inc., Cambridge, Mass.

Appendix

The appendix contains the listing of the KB. The first section is a list of all the model syntactic schemata and the second section is a list of all the model semantic schemata. The syntax and semantics of these schemata are explained in Section 3.2.

Listing of model syntactic schemata:

```
(s     ((*dot np vp (nvagree np vp) (sp vp np agent) (sp np vp agent)
                  (checknum vp))
        (*dot qword (build qword) vp (nvagree qword vp (iftrans)) (setptr)
              (sp qword vp ((in (temp locn))(who agent)(where mod)
                            (when mod)(what obj)))
              (checknum vp))
        (*dot qword (rwordis (which what)) np (sp qword np) vp
              (nvagree np vp (ifnosubj)) (sp vp np (depends))
              (sp np vp (rdepends)) (checknum vp))
        (*dot auxv np (nvagree np auxv) vp (vvagree auxv vp)
              (sp vp np agent) (sp np vp agent) (checknum vp))
        (*dot auxv (rwordis (be)) np (nvagree np auxv) comps
              (nvagree comps auxv) (sp np comps) (sp comps np)
              (checknum np) (checknum comps))))
(np    ((*dot det subnp (dnagree det subnp) (setptr))
        (*dot subnp (setptr))))
(subnp ((*dot npr (isnotposs) (build))
        (*dot n (isnotposs) (build))
        (*dot npr (isnotposs) (build) pps
              (sp subnp pps ((in event) (compose obj))))
        (*dot mods n (isnotposs) (build) (sp subnp mods ((of obj))))
        (*dot n (isnotposs) (build) pps
              (sp subnp pps ((from agent) (by obj) (of obj))))
        (*dot mods n (isnotposs) (build) (sp subnp mods ((of obj)))
              pps (sp subnp pps ((from agent)(by obj)(of obj))))))
(vp    ((*dot v (hasfeature (trans)) (build) np (sp vp np obj))
        (*dot v (hasfeature (trans)) (build) np (sp vp np obj) pps
              (sp vp pps mod))
        (*dot v (hasfeature (intrans)) (build))
        (*dot v (hasfeature (intrans)) (build) pps (sp vp pps mod)
              (sp pps vp ((in event)(of obj))))
        (*dot v (hasfeature (cop)) (gparhasno (auxv)) comps
              (nvagree comps v) (setptr))
        (*dot auxv (gparhasno (auxv)) v (vvagree auxv v) (build) comps
              (nvagree comps auxv (ifcop))
              (sp vp comps ((in mod)(by agent)))
              (sp comps vp ((in event)(of obj)(by obj))))
        (*dot auxv (gparhas (qword)) np (nvagree np auxv) v
              (vvagree auxv v) (build) (sp vp np agent))))
(comps ((*dot pps (setptr))
        (*dot adj (build))
        (*dot np (setptr))))
(mods  ((*dot nmods (setptr3) adjs (setptr))
        (*dot nmods (setptr3))
        (*dot adjs (setptr))))
(nmods ((*dot nposs nmods (sp (schild nmods) nposs obj) (setptr3))
        (*dot nposs (setptr3))))
(adjs  ((*dot adj (build) adjs (setptr))
        (*dot adj (build))))
(nposs ((*dot n (isposs) (build n) (build nposs of) (sp nposs n agent))
        (*dot npr (isposs) (build npr) (build nposs of)
        (sp nposs npr agent))))
(pps   ((*dot pp (setptr3) pps (setptr))
        (*dot pp (setptr3))))
(pp    ((*dot prep (build) np (sp pp np ((from locn)(by agent)
                                         (in (temp locn))(of agent))))))
(qword ((*dot *term)))
(auxv  ((*dot *term)))
(v     ((*dot *term)))
(det   ((*dot *term)))
(adj   ((*dot *term)))
(npr   ((*dot *term)))
(n     ((*dot *term)))
(prep  ((*dot *term)))
```

Listing of model semantic schemata:

```
(composer n s (Bach Handel Haydn Mozart Beethoven Chopin Berlioz
               Tchaikovsky Verdi))
(Bach npr * (Bach))
```

```
(Handel npr * (Handel))
(Haydn npr * (Haydn))
(Mozart npr * (Mozart))
(Beethoven npr * (Beethoven))
(Chopin npr * (Chopin))
(Berlioz npr * (Berlioz))
(Tchaikovsky npr * (Tchaikovsky))
(Verdi npr * (Verdi))
(Leopold npr * (Leopold))

(music n * (voc-music ins-music))
(voc-music pn * (oratorio opera mass))
(ins-music pn * (ballet symphony concerto key-music pro-music))
(key-music pn * (sonata polonaise waltz Well-Tempered-Clavier))
(pro-music pn * (overture-fantasy))
(oratorio n s (Messiah The-Creation))
(opera n s (Giulio-Cesare Don-Giovanni La-Traviata Aida Otello
           Falstaff))
(mass n es (Mass-in-B-minor))
(ballet n s (The-Nutcracker))
(symphony n es (The-Surprise-Symphony Symphony-no.40-in-G-minor
               The-Fifth-Symphony Symphonie-Fantastique))
(concerto n s (The-Emperor-Concerto))
(sonata n s (The-Pathetique-Sonata The-Moonlight-Sonata
            The-Appassionata))
(polonaise n s (Polonaise-in-A-flat))
(waltz n s (Valse-Brilliante))
(overture-fantasy n es (Romeo-and-Juliet))
(Well-Tempered-Clavier npr * (Well-Tempered-Clavier))
(Messiah npr * (Messiah))
(The-Creation npr * (The-Creation))
(Giulio-Cesare npr * (Giulio-Cesare))
(Don-Giovanni npr * (Don-Giovanni))
(La-Traviata npr * (La-Traviata))
(Aida npr * (Aida))
(Otello npr * (Otello))
(Falstaff npr * (Falstaff))
(Mass-in-B-minor npr * (Mass-in-B-minor))
(The-Nutcracker npr * (The-Nutcracker))
(The-Surprise-Symphony npr * (The-Surprise-Symphony))
(Symphony-no.40-in-G-minor npr * (Symphony-no.40-in-G-minor))
```

```
(The-Fifth-Symphony npr * (The-Fifth-Symphony))
(Symphonie-Fantastique npr * (Symphonie-Fantastique))
(The-Emperor-Concerto npr * (The-Emperor-Concerto))
(The-Pathetique-Sonata npr * (The-Pathetique-Sonata))
(The-Moonlight-Sonata npr * (The-Moonlight-Sonata))
(The-Appassionata npr * (The-Appassionata))
(Polonaise-in-A-flat npr * (Polonaise-in-A-flat))
(Valse-Brilliante npr * (Valse-Brilliante))
(Romeo-and-Juliet npr * (Romeo-and-Juliet))

(father n s (Leopold))
(date n s (birthday dday cday))
(birthday n s (|1685| |1719| |1732| |1756| |1770| |1803| |1810|
               |1813| |1840|))
(dday pn * (|1750| |1759| |1787| |1791| |1809| |1827| |1849| |1869|
           |1893| |1901|))
(cday pn * (|1733| |1742|))
(type n s (oratorio opera mass ballet symphony concerto sonata
           polonaise waltz overture-fantasy))
(place n s (birthplace dplace cplace))
(birthplace n s (Eisenach Halle Rohrau Salzburg Augsburg Bonn Warsaw
                 Grenoble Votkinsk Busseto))
(dplace pn * (Leipzig London Vienna Salzburg Paris St-Petersburg
             Milan))
(cplace pn * (Leipzig Dublin))
(|1685| npr * (|1685|))
(|1719| npr * (|1719|))
(|1732| npr * (|1732|))
(|1756| npr * (|1756|))
(|1770| npr * (|1770|))
(|1803| npr * (|1803|))
(|1810| npr * (|1810|))
(|1813| npr * (|1813|))
(|1840| npr * (|1840|))
(|1750| npr * (|1750|))
(|1759| npr * (|1759|))
(|1787| npr * (|1787|))
(|1791| npr * (|1791|))
(|1809| npr * (|1809|))
(|1827| npr * (|1827|))
(|1849| npr * (|1849|))
```

```
(|1869| npr * (|1869|))
(|1893| npr * (|1893|))
(|1901| npr * (|1901|))
(|1733| npr * (|1733|))
(|1742| npr * (|1742|))
(Eisenach npr * (Eisenach))
(Halle npr * (Halle))
(Rohrau npr * (Rohrau))
(Salzburg npr * (Salzburg))
(Augsburg npr * (Augsburg))
(Bonn npr * (Bonn))
(Warsaw npr * (Warsaw))
(Grenoble npr * (Grenoble))
(Votkinsk npr * (Votkinsk))
(Busseto npr * (Busseto))
(Leipzig npr * (Leipzig))
(London npr * (London))
(Vienna npr * (Vienna))
(Paris npr * (Paris))
(St-Petersburg npr * (St-Petersburg))
(Milan npr * (Milan))
(Dublin npr * (Dublin))
 (vocal adj * (oratorio opera mass))
(instrumental adj * (ballet symphony concerto sonata polonaise waltz
                     overture-fantasy Well-Tempered-Clavier))
(keyboard adj * (key-music))
(program adj * (pro-music))
(famous adj * (music composer))
(dead adj * (composer))

(who qword * (composer))
(when qword * (date))
(where qword * (place))
(which qword * (composer music father date place))
(what qword * (composer music father date place))

(the det (the (number any)))
(a det * (equal-num 1))
(one det * (equal-num 1))
(two det (two (number pl)) (equal-num 2))
(three det (three (number pl)) (equal-num 3))
```

```
(some det (some (number pl)) (morethan 1))

(from prep * (from1 from2 from3 from4 from5 from6 from7 from8 from9
              from10)
      (label locn agent)
      ((from1 Eisenach Bach) (from2 Halle Handel)
       (from3 Rohrau Haydn) (from4 Salzburg Mozart)
       (from5 Augsburg Leopold) (from6 Bonn Beethoven)
       (from7 Warsaw Chopin) (from8 Grenoble Berlioz)
       (from9 Votkinsk Tchaikovsky) (from10 Busseto Verdi)))
(by prep * (by1 by2 by3 by4 by5 by6 by7 by8 by9 by10 by11 by12 by13
            by14 by15 by16 by17 by18 by19 by20 by21 by22)
    (label obj agent)
    ((by1 Mass-in-B-minor Bach) (by2 Well-Tempered-Clavier Bach)
     (by3 Messiah Handel) (by4  Giulio-Cesare Handel)
     (by5 The-Surprise-Symphony Haydn) (by6 The-Creation Haydn)
     (by7 Don-Giovanni Mozart) (by8 Symphony-no.40-in-G-minor Mozart)
     (by9 The-Fifth-Symphony Beethoven)
     (by10 The-Pathetique-Sonata Beethoven)
     (by11 The-Moonlight-Sonata Beethoven)
     (by12 The-Appassionata Beethoven)
     (by13 The-Emperor-Concerto Beethoven)
     (by14 Polonaise-in-A-flat Chopin) (by15 Valse-Brilliante Chopin)
     (by16 Symphonie-Fantastique Berlioz)
     (by17 Romeo-and-Juliet Tchaikovsky)
     (by18 The-Nutcracker Tchaikovsky)
     (by19 La-Traviata Verdi) (by20 Aida Verdi) (by21 Otello Verdi)
     (by22 Falstaff Verdi)))
(in prep * (in1 in2 in3 in4 in5 in6 in7 in8 in9 in10 in11 in12 in13
            in14 in15 in16 in17 in18 in19 in20 in21 in22)
    (label event temp locn)
    ((in1 compose1 |1733| Leipzig) (in2 compose3 |1742| Dublin)
     (in3 born1 |1685| Eisenach) (in4 born2 |1685| Halle)
     (in5 born3 |1732| Rohrau) (in6 born4 |1756| Salzburg)
     (in7 born5 |1719| Augsburg) (in8 born6 |1770| Bonn)
     (in9 born7 |1810| Warsaw) (in10 born8 |1803| Grenoble)
     (in11 born9 |1840| Votkinsk) (in12 born10 |1813| Busseto)
     (in13 die1 |1750| Leipzig) (in14 die2 |1759| London)
     (in15 die3 |1809| Vienna) (in16 die4 |1791| Vienna)
     (in17 die5 |1787| Salzburg) (in18 die6 |1827| Vienna)
     (in19 die7 |1827| Vienna) (in20 die8 |1869| Paris)
     (in21 die9 |1893| St-Petersburg) (in22 die10 |1901| Milan)))
(of prep * (of1 of2 of3 of4 of5 of6 of7 of8 of9 of10 of11 of12 of13
            of14 of15 of16 of17 of18 of19 of20 of21 of22 of23 of24
            of25 of26 of27 of28 of29 of30 of31 of32 of33)
    (label obj agent)
    ((of1 Leopold Mozart)
     (of2 Mass-in-B-minor Bach) (of3 Well-Tempered-Clavier Bach)
     (of4 Messiah Handel) (of5 Giulio-Cesare Handel)
     (of6 The-Surprise-Symphony Haydn) (of7 The-Creation Haydn)
     (of8 Don-Giovanni Mozart) (of9 Symphony-no.40-in-G-minor Mozart)
     (of10 The-Fifth-Symphony Beethoven)
     (of11 The-Pathetique-Sonata Beethoven)
     (of12 The-Moonlight-Sonata Beethoven)
     (of13 The-Appassionata Beethoven)
     (of14 The-Emperor-Concerto Beethoven)
     (of15 Polonaise-in-A-flat Chopin)
     (of16 Valse-Brilliante Chopin) (of17 Symphonie-Fantastique Berlioz)
     (of18 Romeo-and-Juliet Tchaikovsky)
     (of19 The-Nutcracker Tchaikovsky)
     (of20 La-Traviata Verdi) (of21 Aida Verdi)
     (of22 Otello Verdi) (of23 Falstaff Verdi)
     (of24 |1685| Bach) (of25 |1685| Handel) (of26 |1732| Haydn)
     (of27 |1756| Mozart) (of28 |1719| Leopold) (of29 |1770| Beethoven)
     (of30 |1810| Chopin) (of31 |1803| Berlioz) (of32 |1840| Tchaikovsky)
     (of33 |1813| Verdi)))

(compose v s-d (trans)
   (compose1 compose2 compose3 compose4 compose5 compose6
    compose7 compose8 compose9 compose10 compose11 compose12
    compose13 compose14 compose15 compose16 compose17
    compose18 compose19 compose20 compose21 compose22)
   (label agent obj mod)
   ((compose1 Bach Mass-in-B-minor in1)
    (compose2 Bach Well-Tempered-Clavier)
    (compose3 Handel Messiah in2) (compose4 Handel Giulio-Cesare)
    (compose5 Haydn The-Surprise-Symphony) (compose6 Haydn The-Creation)
    (compose7 Mozart Don-Giovanni)
    (compose8 Mozart Symphony-no.40-in-G-minor)
    (compose9 Beethoven The-Fifth-Symphony)
    (compose10 Beethoven The-Pathetique-Sonata)
    (compose11 Beethoven The-Moonlight-Sonata)
    (compose12 Beethoven The-Appassionata)
    (compose13 Beethoven The-Emperor-Concerto)
    (compose14 Chopin Polonaise-in-A-flat)
    (compose15 Chopin Valse-Brilliante)
    (compose16 Berlioz Symphonie-Fantastique)
    (compose17 Tchaikovsky Romeo-and-Juliet)
    (compose18 Tchaikovsky The-Nutcracker)
    (compose19 Verdi La-Traviata) (compose20 Verdi Aida)
    (compose21 Verdi Otello)
    (compose22 Verdi Falstaff)))
(write v irr (trans) (compose))
(writes v (write (tns present) (pncode 3sg)))
(wrote v (write (tns past)))
(written v (write (pastpart)))
(born v (born (infin) (pastpart)) (intrans)
      (born1 born2 born3 born4 born5 born6 born7 born8 born9 born10)
      (label agent mod)
      ((born1 Bach in3) (born2 Handel in4) (born3 Haydn in5)
       (born4 Mozart in6) (born5 Leopold in7) (born6 Beethoven in8)
       (born7 Chopin in9) (born8 Berlioz in10) (born9 Tchaikovsky in11)
       (born10 Verdi in12)))
(die v s-d (intrans) (die1 die2 die3 die4 die5 die6 die7 die8 die9 die10)
     (label agent mod)
     ((die1 Bach in13) (die2 Handel in14) (die3 Haydn in15)
      (die4 Mozart in16) (die5 Leopold in17) (die6 Beethoven in18)
      (die7 Chopin in19) (die8 Berlioz in20) (die9 Tchaikovsky in21)
      (die10 Verdi in22)))
(be v irr)
(be auxv irr (cop))
(is auxv (be (tns present) (pncode 3sg)))
(are auxv (be (tns present) (pncode x13sg)))
(was auxv (be (tns past) (pncode 13sg)))
(were auxv (be (tns past) (pncode x13sg)))
(is v (be (tns present) (pncode 3sg)))
(are v (be (tns present) (pncode x13sg)))
(was v (be (tns past) (pncode 13sg)))
(were v (be (tns past) (pncode x13sg)))
(do auxv irr)
(does auxv (do (tns present) (pncode 3sg)))
(did auxv (do (tns past)))
```