The University of Alberta

RESPONSE GENERATION

by

Alison J. Bailes

A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree
of Master of Science

Department of Computing Science

Edmonton, Alberta
Spring, 1986

ISBN 0-315-30255-0

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Alison J. Bailes

TITLE OF THESIS: Response Generation

DEGREE FOR WHICH THIS THESIS WAS PRESENTED: Master of Science

YEAR THIS DEGREE GRANTED: 1986

(Signed) [illegible]

Permanent Address:
8702 Strathearn Drive
Edmonton, Alberta
Canada T6C 4C7

Dated November 15, 1985

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled **Response Generation** submitted by **Alison J. Bailes** in partial fulfillment of the requirements for the degree of **Master of Science**.

[illegible]

Supervisor

[illegible]

[illegible]

[illegible]

November 15, 1985

ABSTRACT

A natural language understanding system under development at the University of Alberta can answer yes-no questions using a resolution theorem prover. It was desired to supplement the initial YES or NO response produced by this system with an explanatory English sentence. The starting point for generating this explanatory response is the set of proof clauses generated by the theorem prover to answer the question. Response generation, in general, is seen as having four stages: filtration, to remove propositions known or obvious to the user from the response; organization, to provide a partial ordering of response propositions; assembly, to map clause form propositions into an appropriate logical form and lastly, verbalization, to translate logical form into English. A partial implementation of a response generator was written. In a first program, the assembly and verbalization stages were combined in a single stage that directly translated clause form into English. The second program attempted only the assembly stage, and translated clause form into logical form. Both programs were limited to work for clause sets presumed to be expressible in a simple English sentence.

Acknowledgements

I would like to thank Dr. Len Schubert, who was a continual source of inspiration, encouragement, and guidance - and all the members of my examining committee: Dr. Renee Elio, Dr. Jeff Pelletier, and Dr. Lisa Markworth-Stanford, for their attentive reading, thoughtful comments, and worthy criticism of this thesis. In addition, I would like to thank all my fellow graduate students: Sven Hurum, Larry Watanabe, Brian Bray, Bopsi Chandramouli, Gurminder Singh, Ajit Singh, Chung Hee Hwang, Fahiem Bacchus, Nicola Ferrier, Brett Hammerlindl, Samuel Pun and David Meechan, to name a few, for making my graduate student days such an enjoyable experience. I would also like to acknowledge the financial assistance of the Department of Computing Science, the Faculty of Graduate Studies, the National Science and Engineering Research Council of Canada, and Mum and Dad.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Considering the relative ease with which humans can communicate with one another, perhaps one should be surprised that it is still difficult for humans to communicate with computers. To get a computer to do something, you have to either press the right button, type the appropriate keyword, or develop a series of commands in the computer's own language that will instruct it how to do what you want it to do. Wouldn't it be nicer if you could simply tell it what you want it to do?

It is difficult, however, for computers to communicate in natural language (like English, French, or Chinese). To say a computer has understood an English communication, means at the least, it has translated that communication into its own internal meaning representation, has stored that meaning in relation to existing facts about the world and can, if necessary, perform inference using that meaning representation to deduce further facts about the world. The difficulty with understanding natural language is that, depending on the context, an English sentence may have several different meanings. In a classic example, the English sentence "Time flies like an arrow" may have several different interpretations depending on whether the word "time" is categorized as a noun, verb or adjective and whether "like" is categorized as a preposition or verb. Conversely, the difficulty in generating natural language is that a particular meaning may have several different English translations, again depending on the context. Beyond a dependence on context, the real difficulty in automating the communication process is the lack of adequate, complete theories of syntax, semantics and pragmatics. Both semantics and pragmatics involve context dependence, and pragmatics involves inference, planning and "user modelling" (including the goals of the conversants); all are difficult problems.

Natural language processing is concerned with both understanding and generation. Natural language understanding involves parsing and translating English (or some other natural language) into a meaning representation. Natural language generation involves translating a meaning representation into an English response. Although response generation may be concerned with non-verbal replies (such as generating an action or producing a pictorial display), the scope of this thesis is narrowed to consider verbal replies only. Response generation in this thesis is thus taken to be a problem of natural language generation.

The purpose of this research has been to provide a rudimentary response generator for an existing natural language understanding system which has been under development at the University of Alberta for some years [Covington&Schubert 80, DeHaan(in prep.), Schubert 76, Schubert, Goebel&Cercone 79]. This system is intended to allow storage and effective use of arbitrarily large amounts of knowledge about an unlimited variety of subjects, including knowledge about the "everyday world" and narratives. A semantic network is used to organize and store propositions in clause form. The clauses (or propositions, in semantic net terminology) are attached to the "concepts" they reference, and at each concept, propositions are organized in conformity with a topic hierarchy. So, for example, knowledge about "Mary" or "elephant" is attached to corresponding concept nodes, and the part of this knowledge concerned with, say, appearance can be selectively accessed by descent to the appropriate topic node in the topical data structure ("topic access skeleton") attached to the concept nodes. In addition to the main net which expresses the system's beliefs about the world, subnets are used to represent different views of the world expressed by such modal predications as ("John believes that ...") and ("Little Red Riding Hood is a story in which ..."). Each subnet has a concept access skeleton allowing individuals to be accessed associatively via properties known about them. Eventually intended as a general intelligent conversation system, the system currently can answer simple yes-no questions about miscellaneous stored facts, including simplified "stories" (but as yet without regard for temporal order, causal connections, the goals characters have, or other relationships not expressible in first order nonmodal predicate calculus). A user inputs a question that is parsed and translated into a logical form. The system translates the logical form question into clause form and then answers the question by generating a proof or disproof using a resolution theorem prover. For instance, answering a yes-no question involves proving the truth or falsity of the proposition implicit in the question. The goal of the response generator is to to supplement the initial YES or NO response produced by the theorem prover with an explanatory English sentence. The clauses used in the proof or disproof provide the general information content of the answer. They are refined into a clause form answer by eliminating propositions known to the user (not implemented). The remaining clauses are assembled into a logical form answer and the logical form answer is translated into English using a grammar (implemented herein as single stage that directly translates clause form into English).

The use of separate logical form and clause form levels of representation seems appropriate for several reasons. First, a logical form that is "close to surface form" arises naturally from a Montague/Gazdar approach to language [Gazdar,Klein,Pullum&Sag 85,Schubert&Pelletier 82]. This approach uses an English grammar made up of phrase structure rules, each rule having a syntactic component that specifies a context-free phrase structure rule, along with a semantic component that specifies a logical translation of that particular syntactic component. The Montague/Gazdar approach was chosen because it combines a relatively simple, parsable syntax with well defined semantics. Second, a "deeper" clausal representation facilitates topical organization of information, and simplifies inference [Covington&Schubert 80]. Clausal form facilitates topical organization because it minimizes the "chunk size" of knowledge, putting logically unrelated facts into separate chunks, and hence allowing them to be stored under different topics. For example, It's not true that either Mary isn't single, or she lives at the YWCA

Logical Form: (NOT ((NOT (Mary SINGLE)) V (Mary LIVES-AT YWCA)))

Clause Form: (Mary SINGLE) (Mary LIVES-AT YWCA)

Topics might be: Social relationships location

Logical form may contain many embedded implications and conjunctions. A flatter representation, obtained by splitting logical form into clause form propositions, offers computational advantages. In particular, since different logical forms may reduce to the same clause form, retrieval and deduction are made easier; also, matching is easier and duplicate propositions with different logical forms are not stored repeatedly. Third, logical form is highly ambiguous (with respect to referents of noun phrase translations, quantifier scopes, and the "intended meanings" of predicates and operators) and an unambiguous representation such as clause form is desired for the knowledge base.

Response generation can be divided into of deciding "what to say" (i.e., selecting particular items of information to be communicated) and deciding "how to say it" (i.e., translating these items of information into English). As mentioned before, even though responses may sometimes be non-verbal, such as just storing information, doing a physical task, or displaying a picture, this thesis is concerned with generating verbal replies.

1.1. Deciding "what to say"

Before the natural language generation process can begin, the general information content of the response must be established. The general information content is a set of unordered propositions that contains the meaning to be conveyed by the response. This set of propositions is obtained by inference and retrieval of propositions from the knowledge base. In the general conversation system mentioned previously, the set of clauses used by the resolution theorem prover to generate a proof contain the required information in propositional form. Relevant clauses are selected by "tracing back" from the empty clause, and selecting only those clauses involved in deriving the empty clause and discarding all the rest.

In a sophisticated question-answering system (or more general conversation system), plans and goals of the speaker and hearer must be taken into account. Knowledge about plans and goals is also required to understand actions of characters in stories. A response generator must be able to reason about its own plans and goals, those of the hearer, and those of characters in stories.

Several stages of natural language processing affect the decision of "what to say". A correct understanding of the intent of the question is crucial to generating the appropriate response. This is assumed to be provided by the natural language understanding component, and therefore the response generator makes no attempt to further re-interpret the question.

1.2. Deciding "how to say it"

This problem is sub-divided into four separate problems. These problems are assumed accomplished by a separate processing stage.

1.2.1. Filtration

Information already known to the user must be filtered from the propositional form to avoid producing verbose and redundant responses. This requires a user model. The user model has two types of information: user goals and user knowledge. Both types of information are required in the initial interpretation of the question.

1.2.2. Organization

Once the user model has been used to refine the set of propositions to contain only information-relevant to the user, the next problem is to organize the propositional form so as to make it sound natural when eventually translated to English. If the response is to consist of multiple sentences then some ordering must be imposed on the propositional form so information flows in a continuous manner from sentence to sentence. If the response is to consist of a single sentence ordering information is still required. Single sentences must be provided with a subject and decisions about pronominalization and ellipses must be made. A discourse model containing a record of the discourse context, the goals of the discourse, and structural knowledge about how to organize discourse would be used to solve these problems. The structural knowledge would have two levels, one for single sentences (e.g. size adjectives should precede colour adjectives) and one for multiple sentences (e.g. the first sentence of a paragraph should express the topic of that paragraph). The general conversation system has no equivalent stage in its design, but a system wishing to produce natural sounding English responses should have this component.

1.2.3. Assembly

While knowledge representation plays a role in all stages of natural language processing, the chosen knowledge representation is solely responsible for the ease or difficulty of doing this stage. The problem is to translate the propositional form into a compact representation which may then be easily translated by a grammar. This stage is necessary because the propositional form is sufficiently far away from surface English that it may have many different English renderings. The conversion to logical form reduces the number of possible renderings. In the general conversation system, this stage corresponds to the problem of translating the set of response clauses into logical form.

1.2.4. Verbalization

The last stage of the generation process is to generate English. By now, the relevant knowledge has been tailored to suit the user, fitted for a particular discourse style, and trimmed into a form ready for translation by a grammar; all that remains is to produce English. In this stage, the logical form and an English grammar interact to produce English output. A grammar like Generalized Phrase Structure Grammar would be ideal for this problem as it provides an explicit representation of the relation between logical form and English syntax.

To facilitate comparison between other systems, what is considered the propositional form and logical form of each system is presented during the discussion of that system. Propositional form and logical form will be defined for this purpose as follows. Propositional form, the deepest level of representation, is more geared towards inference than towards generation of English. An example is the clause form used in a semantic net of the type sketched above.

Clause Form: (X LOVES John) V (NOT(X GIRL))

Logical Form: ((EVERY GIRL) (PRES (LOVES John))

English: Every girl loves John

Logical form is a representation close to surface English, to which a grammar can be easily be applied to generate actual English. Unlike propositional form, logical form may already contain various ambiguities (of reference, scope, and intended interpretation of predicates and other symbols). In general, the generation process can be described as as a series of transformations, each producing a representation closer to English with the final output being surface English.

The rest of this thesis is organized as follows: chapter two is a literature survey, chapter three, a description of the English generation system implemented, chapter four, a design for a comprehensive response generation system and some suggestions for future research, and chapter five, some conclusions.

Chapter 2

A Survey of the Literature

Each system will be discussed completely, covering both the "what to say" and "how to say it" headings and what is to be considered the propositional form and logical form of that system. Although some systems don't contribute much to one topic or the other, organizing the papers this way emphasizes the contribution each paper makes to the theory of response generation, and provides a common framework from which to critique these systems.

I will begin with a question answering system written by Wendy Lehnert[Lehnert 77]. This system decides "what to say", that is, it provides the propositional form of an answer. Another system written by Goldman[Goldman 75] is used to produce the final English answer or decides "how to say it". It translates propositional form into logical form, and then logical form into English. The latter is accomplished using a method similar to one initially described by Simmons and Slocum[Simmons&Slocum 72]. These systems work together to produce an English answer, and thus will be discussed together. Their propositional form corresponds to conceptual dependency diagrams and their logical form, a network of concepts.

2.1. Lehnert

Lehnert concentrates on the process of question answering. In this particular case, the natural language understanding component will be described as well as the generation component, since the two are so interdependent. SAM and PAM are the story understanding systems with which this system was designed to work. After an English story that had been parsed and translated into a conceptual dependency diagrams, Lehnert's system, QUALM, was used to answer questions (that have also been parsed and translated into conceptual dependency diagrams) about a story. Because all the inferencing is done by SAM[Schank 75] and PAM[Wilensky 78], Lehnert's system uses no inference to produce an answer. All possible answers to questions are directly taken from the original story or built by script or plan instantiation. For instance, if SAM knows that "John went to a restaurant" then it also knows that "John was hungry", "John read a menu", "John ordered food from a waitress", etc., without being explicitly told this in the story. This type of forward inferencing provides answers to questions before they are asked, thus all Lehnert's system has to do is retrieve an already existing answer proposition from memory.

To find the correct answer in memory, two stages are needed. The first stage involves understanding the question and the second stage involves finding an answer to the question (i.e. deciding "what to say").

The question is first categorized as one of 13 possible types (see table 2.1) to understand the question. Conceptual dependency diagrams determine the question category. Next, an interpretive analysis is performed in which a series of rules, some of them script specific, test to see if the category should be reassigned. This is so questions whose syntax and semantics dictate alternate answers are appropriately handled. For example, the syntax of the VERIFICATION question "Do you have the time?" demands a "yes or "no" answer. Reclassified as a REQUEST, the semantically appropriate answer (the actual time) can be generated. The reclassification is done by recognizing that the object in question (the time) is of little value, and that the person answering the question possesses that object at this moment. This same rule would not work however if the question was "Can you see the clock?" because the clock cannot be assumed to be of little value. Thus a "yes" or "no" answer would be generated when a more appropriate answer would have been the actual time.

The second stage, determines the information content of the answer and then translates the formal representation of this information into English (see figure 2.1). The information content of the answer is determined by looking at the question

Category	Description	Example
Causal Antecedent	What chain of events led to concept in question?	Why did John go to New York?
Goal Orientation	What was the mental state of person in question concept?	For what purpose did John go to New York?
Enablement	What enabled the concept in question?	How was John able to eat?
Causal Consequent	What did the question concept cause?	What happened when John left?
Verification	The truth of something is in question.	Did John leave?
Disjunctive	Verification with an OR	Did John or Mary leave?
Instrumental/Procedural	The question concept involves unknown instrumentality or procedure.	How do I get to John's house?
Concept Completion	A missing concept must be supplied.	What did John eat?
Expectational	The question concept involves something that was supposed to happen, but didn't.	Why didn't John eat?
Judgemental	Judgement is solicited from answerer.	What should John do next?
Quantification	The answer requires an amount.	How much did John spend?
Feature Specification	A property of something is in question	What colour are John's eyes?
Request	Any question not categorized as one of the above.	Can you give me a ride?

Table 2.1 Lehnert's Thirteen Question Categories

category, the trace of any category reclassifications, the initial answer (e.g YES or NO) and elaboration options that determine how talkative the system will be. Following this, a memory search is conducted. Answers are found in three levels of story representation: causal chains, script structures and planning structures. The system finds an answer by searching until a concept matching the question concept is found. The question category along with script and plan specific heuristics, determine how memory is searched to find the answer. The formal representation of the answer is then translated into English. Another program[Goldman 75] to be discussed later, accomplishes this translation.

It could be argued that the categories Lehnert presents are rather arbitrary. Most of her categories are just subcategories of wh-questions, depending on whether the "missing concept" is a cause, purpose, enabling event or state, result, physical object, Category Selection and Interpretive Analysis

Question Category: VERIFICATION

Logical Representation: ?Ex[x menu]&[W gave John x]

English Translation: Did the waitress give John a menu?

The original category is not reclassified.

Content Specification

Correction/Elaboration Option

```
IF   The system's "mood" is talkative &
     The initial answer is No &
     The question category is VERIFICATION
THEN
     Make up a concept completion question
     Answer that concept completion question
```

The Correction/Elaboration Option will only be executed after the memory search stage.

Memory Search (retrieval heuristics)

Memory is searched for a concept which matches the question concept.

If the concept is not in memory and the initial answer is NO, then the instructions of the Correction/Elaboration option are followed and the concept completion question

?Ex[x gave John menu]

is generated and answered.

So, assuming memory contains [H gave John menu], then the final answer will be:

No, the hostess gave John a menu

Note: Lehnert uses conceptual dependency diagrams, rather than the logical form that I use in this example.

Figure 2.1 An example of how Lehnert's system generates an answer

reason, action, number or property. Other types of wh-questions are not covered. For example, the "missing object" might be a belief, hope, fear, activity, decision, manner. So the questions "Does John believe the waitress will take his order?", "Does John hope the waitress will take his order?", "Is John afraid the waitress will take his order?", "Does John always order?","Does John decide what Mary will order?", and "Does John showoff when he orders?" cannot be answered. Since the entire system relies on these categories, and since different processes are associated with each category (the inferential analysis done differs with category and the specific search strategies change for each category) one could argue the system is arbitrary in nature and as a consequence would be difficult to generalize to the types of wh-questions which are not covered by the thirteen question categories.

In a story understanding system using a theorem prover to answer questions, the body of the proof would reflect the relevant assertions of any script or plan used to answer the question. That is, to prove or disprove a proposition, memory is searched by the theorem prover to find relevant propositions and these are combined using inference rules to obtain a proof or disproof. Since the memory search occurs in the theorem prover, the search is narrowed to those propositions which may be resolved against set of support propositions. The semantic network organization further aids the search because related propositions are physically close together in the semantic network. Starting from the clauses which represent the question concept, the proof is built by selecting those propositions of related topic or associatively retrieving those propositions with the similar properties. No specific search strategies are required and the proof generated by the theorem prover may be expected to contain the answer.

In Lehnert's system, inference occurs before the question answering process begins (i.e. when the English story is initially input). To find an answer no inference is required, just a systematic search of memory to find a concept matching the question concept. Each level of memory (scripts, plans and causal chains) requires an associated set of retrieval heuristics organized according to question category. If the answer cannot be found, then further strategies dependent on question category are used to determine the answer. Many specific search strategies are required because no inference occurs at the time of question answering. Instead of memory search being guided by the inference process, it is guided by the type of question it is trying to answer before that question is even asked. A system where answers to possible questions are automatically produced upon reading a story arbitrarily limits the types of questions which can be answered to those whose answers already exist in memory. This is reflected by the restriction of only allowing thirteen types of questions when in fact other types of questions exist.

Propositions produced by the forward inferencing may never be used to answer a question. That is, forward inferencing may produce more inferences than necessary and thus, add more propositions to memory than are needed. It would make more sense to generate propositions as they are needed to answer certain questions.

The propositional form used by Lehnert is Schank's conceptual dependency formalism. The expressive power of this representation is limited. For instance, it is difficult to express concepts such as "likes", "wants", "hopes", and "believes" in the 11 or 12 primitive acts provided for by Schank. Questions about these concepts are simply not attempted by this system.

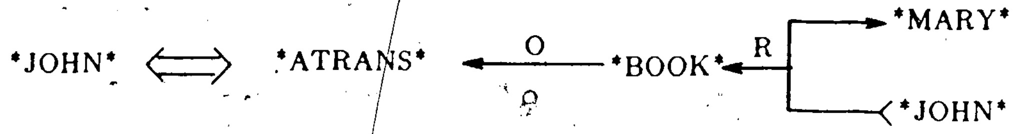
Lehnert's system attempts to find an answer to a propositional form question by searching memory for an existing propositional form answer. The complicated nature of the stage that performs the memory search is the result of the lack of a general mechanism for deductive inference. The decision of "what to say" is embodied in the conceptual dependency diagrams chosen for the answer. The decision of "how to say it", is not attempted by Lehnert; the system lacks both a user model and a discourse model (although taking the "mood" of the system into account does provide a rudimentary discourse guide). The actual generation of English is left to another system to be described next.

2.2. Goldman

This system decides "how to say" a conceptual dependency diagram. Starting from a conceptual dependency representation the system BABEL[Goldman 75] can generate a network of concepts that is eventually translated into English. Two phases, a knowledge compaction phase and a grammar phase, accomplish this translation. The knowledge compaction phase generates a network of concepts (a logical form equivalent) starting from a conceptual dependency diagram (a propositional form equivalent). Goldman's system assembles a logical form answer that can then be verbalized using a grammar into English.

To start, a discrimination net is applied to a conceptual dependency diagram, yielding a verb. A discrimination net can be thought of as a decision tree that selects the appropriate word to best convey the meaning represented by the conceptual dependency diagram (see figure 2.2). At each internal node in the decision tree, a test is performed that determines the next path to take. Once a leaf node has been reached and a verb selected, an associated lexical entry and framework is retrieved. This framework consists of several frames. Each frame has three types of information: relation information, field specification information, and special requirement information. All of these are used to build a network of concepts, called a syntax net, from which English is then generated (see figure 2.3). The special requirement information is used to introduce prepositions into the network.

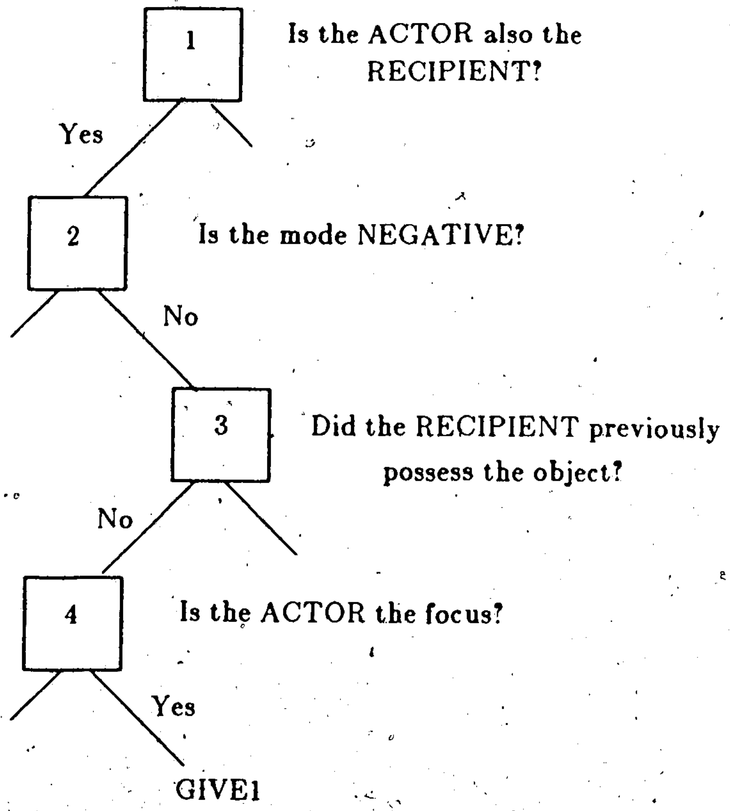
The algorithm used to build the syntax net proceeds as follows. To start, a node is created called the active node. A framework, retrieved by application of a discrimination net to the conceptual dependency diagram, is used to add new nodes to the active node. As each frame in the framework is processed, a new node is added to the



Conceptual dependency representation of

"John gave the book to Mary"

Discrimination net:


1
Is the ACTOR also the RECIPIENT?
Yes
2
Is the mode NEGATIVE?
No
3
Did the RECIPIENT previously possess the object?
No
4
Is the ACTOR the focus?
Yes
GIVE1

This discrimination net yields the concexicon entry GIVE1 when applied to the above conceptual dependency diagram.

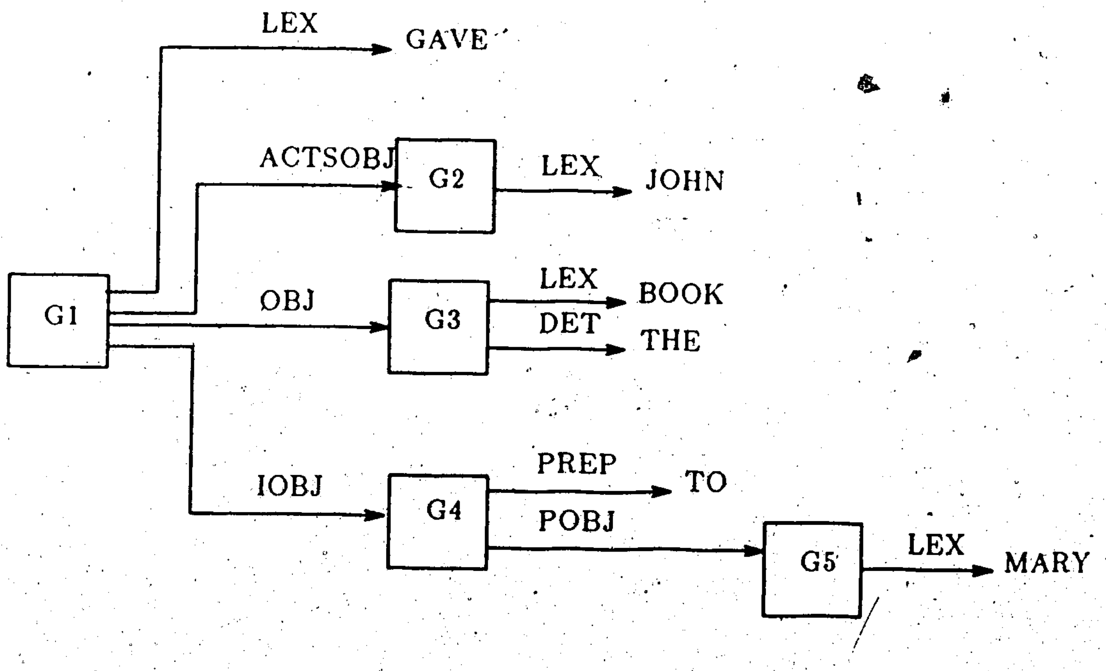
Figure 2.2 A conceptual dependency diagram and discrimination net

network. This node is joined to the active node by the relation specified in the frame. Once joined, the new node becomes the active node, and the conceptual dependency diagram is reduced to the field specified in the current frame. The old conceptual dependency diagram and active node are saved on a stack. The discrimination nets are once again applied to this subset of the original conceptual dependency diagram to

Concexicon Entry for GIVE1

	Syntax Relation	Field Specification	Special Requirements
GIVE1	ACTSOBJ	ACTOR	
	OBJ	OBJECT	
	IOBJ	RECIPIENT	(PREP TO)

This entry is used to construct the syntax net below:



Which, when translated by a case grammar, becomes:

"John gave the book to Mary"

Figure 2.3 A concexicon entry and a sample syntax net

return a new verb and the process starts again with a new framework. When no more frames are left at the current level the stack is popped, and the old active node retrieved. The process stops when no more frames are left and the stack is empty. Once the network is built, English generation proceeds using an Augmented Transition Network grammar.

The structure of the network to be built is contained explicitly in the framework associated with each verb. Therefore, the building blocks for the network are supplied and the task, in generating an answer network, is to hang concepts in the right place. Discrimination nets decide which frameworks should be chosen for the network. These nets are language specific and contain about seven to ten verbs each. All the inference required to assemble propositional form into logical form is pre-determined and defined in these nets. To make the system less deterministic, more than one discrimination net may be applicable at any point in time, a leaf node may have more than one verb associated with it, or it may have a pointer back to another node in the discrimination net.

Some of the tests in the discrimination nets require inference and memory search and this may be the price paid for the use of the conceptual dependency representation as a propositional form. This is because a conceptual dependency diagram decomposes the content of English assertions into numerous "conceptualizations" based on just a few (11 or 12) primitive acts. Thus, more work is required to generate a form suitable for translation by a grammar than would be required for a higher-level propositional representation, whose predicates have direct English translations.

This is not to say some inference is not still not required given a higher level representation. Determining whether "stationwagon" (a certain predicate) can be rendered in the logical form as "automobile" (a more general predicate) requires inference: first, to determine that the latter indeed "covers" the former, and second, to determine that the latter is sufficiently specific to convey the information required by the hearer given the discourse context. The first point raises the necessity of having a type hierarchy in the knowledge base to be able to automatically infer relationships between different types, the second, of having a record of the discourse context. Goldman's system has neither.

Goldman's system, while accomplishing the necessary end, that of producing a form ready for translation by a grammar, solved a problem specific to conceptual dependency representation. That is, the re-interpretation of the original meaning of a conceptual dependency diagram lost in the translation to a deep meaning representation. Goldman's system used specific knowledge to obtain a compacted representation suitable as logical form. It remains to be seen if there is a method for accomplishing this knowledge compaction without using specific rules for translating specific meanings of propositional form. That is, instead of having pre-defined choices for the assembly of propositional form into logical form, can this assembly occur in a systematic manner independent of the meaning of the propositional form. In the next phase, the final verbalization into English, a grammar is used. Goldman used a method first described by Simmons and Slocum, and thus it will be described next under that heading.

2.3. Simmons and Slocum

In an early work on English generation by Simmons and Slocum[Simmons&Slocum 72], a way of translating a network of concepts (a logical form equivalent) into English using a grammar is presented. Goldman's system, discussed previously, used similar method to generate his final translation into English.

Simmons and Slocum describe a method of generating English from semantic networks. Meaning is represented as a group of concepts interconnected by relations. These relations represent case relations such as AGENT, OBJECT and LOCATION. The concepts represent word senses (see figure 2.4).

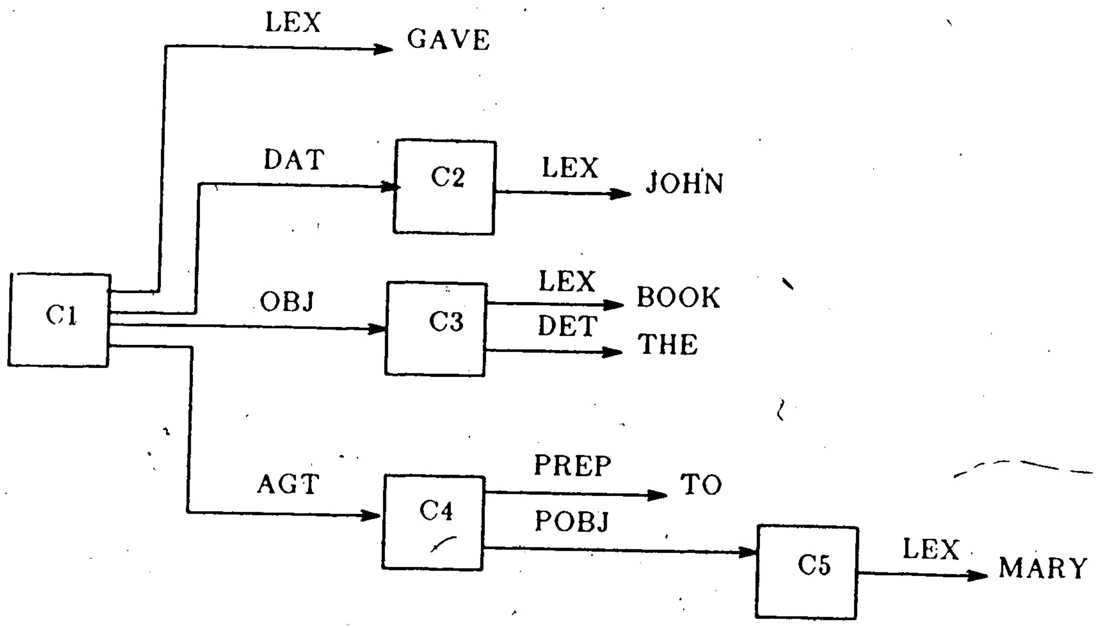
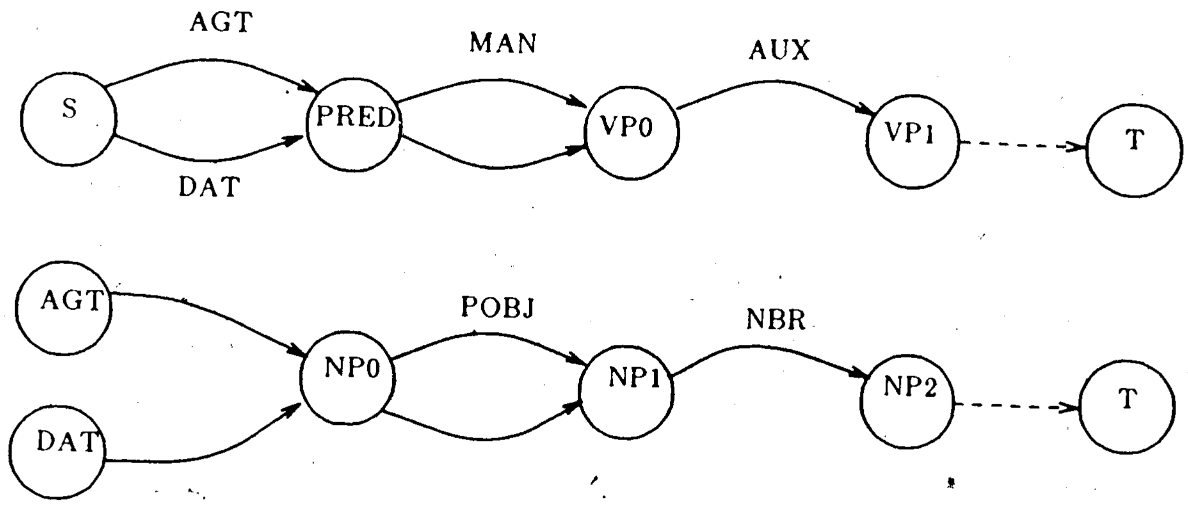


Figure 2.4 Concept network

Syntactic information used to generate English is in the form of an Augmented Transition Network (ATN) grammar (see figure 2.5). The relation between syntax and semantics is represented as an arc in the ATN. Each arc is named to correspond to a semantic case relation, and associated with each arc is a function that relabels the original network of concepts with labels corresponding to syntactic categories. This function also performs appropriate transformations on subject, object and verb string registers.

The syntactic and semantic components of this system are interleaved: the arcs in the grammar represent semantic information and the nodes, syntactic information. By making it difficult to separate the two kinds of information, the task of writing a grammar becomes more complicated. Current linguistic theory favours a clean separation of the two kinds of information. Also, the transformations required to go from meaning representation to surface level English must be clearly specified and easy to generalize, which is not the case for the transformations used by Simmons and Slocum.



Unlabelled arcs denote unconditional transfers.
Arcs correspond to semantic case relations.
T denotes a terminal state.

English is generated by selecting a path through the grammar, guided by the semantic case relations of the concept network.

Figure 2.5 Augmented Transition Network

In summary, for the previous systems, (Lehnert's, Goldman's, and Simmons and Slocum's), the propositional form is a conceptual dependency diagram and the logical form is a network of concepts. Lehnert's system decided "what to say" using question categories, and specific search strategies to find an existing answer concept in memory. Deciding "how to say it", was accomplished by Goldman. The logical form was assembled from propositional form using discrimination nets and an ATN grammar, originally described by Simmons and Slocum, was used to produce the English verbalization.

2.4. Dyer

A more recent question answering system based on Schank's conceptual dependency formalism is BORIS[Dyer 83, Lehnert 83]. This system is similar to the previously mentioned system because it uses some forward inferencing to generate answers, and a prototype hierarchy, which can be thought of as a hierarchy of discrimination nets, to assemble conceptual dependency diagrams into a logical form. A procedural grammar is used to directly translate this logical form in left to right order into English. Work previously done in several separate stages (parsing into conceptual dependency representation, forward inferencing, memory search) and several different systems has been integrated into a single stage.

The main difference between BORIS and QUALM is the types of questions they can answer. This is a direct result of the addition of different knowledge structures to the system. In total seventeen different knowledge structures are used in BORIS. In addition to scripts and plans, MOPs (Memory Organization Packets) META-MOPs and TAUs (Thematic Affect Units) are present. This replacement occurred because scripts and plans allowed inferences about story events to occur, but not inferences about goals and intentions. MOPs incorporate knowledge about events, plans and goals. For instance, a BORROW-MOP includes the outline of a plan to borrow something. META-MOPs incorporate a higher level of plan abstraction such as what is involved in a favour between friends, professional services, personal communications and meetings. TAUs incorporate knowledge commonly expressed in adages such as "A friend in need is a friend indeed" and "Every cloud has a silver lining". TAUs are useful in explaining the emotional affect plot events have on the characters in the story. MOPs also include links to other MOPs, META-MOPs or TAUs, (e.g. the BORROW-MOP may have a link activated by a context of friendship to the FAVOUR-MOP). This allows BORIS to answer questions like "Why did John owe Bill a favour?" and "How did Paul feel?" that could not be answered by QUALM.

Memory is searched at the time of question understanding and by the time a question is understood, the answer may have already been found. This is because to understand a question, BORIS finds a referent to the question concept in episodic memory. (Semantic memory consists of the knowledge the system has before it is told the story, episodic memory, the knowledge added after). This referent may be the answer. Any search of memory used to understand a question was not available to QUALM where memory was searched again to find the answer. This shows that the question understanding can be used to guide the search for an answer for some types of questions.

Not all the inferencing in BORIS occurs at story understanding time. Some inferencing (MOP instantiation) occurs at question understanding time. When looking for an answer, events can be reconstructed from partial instantiations of MOPs. That is, the answer concept may not be explicitly in memory but may be inferred. This differs from QUALM which relies on SAM and PAM to do all the inferencing and simply searches memory for an existing answer concept. The change from specific search strategies based on question types to deductive inference at question answering time, shows the usefulness of inference occuring at question answering time rather than only at story understanding time. Inference can guide the search for an answer in memory.

Although BORIS can answer a wide range of questions about a divorce story, it is limited to only divorce stories. Perhaps this shows that it is not an easy task to build new MOPs and to generalize the system for other types of stories. For example, in newspaper articles, three levels of summarization appear (the headline, a first paragraph summary, followed by a more detailed description of events). This differs from the way fables are told. Thus, different schemas for organizing the knowledge are required, and hence, to answer questions.

BORIS has no user model or discourse model. The translation of propositional form into English is accomplished directly using the prototype hierarchy. A logical form is not used. A phrase stream whose elements are expanded in left to right order until they become words, is used to generate English. Elements in the phrase stream are expanded according to information in the prototype hierarchy. The grammar is represented as procedural expansion information dispersed throughout the prototype hierarchy. This hierarchy contains both the assembly and verbalization information required to generate English from propositional form. This would seem to be a step backwards from BABEL, in which the logical form was clearly specified and a separate stage was used for the assembly and verbalization problems respectively.

BORIS shows inferencing about plans, themes and goals is required for a general question answering system. Next, a different system written by Cohen is described; this system uses planning to decide "what to say". The main difference between this system and BORIS is that BORIS uses planning to answer questions about characters in stories, whereas Cohen's system is using planning to carry on a mixed initiative dialogue with the hearer.

2.5. Cohen

Planning natural language allows the intentions underlying speech acts to be formally modelled[Cohen 78]. That is, the decision of "what to say" is planned. Speech acts are thought of as operators in a planning system whose actions affect the belief models of the speaker and hearer. The preconditions of these speech acts consist of WANT preconditions (the agent of an action has to want to do that action) and CANDO preconditions (propositions that must be true before the operator can be applied). The following speech acts are defined: requests, and informs (answers).

To be able to plan an utterance, the goals (WANTS) of the speaker and hearer must be clearly defined. Within a limited domain, such as a system providing help to people wanting to board or meet trains[Allen 80] (see figure 2.6)

S - System plays the role of information clerk.
A - patron of train station.

A has two goals, to either BOARD or MEET trains:

```
BOARD(agent, train, station):  SOURCE(train,station)
    precondition:  AT(agent,     the x: DEPART.LOC(train,x),
                                 the x: DEPART.TIME(train,x))
    effect:        ONBOARD(agent, train)
```

Speech Acts:

```
INFORMIF(speaker,hearer,P)
    precondition:  speaker KNOWIF P
```

```
REQUEST(speaker, hearer, action)
    effect:    hearer WANT (hearer DO action)
```

Sample Question: **Does the Windsor train leave at 4?**

translates into: REQUEST(**A**,**S**, INFORMIF, (**S**,**A**, LEAVE, train1,1600))
where train1 = the (x:train):PROPERTY of x is WINDSOR.

LEAVE matches the DEPART.TIME and DEPART.LOC preconditions
in the BOARD plan.

GOAL: A KNOWIF LEAVE(train, 1600) is subdivided into 2 goals
(assuming that it is known that **A** knows DEPART.LOC)

GOAL1: **A** KNOWIF DEPART.TIME(train1,1600)
GOAL2: **A** KNOWREF the (x: time): DEPART.TIME(train1,x)

If answer is "YES", then both goals are established; If answer is "NO", then the second goal accounts for the extra information in NO answer.

Answer: **No, the train to Windsor leaves at 5.**

Figure 2.6 Allen's system: using planning to answer questions

specific goals are attributable to the people asking questions and these goals are clearly defined. Thus planning is used effectively to determine the appropriate response.

With a story understanding system, the system has the goal of showing it knows an answer to a question and the questioner has the goal of testing its knowledge. These goals are too general to provide much guidance to the mechanism actually generating an answer. Also, since the questioner of the story understanding system will be asking a question resulting in the system responding with an inform, there is no need for this system as yet to plan other types of speech acts. If the system were to engage in a purposeful dialogue with the user, a planning mechanism would be necessary.

In short, the natural language planning approach solves a much wider range of problems than merely generating natural language. It is a way of understanding sentence fragments and indirect speech acts by anticipating goals. The difficulty with this approach, with regards to response generation, is this: in a context where goals are not clearly defined, planning cannot be of use. Thus, in the absence of specific goals, a different method for determining answer content is needed.

So up till now the systems discussed have provided mechanisms for deciding "what to say". These systems dicussed supply some insight into what type of knowledge must be supplied to effectively answer questions: scripts, plans, goals, and intentions. The generality and extendibility of the conceptual dependency based systems is questioned because of their lack of the capability to perform general deductive inference and the inadequacy of the conceptual dependency formalism for representing certain kinds of knowledge. Cohen's system, on the other hand, poses the problem of how best to answer questions in the absence of user goals, when planning is of no use.

All of the previous systems concerned themselves to the problems of generating single sentence responses. The next few systems tackle the problem of multiple sentence responses. The following system concentrates on the problem of "how to say it".

2.6. Mann and Moore

Mann and Moore have designed a system that generates natural language in several stages[Mann&Moore 81]; "what to say" is presumed given and consists of a body of relevant knowledge. This system determines "how to say" this relevant knowledge.

Information to be communicated is decomposed into propositions and then recomposed in several separate stages to form an English translation. The system is designed to work independently of the knowledge base it has to translate. Theoretically, it could translate any form of knowledge representation into English, for instance, whether knowledge is represented as a semantic net, a set of propositions, schemas, or conceptual dependency diagrams, should make no difference to this system. This independence is gained by a fragmenter that translates any deep knowledge representation into a set of propositions. The set of propositions (a deep representation themselves), contains the equivalent information in the original representation yet provides a common starting point for the English generation process. This assumes all types of deep knowledge representation can be represented by an equivalent set of propositions, and problems specific to each type of knowledge representation can be dealt with by the fragmenter without involving later stages of generation. Writing a fragmenter that can translate all forms of knowledge is definitely a non-trivial problem, and Mann and Moore's system could successfully translate only one type of knowledge: a semantic network containing information about contingency plans. However, if one is to formalize the English generation process, it is useful to separate translation problems specific to one kind of propositional form, from those that in general will arise for any propositional form.

To start, assume a body of given relevant knowledge has been fragmented into propositions. In the example used thoughout this discussion, the relevant knowledge given to translate is a contingency plans database represented as a semantic net. Mann and Moore do not describe the method used to fragment this semantic network into propositions, but since the semantic net is represented as a series of LISP expressions, one can assume that this semantic net is already fairly close to a set of propositions. These propositions are equivalent to clause form propositions in the story understanding system, the so called propositional form.

The amount of information contained in a proposition is less than or equal to that which can be conveyed in an ordinary English sentence. For example, the proposition

```
(WHEN   (CALLS X WELLS-FARGO)
                (CALLS WELLS-FARGO FIRE-DEPT))
```

translates into,

> When someone calls Wells-Fargo,
> Wells-Fargo calls the Fire department.

The fragmentation of the knowledge base allows sentences to be formed that could not be generated directly from the knowledge base. More specifically, it allows items that are not necessarily close together in the knowledge base to be included in the same sentence, it avoids the problem of how to carve the knowledge base into sentence size chunks, and it removes the necessity of determining where a sentence should start and end in the knowledge base. Even though a different fragmentation procedure would be required for different knowledge bases, this module allows the rest of the natural language generation process to proceed in a uniform manner regardless of the structure of the knowledge base.

The unordered propositions, along with an expressive goal, are passed to a problem solving stage. In general this stage attempts to impose an ordering on the propositional form. This stage attempts to formulate the fragmented propositions according to a particular expressive style and is designed to work in the traditional sense of an AI problem solver. An expressive goal (e.g. TELL, INSTRUCTIONAL-NARRATE) is associated with rules that contain structural knowledge about how to organize sentences and paragraphs, and what sort of things to make explicit in the generated text. The expressive goal is predetermined and is defined by the intended effect the generated text is to produce on its reader. Using rules organized by expressive goals, this stage selects those rules which relate to any given expressive goals and performs actions indicated in these rules to attain the desired goal. For example, given the goals TELL and INSTRUCTIONAL-NARRATE (see figure 2.7):

Factoring Rules:
TELL
1. Place all (EXISTS ...) propositions in an upper section.
2. Place all propositions involving anyone's goals in an upper section.
3. Place all propositions involving the author's goals in an upper section.

INSTRUCTIONAL-NARRATE
1. Place all propositions with non-reader actor in an upper section.
2. Place all time dependent propositions in a lower section.

Ordering Rules:
INSTRUCTIONAL-NARRATE
1. Order time-dependent propositions according to the (NEXT ...) propositions.

Advice Giving Rules:
INSTRUCTIONAL-NARRATE
1. YOU is a good thing to make explicit in the text.

Figure 2.7 Problem solving rules

the system factors the propositions into two lists (an upper section and a lower section that will eventually form two paragraphs separated by a paragraph break), orders all time dependent propositions and attaches "advice" to be used at a later stage. Some sample advice would be to make "YOU" explicit in the text. For instance, instead of saying "When SOMEONE hears the alarm bell ..." say "When YOU hear the alarm bell ...".

Mann and Moore's rules are fairly straightforward. For example, all time dependent propositions are ordered one after the other, and all propositions stating the existence of something are placed in the upper paragraph. The latter is based on the notion that the fact that something exists should be mentioned before it is talked about in some other manner. All simple, commonsense rules that supply the necessary structure to the discourse. Mann and Moore's system is limited, however, to producing two paragraphs.

Next, the factored and ordered propositions with attached advice are put through a knowledge filter. The filter removes any propositions that are known or obvious to the user. The user model, a collection of propositions that the user can be presumed to know, is the basis for the removal. This user model is simple and contains no knowledge about user goals. Furthermore, this knowledge is not required in this stage. That is, user goals are implicit in the choice of relevant knowledge and expressive goals. Overall, a complete model of the user, including both goals and knowledge is necessary, but user goals can be assumed to have been used earlier on by a planning mechanism to select the relevant knowledge, and user knowledge will be used later on to filter that relevant knowledge. Both user goals and user knowledge are required in a user model, but each at different stages of processing.

In the next stage, aggregation rules (see figure 2.8), combine clause form propositions into complex clauses wherever possible. These conjoined clauses serve as more suitable platforms for generating sentences and may be thought of as resembling logical form propositions. The combining procedure works as follows. First, all propositions are assigned an initial score by the preference rules. This score indicates there is no initial preference for any particular clause and provides a base score which preference rules can add to or subtract from. Next, the aggregation rules are applied in all possible ways to generate a new set of potential clauses and preference rules assign these clauses a numeric value.

Aggregation rules:

Whenever C then X	Whenever C then Y	-->	Whenever C then X and Y
Whenever X then Y	Whenever Y then Z	-->	Whenever X then Y and then Z
Whenever X then Y	Whenever Y then Z	-->	Whenever X then Z
There is a Y	<mention of Y> (Y is known unique)	-->	<mention of Y>

For example:

There is an alarm Whenever there is a fire then the alarm is set off (The alarm is known to be unique) --> Whenever there is a fire then the alarm is set off

If P then Q
If not P then R --> If P then Q otherwise R.

These were obtained from clause combination rules for English.

The "Whenever C then X" propositional forms are fragmented from the semantic net.

Preference Rules:

1. Every proposition is initially assigned a value of -1000.
2. Every proposition embedded in a composite clause decreases value by 10.
3. If there is ADVICE that a term is GOOD, each occurence of that term increases value by 100.
4. Each sequentially time-linked proposition after the first, increases value by 100.
5. Certain constructs get bonuses of 200. (If then else ; whenever X determines Y).
6. Any clause produced by multiple applications of the same aggregation rule gets a large negative value.

The last rule was necessary for empirical reasons. Results from multiple applications of a rule were deemed awkward and confusing.

Figure 2.8 Aggregation and Preference Rules

The clause with the highest score is selected, the propositions it subsumes are removed from the answer set, and the process continues with the remaining propositions in the answer set.

Using English clause-combining rules to generate a compact form, is a general and fairly simple way of combining knowledge into a form ready for translation by a grammar. It is applicable to the general conversation system described in this thesis because of the similarity between knowledge representations. The propositional forms of both systems are very similar, thus, the English clause combining rules should be as effective in both systems.

While the aggregation rules are based on certain standard rules for English, the numeric values of the preference rules provide a less sound mechanism on which to base a system. If new preference rules are to be added, the empirical method for determining its numerical effect would have to be repeated. This would seem to be largely a trial and error process, on which much effort could be spent each time a new rule is added. Some thought should be given to the possibility of removing the need for numeric values and using instead knowledge about linguistics and logic to select the best combination of propositions.

Finally, when no more improvements through the application of the preference and aggregation rules are possible, the logical form propositions are transferred to a syntactic component that generates English (see figure 2.9).

Whenever there is a fire, the alarm system is started, which sounds a bell and starts a timer. Ninety seconds after the timer starts, unless the alarm system is cancelled, the system calls Wells Fargo. When Wells fargo is called, they, in turn, call the Fire Department.

When you hear the alarm bell or smell smoke, stop whatever you are doing, determine whether or not there is a fire, and decide whether to permit the alarm system or to cancel it. When you determine whether there is a fire, if there is, permit the alarm system, otherwise cancel it. When you permit the alarm system, call the Fire Department if possible, then evacuate. When you cancel the alarm system, if it is more than 90 seconds since the timer started, the system will have called Wells Fargo already, otherwise continue what you were doing.

Figure 2.9 Sample output from Mann and Moore's generator

The grammar used to generate English was described only as context-free. The interesting part of their sentence generator was in a module which generated referring phrases. This would keep track of objects already referred to in the text, so that subsequent references to an object need only distinguish that object from other objects in the reader's attention. This referring phrase generator introduced terms, incomplete descriptions of objects, and pronouns into the text to make it more readable.

Mann and Moore conclude with the observation that deciding "what not to say" is as important as deciding "what to say" when generating natural language. Their knowledge filter stage, equivalent to the stage where various clauses are deleted from the proof in the general conversation system, removes all propositions known or obvious to the user. This knowledge filter stage is necessary to provide fluent, non-verbose, natural sounding English, and, as seen, Mann and Moore's system can generate superior English output. The above, however, is the most complex example that the system can generate, and the range of texts it can produce must be limited to those as similar as possible to contingency plan databases.

The organization of the four components leaves some room for discussion. Discourse rules organize propositions in the response set according to an expressive goal. "Don't Express" advice is attached to propositions already known to the user and these propositions are removed during the next stage, when propositions are combined to form sentence clauses. Mann and Moore do not explicitly delete these propositions right away in case they are "needed as transitional material or to otherwise make the text coherent". If the user knows certain propositions that have been deleted, one can assume that as he is reading the response, he will be supplying the transitional material to make the text coherent if necessary. Also, by fragmenting the knowledge, providing smaller chunks of knowledge that may form a sentence or may easily combine to form a sentence, one removes the necessity to keep extra information around for generating a sentence.

Perhaps a better organization would be to delete the propositions known to the user immediately from the set of relevant knowledge. That is, place the stage that organizes the propositions after the one that does the filtration. This may provide less work for the organization stage and the aggregation stage, by making the set of propositions to be worked on considerably smaller. This should result in increased system efficiency without a loss of coherency, since determining whether a group of sentences is coherent is dependent on the context and the user. It is just not clear why the extra information known to the user must be kept around, especially since there is a special module for generating noun phrase descriptions (the referring phrase generator). Maybe in formulating descriptions (ultimately, noun phrases) we do refer to things the other person already knows, but in formulating the assertional content of statements we want to leave out things the other person knows.

Another point to be raised is the representation chosen for the user model. Representing a user as the collection of propositions he knows is simple and effective. Presumably though, one cannot expect to store all the propositions a user knows as this would tax storage considerably. Some thought should be given to how to provide a simple yet storage efficient model of the user.

Issues raised in this discussion included the sequencing of the user model and discourse model stages, and the representation of the user model. Mann and Moore's system provided a highly modular approach not seen in other systems. It fit fairly well with the framework of the general conversation system, having a propositional form (the fragmented propositions), a filtering stage and a logical form (the complex clauses formed by aggregation rules). In addition it provided some attempt to organize the discourse not seen in the general conversation system.

2.7. McKeown

A different approach to discourse modelling is taken by Kathleen McKeown[McKeown 82]. The application is one involving a natural language interface to a naval database containing information about ships and missiles. The propositions in this database are the propositional form of this system. The system produces English output but the interest here is in how it organizes multiple sentence reponses.

Questions are restricted to one of three categories:

Information: the question is about information in the database.

Definition: a request for a definition of a database item.

Differences: the question is concerned about the differences between database entries.

Information relevant to the answer is selected by partitioning the database around the object in question. For questions about differences, the distance between the entities determines the type of information included. That is, the further away the database entities are in the type hierarchy, then the more general is information used to describe their differences. The closer database entries are, then the more specific the information used to describe their differences.

Rhetorical predicates are used to specify how a speaker will describe information. Examples of these include:

Analogy: comparison with a familiar object.

Constituency: description of sub-types.

Attributive: associating properties with an entity or event.

In answering a question, certain rhetorical combinations are more likely than others. Taking advantage of this, schemas (as figure 2.10) consisting of combinations of rhetorical predicates are used to generate an answer. The question type determines which schema will be used. For instance, the answer to a definition question could be generated from an identification schema as in the above figure. This definition involves identifying an item as a member of a generic class, describing the object's function, attributes and constituency, making an analogy to similar objects and providing an example. The definition is built using rhetorical predicates in a specific order and the semantics of each predicate define the type of information it can match in

Question: What is a ship?

Representation: (definition SHIP)

Identification Schema:

```
identification (class & attribute | function)
      [analogy | constituency | attributive]*
      [particular-illustration | evidence]+
      {amplification | analogy | attributive}
      {particular-illustration}
{}    means optional
[]*   means 0 or more times
[]+   means 1 or more times
```

Assuming the Identification schema is selected then
the following would be generated:

(Rhetorical predicates selected by the schema that are responsible for the english generated are printed in boldface.)

identification
"A ship is a water-going vehicle that travels on the surface."
evidence
"Its surface going capabilities are provided by the database attributes DISPLACEMENT and DRAFT."
attributive
"Other DB attributes of the ship include MAXIMUM_SPEED PROPULSION and FUEL."
particular-illustration
"The DOWNES for example has a MAX-SPEED of 29, PROPULSION of STMTURGRD, ..."

A focusing mechanism is used to select between alternative rhetorical predicates.

Figure 2.10 McKeown's text generation system

the knowledge base. A focusing mechanism is used to select between rhetorical predicates when more than one choice exists in a given schema.

Since the types of questions that can be answered are very general, open-ended types of questions, the relevant information may include many propositions. In other words, the system, having decided "what to say", is still left with a large number of propositions in the response. Therefore, knowledge about what constitutes a "good" response is required. McKeown has provided knowledge about discourse structure (i.e.

"good" responses), in the form of schemas. These schemas select information to be included in the response, and organize them according to an accepted standard. This mechanism allows unwieldy responses containing large numbers of propositions, to be further pruned and tidied.

The domain is answering questions about database structure. Because the system can only respond in certain ways, and only to certain questions, this approach is successful. In other domains, such as story understanding, the use of schemas to select propositions appropriate for the answer would require many specific schemas for each type of question. For instance, questions about expectations, goals, missing concepts, and so on, would each require a separate schema. This would probably result in a system very similar to Lehnert's, which is still limited to thirteen categories of questions. The point is, using schemas to select answer propositions sets arbitrary limits on the types of questions that can be answered. These limits are the natural result of systems designed with no deductive capabilities. McKeown allows only three types of questions so using schemas to select propositions for the response is feasible. Thus, while using schemas is an effective way to organize multiple sentence responses, using schemas to decide "what to say" is suitable only for certain domains in which only a very limited range of questions can be asked. In general, we need both deduction and knowledge of discourse structure to form good responses: deduction, to select the initial response propositions, and knowledge of discourse structure, to impose a partial ordering on those response propositions.

A combination of Mann and Moore's problem solving approach and McKeown's rhetorical predicates should be given some thought as a possible discourse modelling technique. That is, instead of schemas of rhetorical predicates, a problem solver, when given a general expressive goal, would select those rhetorical predicates which satisfy that goal. For instance, given a general expressive goal of "definition", one could realize that this is accomplished when the user has a certain amount of knowledge including an identification of the parts of the concept, a listing of its attributes, an appropriate example, and so on In addition, the system would have the knowledge that it is more appropriate to give a particular example of something after, rather than before, that object has been described. The problem solver would use this information to order the rhetorical predicates. The advantage to this approach is that the knowledge used to select the initial ordering of rhetorical predicates is not hidden in the schema, and may be used to select rhetorical predicates for additional types of responses. This would seem a more general approach as specific schemas would not be required for each type of question. The difficulty with this approach lies with defining the relationship between the high level expressive goals and the low level rhetorical predicates.

2.8. McDonald

For the most part, McDonald is concerned with deciding "how to say" something. With his natural language system, McDonald can take a logical proof and translate it into English, using various mechanisms along the way to make the resulting English paragraph more readable[McDonald 83b]. This will be compared to an earlier work by Chester[Chester 76]. This system is particularly related to the general conversation system because it is concerned with the translation of logical proofs. However, it differs from the general conversation system, because it is concerned with generating a literal English translation of that proof rather than extracting propositions from that proof to form an answer. It also differs because the proof is a logical form proof generated by natural deduction and the general conversation system's proof is a set of clauses generated by a resolution theorem prover. McDonald's proof is already in logical form.

Whereas Chester's translation is direct (one sentence per proof step), McDonald attempts to go beyond the literal content of a proof. For instance, a step in the proof may not be realized if there is reason to believe that the reader can infer that step automatically. This is the case for the tautology of step 3 in figure 2.11 :

1. premise
   Ex(barber(x) & Ay(shaves(x,y) ↔¬shaves(y,y)))
2. existential instantiation (1)
   barber(g) & Ay (shaves(g,y) ↔¬shaves(y,y))
3. tautology (2)
   Ay(shaves(g,y) ↔¬shaves(y,y))
4. univeral instantiantion (3)
   shaves(g,g) ↔¬shaves(g,g)
5. tautology (4)
   shaves(g,g) & ¬shaves(g,g)
6. conditionalization (5,1)
   Ex(barber(x)&Ay(shaves(x,y)↔¬shaves(y,y)))
   ⊃(shaves(g,g)&¬shaves(g,g))
7. reductio-ad-absurdum (6)
   ¬Ex (barber(x) & Ay(shaves(x,y) ↔¬shaves(y,y)))

Chester's translation:

> Suppose that there is some barber such that for every person the barber shaves the person iff the person does not shave himself. Let A denote such a barber. Now he shaves himself iff he does not shave himself, therefore a contradiction follows. Therefore if there is some barber such that for every person the barber shaves the person iff the person does not shave himself then a contradiction follows. Thus there is no barber such that for every person the barber shaves the person iff the person does not shave himself.

MacDonald's translation:

> Assume that there is some barber who shaves every one who doesn't shave himself (and no one else). Call him Giuseppe. *Now, anyone who doesn't shave himself would be shaved by Guiseppe. This would include Guiseppe himself. That is, he would shave himself, if and only if he did not shave himself, which is a contradiction.* This means that the assumption leads to a contradiction. Therefore, it is false, there is no such barber.

Figure 2.11 Proof translation by Chester and McDonald

its English equivalent is not generated. Other proof steps may actually be split into multiple sentences if they are deemed liable to confuse the reader. The universal instantiation (step 4) is split into the three italicized sentences in the proof translation. The result is a far more readable proof translation.

How the system generates English will described later. Of interest here is the recognition that some logical structures, are too complicated to communicate in a single sentence and must be split into separate sentences. This observation would seem to support Mann and Moore's arguments for fragmentation of the knowledge base. That is, if the logical proof were instead expressed in clause form, complicated logical structures would have already been split into several clause propositions.

Other logical structures can be assumed to be common knowledge and are not communicated as it would be redundant to do so. This observation would tend to support the necessity of removing propositions known or obvious to the user.

In a later work, McDonald covers the problem of deciding what to talk about first when generating multi-sentence text. Mann and Moore use rules about discourse structure and McKeown uses schemas to determine the relative ordering of subjects. McDonald approaches the problem differently by using an object's salience to determine its ordering[Conklin 82]. Salience may be determined by structural knowledge (task specific), "a priori" or intrinsic knowledge (e.g. people are more salient than mailboxes) and "goodness of fit" knowledge (salience is inversely related to how well an object fits into a particular frame). Propositions with higher salience would be mentioned before those with lower salience. Propositions with a salience rating below a certain threshold would be left out of the description.

In McDonald's system, visual salience is computed as a by-product of visual processing. The result is an English paragraph describing a house. The relevant knowledge has been determined by the visual processing system. It generates a series of propositions with an associated measure of salience. The visual processing system that determines the salience pre-determines the ordering and inclusion of propositions in the response. This approach can only be used in systems where salience can be easily computed. For instance, salience would not be useful in a system like

McKeown's, as it would be difficult to pre-determine the salience ratings of database propositions.

If one is to regard salience as a tool for imposing structure on discourse, then, if salience is to be a static rather than a dynamic calculation, and if salience is not influenced by the current topic, then salience cannot be used to generate cohesive multi-sentence text. By this I mean text which flows smoothly from one topic to the next. A focusing mechanism, as in McKeown's system, is also required if a fluent paragraph is to be generated.

When generating the logical proof translation and the description of the house, McDonald translates directly from a logical form representation into English. McDonald's system differs from the general conversation system and Mann and Moore's system, because his system is already in logical form and requires no knowledge compaction phase to translate propositional form into logical form.

Associated with the logical representation is a set of realization specifications (rspecs), that are assumed provided by a previous component. These realization specifications constrain the top-down process by which the sentence is built. Decisions once made cannot be altered but it is usually unnecessary to change a decision because of the realization specification constraints (see figure 2.12).

```
(rspec1
     (r-spec2 color-of (door3 red))
     (r-spec3 part-of (door3 house1))
     (r-spec4 condense-on-property (r-spec2 r-spec5 red))
     (r-spec5 color-of (gate4 red))
     (r-spec6 part-of (gate4 fence2))
)
```

"condense-on-property" informs the generator to combine two clauses focusing on the common property red.

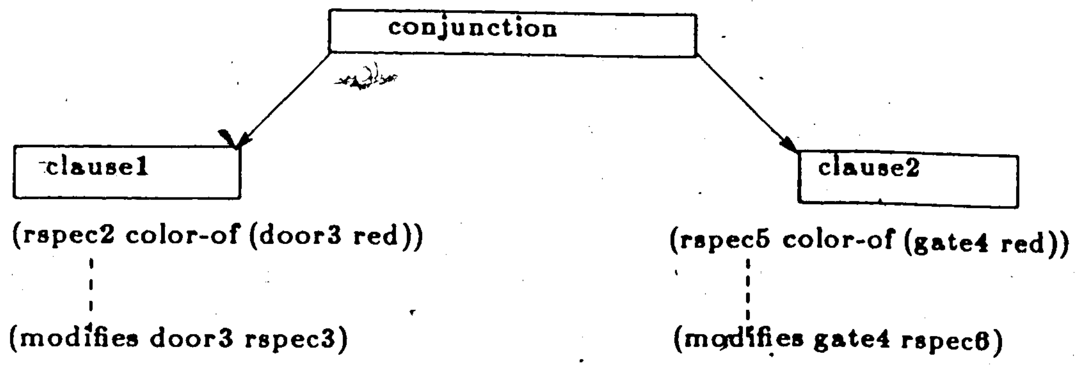
"The door of the house is red and so is the gate in the fence"

Figure 2.12 A sample realization specification

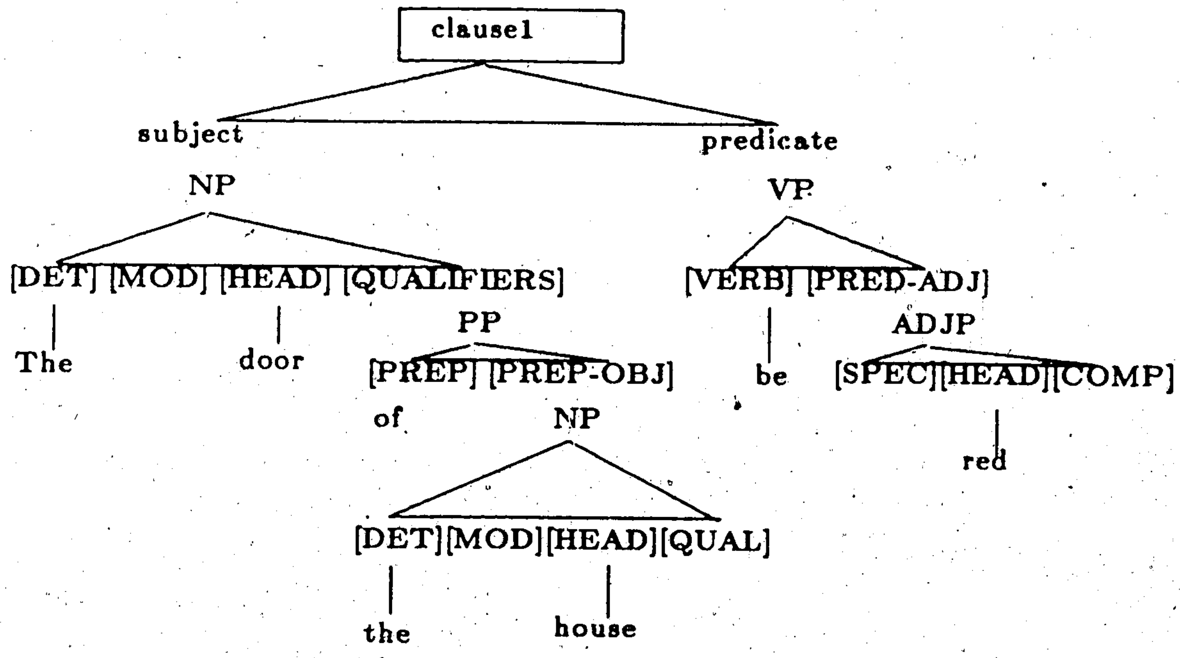
The realization specifications not only supply ordering information, but also, information about which propositions to combine. Thus, the realization rules contain knowledge about how to order propositions and how to assemble them further into a more compact logical form. In McDonald's system, this information conveniently comes "free of charge" as a by-product of a vision understanding system. In general, most systems could not expect this knowledge to be supplied free of charge, so this seems to support the necessity of providing separate pre-processing stages which imposes an ordering on multiple propositions and assembles the propositions into a logical form.

McDonald uses a procedural grammar known as systemic grammar for the final English verbalization. In Figure 2.13 the realization specifications are translated into English by replacing each rspec with a syntactic structure. The appropriate syntactic structure for each realization specification is selected by a decision procedure. This procedure, a related set of mutually exclusive grammatical features one of which must be chosen upon entering the procedure, is known as a system. Associated with each system are entry conditions which must be satisfied before the system may be entered. The choice selected by the system ranges from a word to a syntactic constituent or even further realization specifications.

In figure 2.13 , the conjunction node, the "modifies" relations, and the "VP-deletion" on the second clause are direct results of the default decision to realize the "condense-on-property" rspec as a conjunction with the verb phrase deleted from the second clause. As the realization process progresses, rspecs are replaced by specific syntactic structures, these in turn may expanded into other syntactic constituents until finally a word is selected. Rspecs contain the information used to select particular words and these words are passed to a morphology routine before being output.



Expansion of clause1 produces the following subtree:



The tree is generated in depth-first, left-to-right order.

Generated so far: "The door of the house is red"

Figure 2.13 A sample syntactic tree structure produced by McDonald's system

A record of the discourse context is used to pronominalize and select appropriate determiners. For instance, "a" is chosen for objects when they are first mentioned, and "the" for objects already introduced in the discourse.

Systemic grammar is a very popular grammar for generating natural language, (see[Davey 78, Mann 83, Patten 85, Winograd 71]). The difference between generation systems that use systemic grammar is reflected by the choice mechanism they use to select features upon entering a system. Mann uses procedures called "choosers" which interact with the environment to select appropriate features. Patten uses a problem solving approach, capitalizing on the similarity between systems and operators in a planning system, to make the appropriate choice. McDonald, in addition to the constraints imposed by the realization specifications, cannot alter a decision once it has been made, and must base a decision on information local to the point in the surface structure tree which is currently being generated.

Systemic grammar has no well specified semantic component, and no explicit representation of the relationship between the syntax and semantics. Since there are grammars that do provide this, it would seem worthwhile to try natural language generation using one of these. Generalized Phrase Structure Grammar and Lexical Functional Grammar are possible candidates.

All the systems discussed made a distinction between deciding "what to say" and deciding "how to say it". In particular, each presented some ideas for what must go into each stage of a response generation system.

The systems based on Schank's conceptual dependency formalism were both characterized by forward inferencing. The earlier system used only forward inferencing and thus deciding "what to say" involved searching memory for an already existing answer concept. The later system, BORIS, used deductive inference as well and was not dependent on question category for finding the answer in memory; the search for the answer in memory was aided by inference carried on as the question was understood and by deductive inference if necessary. It would appear that the ability to perform deductive inference as well as forward inferencing is an important feature of the "what to say" stage.

The knowledge structures required for the "what to say" stage should be able to reflect goals and intentions as well as simple propositions. This type of knowledge is expressed in the MOPs of BORIS, and the speech act operators in Cohen's natural language planning system. Models of the speaker's and hearers beliefs and goals were used in the natural language planning system to decide "what to say". This approach however fails in the absence of specific goals which may be attributed to the speaker and hearer.

Using the conceptual dependency formalism, it is difficult to express concepts such as "fear", "hope", "belief", "wants", etc. Thus a system based on this formalism can expect to be somewhat limited.

As far as deciding "how to say it", Mann and Moore's system was the only one to incorporate a user model to allow filtration of known propositions from the response. This filtration stage is seen as necessary to producing non-verbose English output. The rather impractical representation of the user model as all the propositions that a user can be assumed to know, ould seem indicate that further thought should be given to the user model representation.

Mann and Moore's system also provide an idea as to how to order propositional form for multiple sentence responses, namely, according to rules specified for a particular expressive goal. McKeown and McDonald's system ordered propositional form according to schemas and salience respectively. All these systems have some way of ordering propositions before they are translated into English.

To assemble propositional form into logical form, Goldman used discrimination nets and Mann and Moore used clause combining rules for English. The first used pre-defined choices to assemble a logical form; the second, numerically evaluated preference rules which determined the best possible logical form. It remains to be seen if the assembly of propositional form into logical form can proceed without the use of pre-defined choices for each kind of propositional form (too deterministic) or numeric values intended to reflect the best combination of propositional form into logical form (too difficult to generalize for each new preference rule). Some thought should be given to the possibility of removing the need for numeric values and using instead knowledge about linguistics and logic to perform the assembly stage.

To produce the English verbalization, none of the systems used a grammar which had a semantic component and explicit representation of the relationship between syntax and semantics. It would definitely seem worthwhile to use such a grammar in a natural language generation system.

Lastly, there seems to be a lack of what could be called a "theory of natural language generation". Unlike its counterpart, natural language understanding, there seems to have been no attempt to describe the theoretical processes underlying natural language generation in terms of current syntactic, semantic and pragmatic theories. The generation process should be broken into well-defined stages; each stage accomplishing a mapping from one level of representation to another, each exploiting different levels of linguistic knowledge to accomplish the mapping, eventually generating English.

Chapter 3

An English Generation System

3.1. Introduction

A small English generation program was written and will be described in this chapter. The system was implemented first in MACLISP (running MTS on an AMDAHL 5860), and then in Franzlisp (running 4.2BSD UNIX on a VAX 11/780). Following a description of the story understanding system that it was designed to work with, the English generation program itself will be described, after which, some shortcomings of the design will be discussed.

The general conversation system can answer questions about stories. The system's knowledge, both story specific and general, is represented as propositions attached to concepts, and organized at each concept in conformity with a general topic hierarchy. These propositions contain the information required to answer the questions. To start, a question that has been parsed and translated into logical form is posed to the system. To answer this question, it is first converted to clause form and then a resolution theorem prover simultaneously attempts a proof and a disproof. When either has completed, the question is answered either "Yes" or "No", depending which finishes first: the proof or the disproof. The goal of the English generation facility was to improve on the simple "Yes/No" answer by appending an explanatory English sentence to it.

The theorem prover has decided "what to say"; the relevant knowledge to be expressed in the response is a subset of the clauses in the proof. Assuming a mechanism for choosing that subset exists, then what remains is to decide "how to say" these proof clauses in English. The problem is to translate a series of propositions in clause form into an English sentence. A program was written that could accomplish this translation, and will be described next.

3.2. Input

The program accepts as input an initial answer (YES or NO) and a set of proof clauses presented in a list. A clause is a list of either positive or negative literals, and literals are negated or unnegated atoms. The terms are ordered in infix notation, specifically a subject term, a predicate term and an optional object term.

```
INPUT      ::=  (YES | NO
                     (SEQUENCE OF ZERO OR MORE CLAUSES))

CLAUSE     ::=  (SEQUENCE OF ZERO OR MORE LITERALS)

LITERAL    ::=  ATOM | (NOT ATOM)

ATOM       ::=  (SUBJECT PREDICATE {OBJECT})
```

Subjects and objects may be skolem constants, name constants or variables. Constants and variables are distinguished by the associated property "TYPE" that has the value either "CONST" or "VAR". Skolem constants begin with a lower case letter and name constants, an upper case letter. Functions are not allowed. The types of predicates allowed are adjective-like, noun-like, verb-like and preposition-like logical predicates. A list of the limitations of this system follows.

3.2.1. Limitations

Because of the simple grammar, only a certain class of clauses could be directly translated into English. The input restrictions are as follows:

1. As mentioned at the outset, the restrictions to standard clause form already entails severe limitations in what can be talked and reasoned about. Time order in narratives, causal connections, goals of characters in stories, and other modal relationships are not handled. So, for example, questions like "Why did the wolf get dressed up as Little Red Riding Hood's grandmother?" can not be asked or answered, and hence are not considered here.

2. Clauses may contain any number of adjectives but only one noun.

3. Skolem functions are not allowed.

4. Predicates that are verbs may have only one or two operands. In particular, verbs like GIVE, TAKE and LEND are not allowed.

5. Clauses may only contain at most two literals. The second literal must be negative.

6. The subject of the generated sentence is arbitrarily chosen as the first term at the beginning of the first clause.

7. If one clause has a NOT, then no other clause may contain a NOT.

While most of these limitations are arbitrary and mainly due to the simple grammar used, some are inherent difficulties that arise when attempting to translate clause form into English. In particular, skolem functions are difficult to translate. Also, a subject is difficult to choose without focus information, and it is difficult to translate a series of negative clauses without some stylistic knowledge of the type of sentence to be built.

3.3. Program description

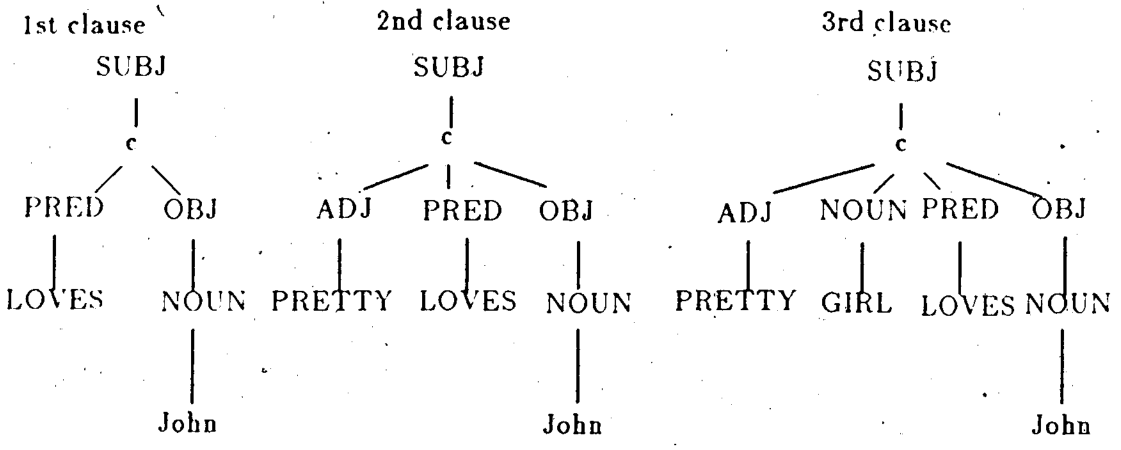
The program works by parsing the list of clauses and generating a tree representation of the sentence to be generated. The root of the tree is arbitrarily chosen to be the subject of the first clause, and it will be the subject of the English sentence to be generated. As each clause is parsed, each term in each literal is classified according to syntactic category. A logic to English lexicon, where the syntactic category and English equivalent of a logical term are stored, is used to classify terms. After being classified, the atom is added to the tree underneath its category. To generate English, a quantifier is chosen (see section on quantifier selection) and then each leaf node is visited once in depth-first, left to right order and replaced by its English equivalent.

Figure 3.1

Input: (YES (((c LOVES John)) ((c PRETTY)) ((c GIRL))))

Clause Form: (c LOVES John) & (c PRETTY) & (c GIRL)

After parsing each clause:



English : Yes, a pretty girl loves John.

Figure 3.1 Translating Clause Form into English

shows an example of how English is generated with this system. As shown in this figure, the information in each clause is added successively to the tree. (See Appendix A1 for some sample output).

Variables or constants with no associated type information (i.e. they have no noun-like predicate to describe them), are automatically assigned the default category "thing". For names, the English translation is printed and for skolem constants, an indefinite determiner is added as well. Nouns requiring a definite determiner are treated as special name constants where the name of the object is actually "the <noun>".

3.3.1. Quantifier Selection

The part of the program that remains to be described is the part which selects the appropriate quantifier for each noun phrase. Different quantifiers are chosen depending on whether the subject or object of a sentence is a constant or a variable. The choice of quantifier also depends on the mode of the sentence to be generated; that is, whether the sentence is positive or negative. In general, skolem constants have the quantifier "some" or "a"; variables have the quantifier "every" or "no". The mode of the sentence was assumed to be singular. A list of rules for quantifier choice follows:

1. Skolem constants always have positive quantifiers, and the positive or negative aspect of the sentence is expressed by the main verb.

Logical form:	EX[(X GIRL) & (X LOVES John)]
Clause form:	(c GIRL) & (c LOVES John)
English:	**Some girl loves John**
Logical form:	EX[(X GIRL) & (NOT (X LOVES John)]
Clause form:	(c GIRL) & (NOT (c LOVES John))
English:	**Some girl does not love John**

2. For verb-like predicates with more than one operand: variables always have positive main verbs, and the positive or negative aspect of the sentence is expressed by the quantifier. For example:

Logical form:	AX[(X GIRL) → (X LOVES John)]
Clause form:	(X GIRL) OR (NOT (X LOVES John))
English:	**Every girl loves John**
Logical form:	AX[(X GIRL) → (NOT(X LOVES John))]
Clause form:	(NOT (X GIRL)) OR (NOT (X LOVES John))
English:	**No girl loves John**

3. For ~~verb-like predicates w~~ith a single operand: variables always have the positive quantifier "Every" and the positive or negative aspect of the sentence is expressed by the main verb.

Logical form:	AX [(X LEFT)]
Clause form:	(X LEFT)
English:	**Everything left**
Logical form:	AX [NOT(X LEFT)]
Clause form:	(NOT (X LEFT))
English:	**Everything did not leave**

4. Name constants require no quantifiers.

These rules were used to determine the appropriate quantifier for each sentence. A post-processor scanned the sentence and changed "a" to "an" if it was followed by a noun beginning with a vowel.

Though this program attempted a very limited covering of English, and the types of clauses which it could translate were also limited, it was enough to discover some problems with translating clause form directly into English.

3.4. Difficulties with direct translation

3.4.1. Grammar

The procedural grammar I used complicated matters considerably. As a result only a limited coverage of English was achieved, and only a limited type of clause could be successfully translated. The procedural grammar lacked an explicit representation of the relationship between clause form and the syntax of English. Instead, this relation was hidden in the code. Extending the grammar usually resulted in some unwanted side effects. Even with well written, modular code, the tendency of the programmer is to the hide the semantics in the code. An explicit representation of the relation of clause form to logical form and logical form to surface English is necessary if one is to write a generation system with an extensive coverage of English,

3.4.2. Absence of logical form

The direct translation of clause form into English was difficult in the absence of an intermediate processing stage that generated logical form. In particular, it was very difficult to know when a "NOT" is a "NOT", or part of an "implies". The following are equivalent expressions when changed to clause form:

Logical form:

$AX((X\ WHALE) \rightarrow (NOT(X\ FISH)))$	No whale is a fish.
$AX((X\ FISH) \rightarrow (NOT(X\ WHALE)))$	No fish is a whale.

Clause form:

$$(NOT\ (X\ WHALE))\ OR\ (NOT\ (X\ FISH))$$

Without any extra stylistic information, there is no way of knowing which would be the most appropriate condensation. That is, there is no way of knowing whether "whale" should be mentioned first or "fish" should be mentioned first. The best answer would depend on the current discourse context. A separate pre-processing stage using knowledge about discourse and how to organize it, as well as a record of the previous discourse context, would be the most appropriate place to make a decision about which is the appropriate logical form to generate. Assuming the information required to render the appropriate logical form is provided by the discourse component (even though this just shifts the problem from one component to another) is valid because the appropriate rendering does not automatically arise from the structure of the clauses. Also, the type of information required to decide upon a particular logical form differs from that required to perform the actual translation into logical form.

Because of the above problem, only one clause in the input clauses could contain a "NOT", but that clause could have more than one "NOT" and to generate the appropriate English, the second literal had to be negative. Also, an arbitrary choice was made whenever possible to produce the most natural sounding version of a particular sentence.

Another problem that arose was in the formation of descriptive declarative English sentences. Starting from the same clause form and logical form, at least two different English translations are possible.

Clause form:

((c PRETTY) AND (c GIRL))

Logical form:

(PRES (SOME (L X (X PRETTY) (X GIRL)))
There is a pretty girl.

((SOME GIRL) (PRES (PRETTY)) Some girl is pretty.

Also,
Clause form:

(GM ASLEEP)

Logical form

(GM (PRES ASLEEP)) Grandmother is asleep.

((GRANDMOTHER LRRH) (PRES (PROG SLEEPS)))
Little Red Riding Hood's grandmother is sleeping.

((THE (L X(X OLD)(X LADY))) (PRES (PROG SLEEPS)))
The old lady is sleeping.

- etc.

The appropriate English rendering is not implicit in the structure of the clauses. Again, the decision between the possible versions hinges on the structure of the current discourse. From these two examples one can see that the appropriate logical form rendering is not inherent in the clause form, and additional information, other than how to assemble clause form into logical form, (i.e., information about discourse structure) is required to generate the appropriate logical form. Information about discourse structure is needed to decide what to put first in a sentence (e.g. "It was *grandmother* that the wolf ate"), for connectedness (e.g. "Anyway ...", indicates we're resuming an earlier point or story, after a digression), and for descriptions, so they may be chosen to pick out referents in the current context. In the first example, the response generator written arbitrarily chose the second version.

While clause form propositions are an ideal representation for carrying on inference, and topical classification, they are less than ideal as a starting point for translation into English. This is because a sentence expressed in clause form is too dissimilar from English. For instance, twenty clauses could perhaps be condensed to one English sentence. Also, there are no grammars for converting clause form to English, while there are linguistically justified grammars for converting logical form into English. That is not to say such a grammar could not exist; if it did exist it would of necessity be much more complicated than a logical form to English grammar, and it would seem simpler to provide a two stage mapping of clause form to logical form and then logical form to English rather than complicating the problem by attempting a single complex mapping.

3.4.3. Modularity

The generation of English was done in a single step. To generate English in a single stage, one has to consider information about the user, discourse structure, knowledge condensation, and English grammar. In other words, all four "how to say it" components of a generator must interact at once. Although the program used only the latter two kinds of information, it became very complex and extending it became difficult. If it was using all four kinds of knowledge in one stage, the program would be exceedingly complex.

The purpose of this program was to provide a useful generation program quickly, thus, several components of an ideal generation program were sacrificed for the purpose of expediency. It did not, for instance, have a filtration stage to remove clause propostions known or obvious to the user. It did provide some idea of the requirements for translating clause form to English. Specifically, the necessity of a preprocessing stage to prov ie some organizational information, and the desirability of using an intermediat logical form and a linguistically justified grammar.

All of the above comments point to a need for a better designed system that could circumvent some of the problems encountered in this rudimentary discourse generator. Some ideas for a better design that incorporates some of the issues raised by this and other previously mentioned natural language generation systems will be presented next.

Chapter 4

Towards a better design

This chapter will present further detail for the design of a response generation system. In particular, this design will attempt to cover some of the deficiencies of the rudimentary response generator discussed in the previous chapter. First, some ideas for the initial deciding "what to say" stage, followed by some ideas for the "how to say it" stage will be presented. In particular the filtering, organization, assembly, and verbalization stages will be described in more detail. The ideas for the "how to say it" stage relate to the general conversation system mentioned in previous chapters. That is, the inference and retrieval used to generate the relevant information to be contained in the response is assumed done by a resolution theorem prover. The problem is to translate these proof clauses into an English answer.

4.1. Deciding "what to say"

A first order resolution theorem prover is not sufficient for certain types of questions. Higher order logic may be needed. More importantly, full modal logic is needed if one is to reason about propositional attitudes, causes, counter-factual conditionals, and so on. Although the semantic network may represent modal logic, there is no inference mechanism to handle modal predications.

A general conversation system must take into account the user's goals when generating responses, therefore, in addition to modelling the user's knowledge or beliefs, the system must also have a record of the user's goals. A natural language planning mechanism would be a first extension to the deciding "what to say" stage. A first order resolution theorem prover is not sufficient if one is to answer questions about goals of characters in stories, or initiate dialogue rather than just passively respond. The knowledge representation in the story understanding system, is suited to a certain extent to this problem already. That is, it already has the capability of representing different points of view of the "real world" imposed by modal predications such as "John believes ...", or "Little red riding hood is a story about ...", using subnets. These subnets represent a storage efficient, and fast way of accessing propositions a particular user can be assumed to believe. However, as mentioned before, there is not as yet a way to reason about these modalities. The current system lacks a representation for goals, and a planning mechanism for using the goals of the speaker and hearer. It would be desirable to implement a planning mechanism if one wanted to extend the general conversational capabilities of the system.

The natural language planning mechanisms discussed in chapter two, speech acts were limited to requests, and informs. Since other types of speech acts exist, a general conversation system should also be able to generate other types of speech acts, such as permitting, warning, promising and retracting.

4.2. Filtration

Relevant response propositions can be filtered from a proof by removing clauses that are either set of support (i.e. clauses immediately derivable from the denial of question to be answered), type predications, or meaning postulates. The rationale behind omitting these types of clauses is that we can assume they are already known to the user and it would be redundant to communicate these facts to him. For example, to answer the question "Is the wolf grey?" the following proof was generated by the resolution theorem prover:

```
Question: ?[W grey]                 "Is the wolf grey?"

    1. ~[W grey]                denial of conclusion
    2. ~[x wolf] or [x grey]    "All wolves are grey"
    3. ~[W wolf]                r[1,2b]
    4. [W wolf]                 "W is a wolf"
    5. []

Answer:      Yes, all wolves are grey.
```

This answer was obtained by removing the set of support clauses (1 and 3), and the type predication clause (4). Since the remaining clause (2) is not a meaning postulate it was selected and appended to the initial response. The theorem prover generates the initial response "Yes" or "No" depending on whether the proof or disproof succeeds. If clause 2 had been marked as a meaning postulate then the answer would have been simply "Yes".

A more general approach, instead of relying on meaning postulates which supply an implicit prototypical user model to the system, would be to delete propositions from the proof based on a detailed model of the individual user. However, it is not clear how to best represent individual users; storing detailed models of each user would use a lot of memory. More importantly, it is not clear where these models would come from. There is a trade-off between detailed representations of each user and storage requirements. By supposing set of support clauses to be known to the user, one immediately reduces the number of proof propositions to be considered for the response. This is achieved without any increase in the specific facts assumed known by each user but by the higher level knowledge that "anything immediately derivable from the question clauses can be assumed known by the user". The advantage is that while no extra storage is required, some of the propositions known or obvious to the user can be determined from the question itself. Taking this one step farther, it is not unreasonable to assume that the user is aware of the previous discourse. Propositions that occurred in this context can be filtered out. So one can tailor responses to an individual using a prototypical user model (i.e., propositions flagged as meaning postulates in the knowledge base), supplemented with a record of the previous discourse.

There is a problem with assuming a prototypical user whose knowledge consists of those propositions marked as meaning postulates plus, presumably other "common knowledge". If someone is asking for a definition, the answer will most likely be made up of meaning postulates. It would not be appropriate therefore to delete these propositions from the response. Requests for definitions, information and differences, would require special inference methods (like McKeown's schemas[McKeown 82]) to answer so it is not likely that the proof from a first order resolution theorem prover would be the starting point for generating these kinds of responses. Deleting meaning postulates would not be useful for generating responses to these types of requests.

However, it is also unlikely that a prototypical user model would be sufficient however for domains where what could be deemed common knowledge varies vastly from user to user. The tutoring or expert system domain are examples of the types of domains that must explain their behaviour and tailor their explanation to the user. In giving a description of its reasoning an expert system may supply different levels of detail depending on whether it is talking [illegible] an expert or novice. A tutoring system must assume at the start that the user has little or no knowledge of the subject area in which he is about to be tutored. This model must be updated as the user gradually acquires more knowledge through the tutoring process. User knowledge could perhaps be organized in a hierarchal fashion where each node in the hierarchy represents the knowledge that a user can be assumed to know. An expert would be at the top level of the hierarchy.

In general, then, one can distinguish between the types of user models needed for everyday conversation and those which must be used in systems which use a lot of domain specific knowledge. For everyday conversation, assuming a prototypical user model to encode common knowledge is sufficient when supplemented with the additional knowledge that the user is aware of all that has occured in the previous discourse.

One thing I haven't mentioned are goals, which of course must be part of a user model especially if one is to use planning to decide what to say. One is assuming, though, that user goals do not pose an organizational problem since they may be assumed limited in number at any given point in time. The goals of the user would be needed if one was to do any natural language planning.

4.3. Organization

This stage should provide a partial order for the response propositions, assuming that a response is potentially several sentences long. It is not as vital for producing single sentence responses but it is necessary for natural sounding multi-sentence responses.

The response propositions could be ordered using common sense knowledge about discourse. For instance, when describing an event propositions could be ordered according to time or using knowledge like the fact that something exists should be mentioned before it is mentioned in some other way (as in Mann and Moore's system). One could also use the trace left behind by the inference process to structure the answer. So response propositions could be ordered according to the proof steps (as in McDonald's system). Other techniques such as using schemas (McKeown's) and salience (McDonald) could also be used to order propositions.

Some mechanism must be designed to order propositions so that discourse flows naturally from one topic to another. A hint of what the topic of each sentence is must therefore be provided. By topic, one means the current concept of most relevance in the discourse. This could also use common sense rules such as group propositions similar in topic together and keep on the same topic long as long as possible before changing topics.

4.4. Assembly

Even though clause form representation is unambiguous, a particular clause form may have several different logical form translations. The task in this stage is finding a way to determine a logical form from a particular clause form. The organization stage has selected a set of clauses that are to be expressed as a single sentence and passed them on to this stage. This stage involves splitting the response into multiple sentences if necessary, choosing the form of each sentence, and picking appropriate descriptions for objects within each sentence.

It is not clear how to select the clauses that should appear a single sentence. Given a group of propositions that must be expressed as multiple sentences, it is not clear how to divide those clauses up into sets, where each set of clauses comprises the information to be conveyed in a single sentence. In Mann and Moore's system, propositions are combined using rules based on simple clause combining rules of English. The resulting aggregates were evaluated using numerical preference rules. It is not clear whether preference rules based on linguistic knowledge rather than numerical values could be used to evaluate the sentence size groupings produced by the aggregation rules. Also, it is not clear how these rules would handle sentences with several conjunctions and disjunctions. A technique for selecting the appropriate amount of information to be conveyed in the sentence must rely on knowledge about discourse structure (e.g., it is normal to vary sentence length in a paragraph and to vary between using simple and complex sentences) as well as any inherent clusterings of propositions which may arise naturally from the clause form.

Assuming that the set of clauses to be expressed in a single sentence have been chosen, some more work must be done to translate these clauses into logical form. Since clause form may be considerably larger than logical form, one must first determine a logical form with the minimum number of terms. In the response generator described in the previous chapter, syntactic categories were used to group predicates that contained the same variable or constant to obtain the logical form. The desirability of confining the use of syntactic knowledge to the verbalization stage (to make it easier to generalize the system to other natural languages) prompted the search for alternative ways of combining clauses into logical form.

The problem is to compact a set of clauses into a logical form with the minimum number of terms. For example, assuming a simple T/F propositional logic rather than predicate logic:

> "Either it's winter and John is wearing boots, or it's summer and John is wearing sandals.[Rich 83]

translates into logical form as:

[[winter] Λ [wearingboots]] V [[summer] Λ [wearingsandals]]

and into clause form as:

[winter] v [summer] Λ
[winter] v [wearingsandals] Λ
[wearingboots] v [summer] Λ
[wearingboots] v [wearingsandals]

Assuming the above clauses are to be translated into English, there is an initial problem of compacting these clauses into a more concise notation. The problem is to find the most compact representation for the above clauses. One way to do this is to treat this problem as equivalent to the problem in switching theory of minimizing the number of inputs to a set of logic gates, or minimizing the number of literals appearing in a logical expression. Since the clauses in the proof will be in a product of sums form (conjunctive normal form), this must first be converted into a sum of products form (disjunctive normal form) before the minimization can begin.

a - winter b - summer
c - wearingsandals d - wearingboots

Product of sums:

$$f = (a + b)(a + c)(d + b)(d + c)$$

(In this notation an addition "+" means OR and
a product means AND.)

The above expression can be minimized by, first, converting it into its canonical form. This is done by adding the product xx′ to each missing variable x in each factor.

$$\begin{aligned} f = \; & (a + b + cc' + dd')(a + bb' + c + dd') \\ & (aa' + b + cc' + d)(aa' + bb' + c + c) \end{aligned}$$

$$\begin{aligned} f = \; & (a + b + c + d)(a + b + c + d')(a + b + c' + d) \\ & (a + b + c' + d')(a + b' + c + d)(a + b' + c + d') \\ & (a + b + c' + d)(a' + b + c + d)(a' + b + c' + d) \end{aligned}$$

Once the canonical product of sums is obtained, this can be converted into a sum of products expression by using the involution theorem $(x')'$.

$$f = (f')'$$

The complement f′ consists of those factors which are not contained in f:

$$\begin{aligned} f = \; & [((a' + b' + c' + d')(a' + b' + c' + d)(a + b' + c' + d') \\ & (a' + b + c' + d')(a + b' + c' + d)(a' + b + c + d') \\ & (a' + b' + c + d'))']' \end{aligned}$$

$$f = abcd + abcd' + a'bcd + abc'd + a'bcd' + ab'c'd + ab'cd$$

The last form is the canonical sum of products. This can be minimized by the tabulation method (also known as the Quine-McCluskey algorithm). The tabulation method is a systematic procedure for finding the set of prime implicants, which may then be used to find the minimal sum of products expression. Applying this technique, the following minimal sum of products expression for f can be obtained:

$$f = ad + bc$$

Although this seems like a lot of work, it is guaranteed to find the minimal sum of products expression for a given logical expression. To find the minimal expression whether it be product of sums or sum of products one must determine both forms and pick the one with the least literals. The above exercise was an attempt to show that there is a systematic way of achieving the most compact representation of a set of clauses.

The advantage of this approach is that it should always find the most compact representation of a set of clauses. The disadvantage is that the efficiency of the above method would degrade rather rapidly with an increase in the number of variables. It also seems to be a slight case of overkill. It remains to be seen if there is another way of combining clauses into logical form, that does not use syntactic knowledge yet accomplishes the task in a simple manner. Also, should the organization component affect the logical form generated, it is conceivable that the above clauses should not be compacted at all, but expressed as they are. It may be that a more verbose description is demanded for the current discourse context.

The last problem to be considered when translating clause form into logical form, is determining the best way to express a constant appearing in a clause form proposition, as a logical form term. In other words, appropriate descriptions of objects must be built. For example, the clause form

[c PRETTY]

could be expressed in logical form, as one of the following three:

<SOME (L x (x PRETTY) (x THING))>

<THE (L x (x PRETTY) (x THING))>

<SOME (L x (x PRETTY) (x GIRL))> (assuming [c GIRL] present)

Some rules for expressing a constant c in logical form are as follows:

1. If c is upper-case it remains c in the logical form. This is so name constants like John1 and Mary2 are handled appropriately.

2. If c is lower case then:

(a) Find the TYPE P of c where:

P is THING by default and is whatever predicate appears in the known type predication of the form [c P] otherwise.

(b) If c is not in the current conversational context then its logical form is <SOME P>

(c) If c is in the current conversational context then the logical form is <THE P> in the simplest case (i.e. if c is the only P in the current context).

The last two rules require a record of the previous discourse to select the appropriate quantifier.

A small clause form to logical form translator was written. The set of inputs, the same as described in the last chapter, were sufficiently constrained so that no compaction was needed, and whether or not c, the constant to be expressed, was to be expressed as <THE P> or <SOME P> was assumed given.

To combine clauses into a single logical form, some grammatical classifications were used. For instance, type (noun-like) predicates were combined with modifier (adjective-like) predicates into lambda expressions. The presence or absence of an action (verb-like) predicate determined whether a lambda expression or a term should be formed (see Appendix A-1 for a sample of the logical form which this program produced).

(e.g. [Mary (PRES (L x (x PRETTY) (x GIRL)))]

versus

[Mary (PRES (HATES (SOME (L x (x PRETTY) (x GIRL))))]).

Notice an explicit "AND" is not generated.

The "PRES" predicate modifier is intended to represent the present tense of the verb "be" in the absence of a verb-like predicate, and the present tense of a verb-like predicate.

It is expected that predicate modifiers could be nested so that verb phrases with the appropriate tense and aspect could be generated. For example, given the following tense and aspect operators:

Aspect operators:	Tense operators:
PERF - "have - en"	PRES = "is"
PROG - "be - ing"	PAST = "was"
GOING-TO - "be going to"	FUT = "will"

(PAST (GOING-TO (PERF sleep))) = was going to have slept
(PAST (PERF (PROG sleep))) = had been sleeping .

This was not implemented, and if it was it would be part of a morphology routine associated with the final verbalization stage. It is expected that either the initial decision of "what to say" would actually generate the modifiers or they would already be part of the knowledge base. It was assumed that the default tense would be PRES for the system I wrote.

As mentioned before, grammatical knowledge used in this stage detracts from the generality of the system because if a language other than English was to be generated, then it would involve more than simply replacing the grammar in the verbalization stage. For this reason, it is desirable to find other ways of distinguishing whether a term or lambda expression should be generated. It did use a relatively simple method for combining clauses into logical form, that of grouping predicates relating to the same logical term with that logical term. So all the predicates relating to Mary would be grouped with Mary. The order of predicates appearing in the logical form depends on the type of predicate: modifier predicates come before type predicates when generating a lambda expression and then the action predicate, followed by any predicates relating to the object. This may appear to restrict the logical form generated, but it will not restrict the English sentences generated (as seen in the next section).

This program was considerably simpler than the one that directly translated clause form to English. This was to be expected as it did not attempt to generate English. The logical form generated by this program would be the starting point for the next component, which translates logical form into English using a grammar. This program should be extended to be able to handle explicit "ANDs" like (JOHN (PRES (LOVES (AND MARY SUSAN)))) so that complex as well as simple English sentences may be generated.

4.5. Verbalization

Verbalizing the logical form into English would be accomplished using Generalized Phrase Structure Grammar (GPSG). This grammar is chosen because of explicit representation of the relationship between logical form and surface English in the grammar. Because of this explicit representation, as the logical form is parsed, the appropriate English translation can be generated at the same time. Since the natural language understanding component is going to be using GPSG to parse English into logical form, using GPSG to do the reverse adds a certain symmetry to the system. Assuming a logic to English lexicon where English words for logical terms may be looked up, a part of a grammar going from logical form to English might look like the one shown in figure 4.1.

The generation of the phrase structure tree would proceed in a bottom up manner. Each atomic term in the logical form would be assigned a syntactic category using the lexicon "in reverse". In general several choices may be available; e.g. the predicate ASLEEP may be verbalized as either the adjective "asleep" or the verb "sleeps". The grammar rules would be applied to build the tree from the bottom up.

<1,[(NP) (PN)], PN'>

<2,[(VP) (V)], V'>

<3,[(S) (NP) (VP)], [NP' VP']> [] = infix form

<4,[(AN) (ADJP) (N),(L x (x ADJ') (N'))>

<5,[(NP) (Q) (AN)], <Q' AN'>>

<6,[(VP) (V) (VP INF)],(L x (x V' (x VP')))>

<7,[(VP INF) (to) (VP BASE)], VP'>

<8,[(VP) (V) (NP) (NP), (V' NP' NP')> ( ) = prefix form

<9,[(VP) (V)], V'>

<10,[(VP) (V) (NP) (PP to), (V' NP' PP')>

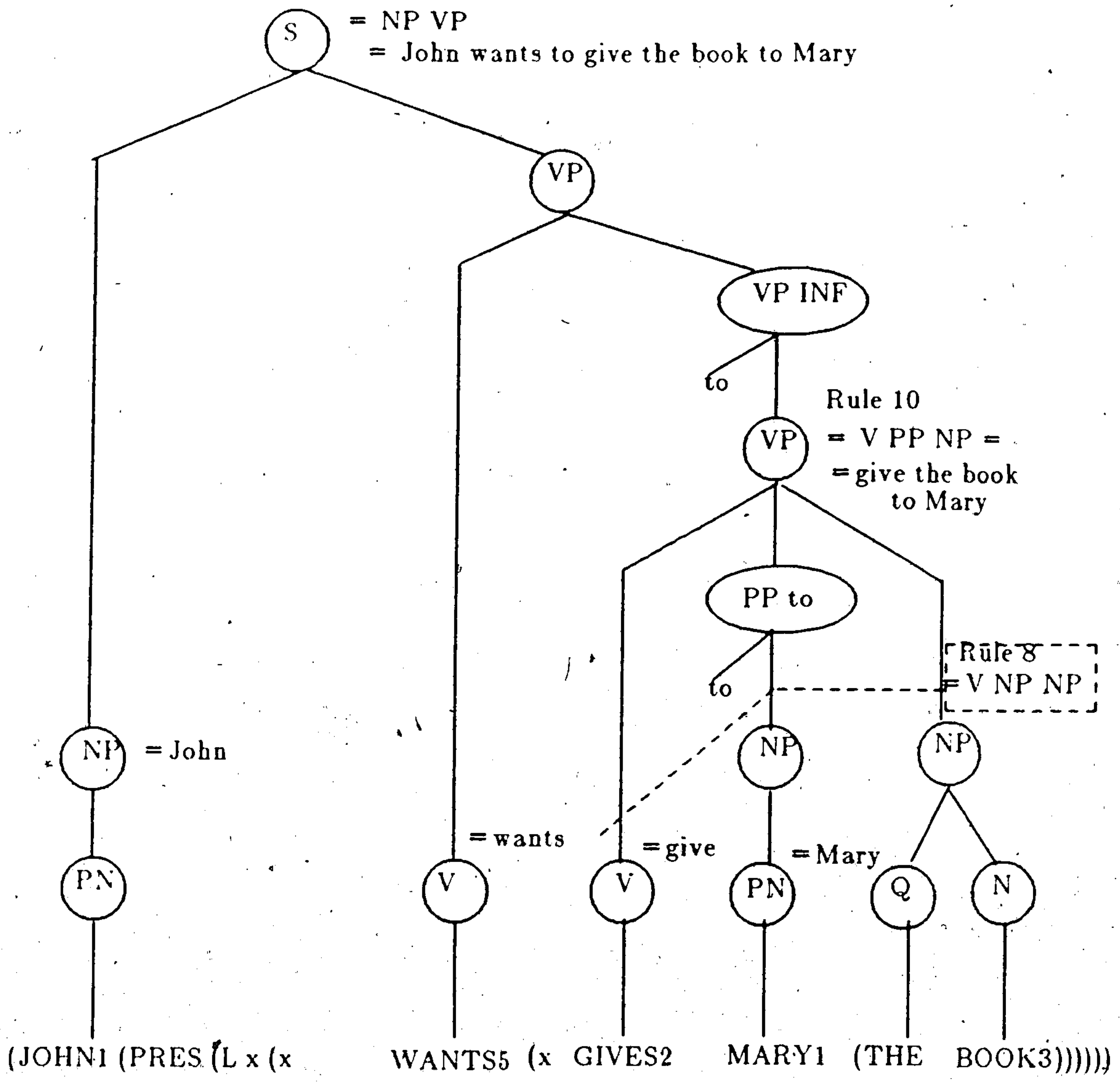
<11,[(PP to) (to) (NP)], NP'>

where each rule consists of the following,

a number, a phrase structure rule, and its logical translation.

Each phrase structure rule consists of a lexical category followed by
a the phrase structure constituents which make up that lexical category.
The primed category symbols denote English translations of constituents.
A lambda function is represented as (L x (x P) ... (x Q)).

Along the way, the translation part of the grammar rules is matched against a fragment of the input logical form, whose highest-level constituents have already been labelled with syntactic categories (via the lexicon or previous matches). If the match succeeds, the category of the left hand side of the corresponding syntactic rule is assigned to the fragment as its label. As in the lexical phase there may be several matches for a fragment and hence possibly several distinct labellings; in this way, for example (JOHN1 WANTS2 (PRES (L X (X GIVES2 MARY1 BOOK3))) may be translated as either "John wants to give Mary the book" or "John wants to give the book to Mary" depending on whether the logical form matches rule 8, or rule 10 in the above grammar. See figure 4.1 for an example tree.



Rule 8 could have generated the sentence:

"John wants to give Mary the book" instead of the sentence generated with rule 10:

"John wants to give the book to Mary"

Figure 4.1 A sample phrase structure tree

To help choose the appropriate verbalization, pragmatic information such as the current topic or an objects salience could be used; or, one could always eliminate the non-determinism by making an arbitrary choice among possible verbalizations (as the meaning being conveyed would be the same), but this would eliminate any variability in the English sentences being produced and thus reduce any possibility of generating what could be called natural sounding discourse.

In summary, some strategies have been proposed for solving some of the problems that pop up in each stage of the generation process. In the filtration stage, to remove propositions known or obvious to the user one must have an adequate representation of the user. A prototypical user model represented as those propositions marked in the knowledge base as meaning postulates, as well as a record of the previous discourse which the user can be supposed to be aware, is proposed as sufficient for a general conversation system. In the organization stage, to place a partial ordering on response propositions, common sense rules that reflect natural tendencies for organizing discourse must be defined. In the assembly stage, one must find a way to generate the appropriate logical form(s) from a given set of clauses. The main difficulties in this stage were finding a way of grouping the response clauses into sentences, translating those sets of clauses into a logical form and generating appropriate descriptions of objects in the logical form. Finally, in the verbalization stage, it is proposed to use GPSG to translate the logical form into English. Some of the ideas presented could most effectively be tested by implementing them with the existing natural language understanding system. This chapter merely contains an outline of some of the problems that will be encountered when translating clause form to English. The solutions proposed are by no means complete; all stages require further work.

Chapter 5

Conclusion

The proposed and partially implemented system is different from most of the systems mentioned in the literature survey because it is the first based entirely on formal logic and takes advantage of methods long established in understanding natural language and applies them to generation. For instance, conversion of clause form to logical form, is seen as the reverse of turning logical form into conjunctive normal form, the first step in resolution theorem proving. To translate logical form into English, the same grammar may be used, GPSG, with translations mapping from logical form into English rather than the other way around. This approach has a uniformity and elegance because the same grammar and methods are used in understanding as well as generation.

No attempt has been made so far to develop what could be called a theory of natural language generation. The attempt to develop several well defined stages, as well as the expected input and output of each stage is seen as a necessary step towards the formalization of natural language generation. Each stage in the generation process obtains a meaning representation closer to English than its preceding form. Defining the intermediate levels of meaning representations: clause form and logical form, provides a framework for dividing up the work that must be done to generate English; each level of meaning representation requires different manipulations to obtain the next level.

It is expected that a system based on formal logic could also be used in other areas such as natural language interfaces to databases, since propositions in a database are very close to logical form.

A natural language generation system should be able to handle other types of questions, most notably wh-questions, in addition to handling yes-no questions. In this respect, the response generator I wrote was limited in comparison to other systems, such as Lehnert's QUALM, which could answer several types of wh-questions. It is expected that once the system has decided "what to say", the actual translation into English can proceed in the same manner for simple fill in the blank type of wh-questions, as for yes-no questions. Other types of wh-questions may require more sophisticated answer generation techniques such as McKeown's schemas for definitions, explanations and describing differences.

Also, since question answering is not the only type of communication between man and machine, a natural language generation system should be able to initiate dialogue as well as passively respond to it. Natural language planning systems[Allen 80, Appelt 82] are more advanced in this respect than my system. It is assumed that any system which can approach general conversational capabilities of humans should be able to take an active role in the dialogue when necessary. Thus any such system should be able to plan speech acts to satisfy general goals of discourse.

To summarize the directions for future research;

1. The first order resolution theorem prover should be extended to handle higher order logics and modal logic.
2. A natural language planning facility should be implemented, so that questions involving goals and plans could be answered, and the system could initiate dialogue as well as passively respond. Some thought should be given to extending the types of speech acts (requests, informs) allowed as operators in the planning systems mentioned in chapter two, to handle speech acts such as permissions, warnings, promises, etc.. A general conversation system should be capable of all different kinds of speech acts.
3. To determine the level of detail required in the user model. That is, determine whether a prototypical user model, with a record of previous discourse, is

sufficient for most general conversation skills and question answering, or whether a specific model of each user's knowledge and beliefs is required.

4. The assembly stage requires further work. A version of the Quine-McCluskey algorithm should be implemented to see if it is a practical means of determining the most compact representation for a logical form, or whether a simpler method, based on grouping related predicates together with the variables they have in common, would be a better approach.

5. A logic-to-English parser should be written and used as a basis for the implementation of the verbalization stage. This verbalization stage would also have a morphology routine to generate the appropriate tense and aspect of verbs.

6. To determine the effectiveness and practicality of the design suggested in the previous chapter, it should be implemented and tested with the existing natural language understanding system underdevelopment at the University of Alberta. This implementation should attempt to remove some of the arbitrary restrictions the program described in chapter 3 placed on its input.

The suggested design for a response generator constitues a possible starting point of a theory of response generation, and defines some of the intermediate meaning representations which must be generated on the way to generating English. I believe that this type of modular, theoretically well-founded design is necessary if one is ever to produce a system capable of engaging in human-like conversation.

References

[Allen 80] James F. Allen and C. Raymond Perrault, Analyzing intention in utterances, *Artificial Intelligence 15*, (1980), 143-178.

[Appelt 82]

Douglas E. Appelt, Planning Natural Language Utterances to Satisfy Multiple Goals, Technical Note 259, SRI International, Menlo Park, March 1982.

[Chester 76]

D. Chester, The translation of formal proofs into English, *Artificial Intelligence* 7, 3 (1976), 261-278.

[Cohen 78]Philip R. Cohen, On knowing what to say: planning speech acts, Technical Report No. 118, University of Toronto, January 1978.

[Conklin 82]

E. Jeffrey Conklin and David D. McDonald, Salience: the key to the selection problem in natural language generation, in *Proc. 20th Ann. Meeting of the Association for Computational Linguistics*, University of Toronto, 16-18 June 1982, 129-135.

[Covington&Schubert 80]

A. R. Covington and L. K. Schubert, Organization of modally embedded propositions and of dependent concepts, *Third National Conference of the CSCSI/SCEIO*, Victoria, BC, May 14-16, 1980, 87-94.

[Davey 78]Anthony Davey, *Discourse Production*, Edinburgh University Press, Edinburgh, 1978.

[DeHaan] John DeHaan, *Theorem Proving*, M.Sc. Thesis, University of Alberta, (in preparation).

[Dyer 83] Michael George Dyer, *In Depth Understanding*, MIT Press, Cambridge, Massachussetts, 1983.

[Gazdar,Klein,Pullum&Sag 85]

Gerald Gazdar, Ewan Klein, Geoffrey K. Pullum and Ivan A. Sag, *Generalized Phrase Structure Grammar*, Harvard University Press, Cambridge, Ma., 1985.

[Goldman 75]

Neil Goldman, Sentence paraphrasing from a conceptual base, *Comm. ACM 18*, 2 (February 1975), .

[Kohavi 70]

Zvi Kohavi, *Switching and Finite Automata Theory*, McGraw-Hill Book Company, New York, 1970.

[Lehnert 77]

Wendy Lehnert, The Process of Question Answering, Research Report #88, Yale University, May 1977.

[Lehnert 83]

Wendy G. Lehnert, Michael G. Dyer, Peter N. Johnson, C. J. Yang and Steve Harley, BORIS - an experiment in in-depth understanding of narratives, *Artificial Intelligence 20*, (1983), 15-62.

[Mann&Moore 81]

William C. Mann and James A. Moore, Computer generation of multiparagraph English text, *American Journal of Computational Linguistics 7*, 1 (1981), 17-29.

[Mann 81]William C. Mann, Madeline Bates, Barbara J. Grosz, David D. McDonald, Kathleen R. McKeown and William R. Swartout, Text Generation: The State of the Art and the Literature, ISI/RR-81-101, Information Sciences Institute, December 1981.

[Mann 83]William C. Mann, An Overview of the NIGEL text generation grammar, in *Proc. of the 21st Ann. Meeting of the Association for Computational Linguistics*, MIT, Cambridge, Massachussetts, 15-17 June 1983, 79-84.

[McDonald 83a]

David D. McDonald, Description Directed Control: Its Implications for Natural Language Generation, *Comp. & Maths. with Appls. 9*, 1 (1983), 111-129.

[McDonald 83b]

David D. McDonald, Natural language generation as a computational problem: an introduction, in *Computational Models of Discourse*, Michael Brady and Robert C. Berwick (ed.), MIT Press, Cambridge, Massachussetts, 1983.

[McKeown 82]

Kathleen R. McKeown, The TEXT system for natural language generation: An overview, in *Proc. of the 20th Ann. meeting of the Association for Computational Linguistics*, University of Toronto, 16-18 June 1982, 113-120.

[Patten 85]

Terry Patten, A problem solving approach to generating text from systemic grammars, in *Proc. of the European Conference of the Association for Computational Linguistics*, University of Geneva, Switzerland, 1985.

[Rich 83] Elaine Rich, *Artificial Intelligence*, McGraw-Hill Book Company, New York, 1983.

[Schank 75]

Roger C. Schank and Yale A.I. Project, SAM -- A Story Understander, 43, Yale University, 1975.

[Schubert 76]

L. K. Schubert, Extending the expressive power of semantic networks, *Artificial Intelligence 7*, (1976), 163-198.

[Schubert,Goebel&Cercone 79]

L. K. Schubert, R. G. Goebel and N. J. Cercone, The structure and organization of a semantic net for comprehension and inference, in *Associative Networks*, N. V. Findler (ed.), Academic Press, 1979, 121-175.

[Schubert&Pelletier 82]

Lenhart K. Schubert and Francis Jeffry Pelletier, From English to logic: context-free computation of 'conventional' logical translations, *American Journal of Computational Linguistics 8*, 1 (1982), 26-44.

[Simmons&Slocum 72]

Robert F. Simmons and J. Slocum, Generating English discourse from semantic networks, *Comm. ACM 15*, 10 (October 1972), 891-905.

[Tennant 81]

Harry Tennant, *Natural Language Processing*, PBI-Petrocelli Books, inc., New York, 1981.

[Wilensky 78]

Robert Wilensky, Understanding Goal-Based Stories, Research Report 140, Yale University, 1978.

[Winograd 71]

Terry Winograd, Procedures as a Representation for Data in a Program for Understanding natural language, AI Tech. Rep.-17, MIT Artificial Intelligence Laboratory, Cambridge, Massachussetts, February 1971.

Appendix A1

Sample Output

```
Franz Lisp, Opus 38.79

INPUT=(YES (((c HARD))))
Yes, something  is hard.
*********
cpu time used = 0.06666666666666667 seconds in translating directly to English
*********

Logicalform=((SOME THING) (PRES HARD))
*********
cpu time used = 0.06666666666666667 seconds in translating to logical form
*********


INPUT=(NO (((NOT (c HARD)))))
No, something  is not  hard.
*********
cpu time used = 0.06666666666666667 seconds in translating directly to English
*********

Logicalform=((SOME THING) (not (PRES HARD)))
*********
cpu time used = 0.05 seconds in translating to logical form
*********


INPUT=(YES (((c HARD)) ((c ROCK))))
Yes, there  is a  hard rock.
*********
cpu time used = 0.08333333333333333 seconds in translating directly to English
*********

Logicalform=((SOME ROCK) (PRES HARD))
*********
cpu time used = 0.06666666666666667 seconds in translating to logical form
*********


INPUT=(YES (((Mary PRETTY))))
Yes, Mary is pretty.
*********
cpu time used = 0.06666666666666667 seconds in translating directly to English
*********

Logicalform=(Mary (PRES PRETTY))
*********
cpu time used = 0.05 seconds in translating to logical form
*********
```

```
INPUT=(NO (((NOT (Mary PRETTY)))))
No. Mary is not  pretty.
*********

cpu time used  =  0.05 seconds in translating directly to English
*********

Logicalform=(Mary (not (PRES PRETTY)))
*********

cpu time used  =  0.06666666666666667 seconds in translating to logical form
*********


INPUT=(YES (((Mary PRETTY)) ((Mary GIRL))))
Yes, Mary is a pretty girl.
*********

cpu time used  =  0.1 seconds in translating directly to English
*********

Logicalform=(Mary (PRES (L x ((x PRETTY) (x GIRL)))))
*********

cpu time used  =  0.7333333333333333 seconds in translating to logical form
*********


INPUT=(NO (((NOT (Mary PRETTY))) ((Mary GIRL))))
No, Mary is not  a pretty girl.
*********

cpu time used  =  0.1 seconds in translating directly to English
*********

Logicalform=(Mary (not (PRES (L x ((x PRETTY) (x GIRL))))))
*********

cpu time used  =  0.1 seconds in translating to logical form
*********


INPUT=(YES (((John LOVES Mary))))
Yes, John loves Mary.
*********

cpu time used  =  0.06666666666666667 seconds in translating directly to English
*********

Logicalform=(John (PRES (LOVES Mary)))
*********

cpu time used  =  0.05 seconds in translating to logical form
*********


INPUT=(NO (((NOT (John LOVES Mary)))))
No, John does not love  Mary.
*********

cpu time used  =  0.05 seconds in translating directly to English
```

```
*********

Logicalform=(John (not (PRES (LOVES Mary))))
*********
cpu time used = 0.05 seconds in translating to logical form
*********


INPUT=(YES (((John LOVES c))))
Yes, John loves something.
*********
cpu time used = 0.08333333333333333 seconds in translating directly to English
*********

Logicalform=(John (PRES (LOVES (SOME THING))))
*********
cpu time used = 0.05 seconds in translating to logical form
*********


INPUT=(NO (((NOT (John LOVES c)))))
No, John does not love  something.
*********
cpu time used = 0.06666666666666667 seconds in translating directly to English
*********

Logicalform=(John (not (PRES (LOVES (SOME THING)))))
*********
cpu time used = 0.08333333333333333 seconds in translating to logical form
*********


INPUT=(YES (((John LOVES c)) ((c PRETTY))))
Yes, John loves a pretty thing.
*********
cpu time used = 0.1166666666666667 seconds in translating directly to English
*********

Logicalform=(John (PRES (LOVES (SOME (L x ((x PRETTY) (x THING)))))))
*********
cpu time used = 0.08333333333333333 seconds in translating to logical form
*********


INPUT=(NO (((NOT (John LOVES c))) ((c UGLY))))
No, John does not love  an ugly thing.
*********
cpu time used = 0.1 seconds in translating directly to English
*********

Logicalform=(John (not (PRES (LOVES (SOME (L x ((x UGLY) (x THING))))))))
*********
cpu time used = 0.08333333333333333 seconds in translating to logical form
```

```
*********

INPUT=(YES (((John HAS c)) ((c GRAND)) ((c PIANO))))
Yes, John has a grand piano.
*********
cpu time used = 0.1166666666666667 seconds in translating directly to English
*********

Logicalform=(John (PRES (HAS (SOME (L x ((x GRAND) (x PIANO)))))))
*********
cpu time used = 0.1 seconds in translating to logical form
*********


INPUT=(NO (((NOT (John HAS c))) ((c PIANO)) ((c GRAND))))
No, John does not have a grand piano.
*********
cpu time used = 0.1166666666666667 seconds in translating directly to English
*********

Logicalform=(John (not (PRES (HAS (SOME (L x ((x GRAND) (x PIANO))))))))
*********
cpu time used = 0.08333333333333333 seconds in translating to logical form
*********


INPUT=(YES (((c LOVES John)) ((c PRETTY)) ((c GIRL))))
Yes, a pretty girl loves John.
*********
cpu time used = 0.1333333333333333 seconds in translating directly to English
*********

Logicalform=((SOME (L x ((x PRETTY) (x GIRL)))) (PRES (LOVES John)))
*********
cpu time used = 0.1 seconds in translating to logical form
*********


INPUT=(NO (((NOT (c LOVES John))) ((c PRETTY)) ((c GIRL))))
No, a pretty girl does not love John.
*********
cpu time used = 0.8 seconds in translating directly to English
*********

Logicalform=((SOME (L x ((x PRETTY) (x GIRL)))) (not (PRES (LOVES John))))
*********
cpu time used = 0.1 seconds in translating to logical form
*********


INPUT=(YES (((c ATE GM))))
Yes, something ate grandmother.
```

```
*********
cpu time used = 0.06666666666666667 seconds in translating direttly to English
*********

Logicalform=((SOME THING) (PRES (ATE GM)))
*********
cpu time used = 0.05 seconds in translating to logical form
*********


INPUT=(NO (((NOT (c ATE GM)))))
No, something did not eat grandmother.
*********
cpu time used = 0.06666666666666667 seconds in translating directly to English
*********

Logicalform=((SOME THING) (not (PRES (ATE GM))))
*********
cpu time used = 0.06666666666666667 seconds in translating to logical form
*********


INPUT=(YES (((c ATE GM)) ((c HUNGRY))))
Yes, a hungry thing ate grandmother.
*********
cpu time used = 0.1166666666666667 seconds in translating directly to English
*********

Logicalform=((SOME (L x ((x HUNGRY) (x THING)))) (PRES (ATE GM)))
*********
cpu time used = 0.06666666666666667 seconds in translating to logical form
*********


INPUT=(YES (((c ATE GM)) ((c HUNGRY)) ((c WOLF))))
Yes, a hungry wolf ate grandmother.
*********
cpu time used = 0.1166666666666667 seconds in translating directly to English
*********

Logicalform=((SOME (L x ((x HUNGRY) (x WOLF)))) (PRES (ATE GM)))
*********
cpu time used = 0.08333333333333333 seconds in translating to logical form
*********


INPUT=(YES (((W ATE GM))))
Yes, the wolf ate grandmother.
*********
cpu time used = 0.08333333333333333 seconds in translating directly to English
*********

Logicalform=(W (PRES (ATE GM)))
```

```
*********
cpu time used = 0.05 seconds in translating to logical form
*********


INPUT=(NO (((NOT (c ATE GM))) ((c HUNGRY)) ((c PERSON))))
No. a hungry person did not eat grandmother.
*********
cpu time used = 0.1166666666666667 seconds in translating directly to English
*********

Logicalform=((SOME (L x ((x HUNGRY) (x PERSON)))) (not (PRES (ATE GM))))
*********
cpu time used = 0.0833333333333333 seconds in translating to logical form
*********


INPUT=(YES (((T IN BAS))))
Yes, the tart is in the basket.
*********
cpu time used = 0.06666666666666667 seconds in translating directly to English
*********

Logicalform=(T (PRES (IN BAS)))
*********
cpu time used = 0.05 seconds in translating to logical form
*********


INPUT=(NO (((NOT (T IN BAS)))))
No, the tart is not in the basket.
*********
cpu time used = 0.06666666666666667 seconds in translating directly to English
*********

Logicalform=(T (not (PRES (IN BAS))))
*********
cpu time used = 0.05 seconds in translating to logical form
*********


INPUT=(YES (((H IN b)) ((b BIG)) ((b FOREST))))
Yes, the hunter is in a big forest.
*********
cpu time used = 0.1333333333333333 seconds in translating directly to English
*********

Logicalform=(H (PRES (IN (SOME (L x ((x BIG) (x FOREST)))))))
*********
cpu time used = 0.0833333333333333 seconds in translating to logical form
*********
```

```
INPUT=(NO (((NOT (H IN b))) ((b BIG)) ((b RED)) ((b HOUSE))))
No, the hunter is not in  a big red house.
*********

cpu time used = 0.15 seconds in translating directly to English
*********

Logicalform=(H (not (PRES (IN (SOME (L x ((x BIG) (x RED) (x HOUSE))))))))
*********

cpu time used = 0.08333333333333333 seconds in translating to logical form
*********


INPUT=(YES (((b IN HOUSE))))
Yes, something is in house.
*********

cpu time used = 0.06666666666666667 seconds in translating directly to English
*********

Logicalform=((SOME THING) (PRES (IN HOUSE)))
*********

cpu time used = 0.03333333333333333 seconds in translating to logical form
*********


INPUT=(NO (((NOT (b IN HOUSE)))))
No, something is not in house.
*********

cpu time used = 0.06666666666666667 seconds in translating directly to English
*********

Logicalform=((SOME THING) (not (PRES (IN HOUSE))))
*********

cpu time used = 0.05 seconds in translating to logical form
*********


INPUT=(YES (((John IN c))))
Yes, John is in something.
*********

cpu time used = 0.06666666666666667 seconds in translating directly to English
*********

Logicalform=(John (PRES (IN (SOME THING))))
*********

cpu time used = 0.05 seconds in translating to logical form
*********


INPUT=(NO (((NOT (John IN c)))))
No, John is not in  something.
*********

cpu time used = 0.08333333333333333 seconds in translating directly to English
*********
```

```
Logicalform=(John (not (PRES (IN (SOME THING)))))
*********
cpu time used = 0.05 seconds in translating to logical form
*********


INPUT=(YES (((John LEFT))))
Yes, John left.
*********
cpu time used = 0.05 seconds in translating directly to English
*********

Logicalform=(John (PRES LEFT))
*********
cpu time used = 0.05 seconds in translating to logical form
*********


INPUT=(NO (((NOT (John LEFT)))))
No, John did not leave .
*********
cpu time used = 0.05 seconds in translating directly to English
*********

Logicalform=(John (not (PRES LEFT)))
*********
cpu time used = 0.06666666666666667 seconds in translating to logical form
*********


INPUT=(YES (((LR LEFT))))
Yes, little red riding hood left.
*********
cpu time used = 0.05 seconds in translating directly to English
*********

Logicalform=(LR (PRES LEFT))
*********
cpu time used = 0.05 seconds in translating to logical form
*********


INPUT=(NO (((NOT (LR LEFT)))))
No, little red riding hood did not leave .
*********
cpu time used = 0.05 seconds in translating directly to English
*********

Logicalform=(LR (not (PRES LEFT)))
*********
cpu time used = 0.05 seconds in translating to logical form
*********
```

```
INPUT=(YES (((c LEFT))))
Yes, something left.
*********

cpu time used = 0.05 seconds in translating directly to English
*********

Logicalform=((SOME THING) (PRES LEFT))
*********

cpu time used = 0.05 seconds in translating to logical form
*********


INPUT=(NO (((NOT (c LEFT)))))
No, something did not leave .
*********

cpu time used = 0.05 seconds in translating directly to English
*********

Logicalform=((SOME THING) (not (PRES LEFT)))
*********

cpu time used = 0.05 seconds in translating to logical form
*********


INPUT=(YES (((X HARD))))
Yes, everything is hard.
*********

cpu time used = 0.06666666666666667 seconds in translating directly to English
*********

Logicalform=((EVERY THING) (PRES HARD))
*********

cpu time used = 0.06666666666666667 seconds in translating to logical form
*********


INPUT=(NO (((NOT (X HARD)))))
No, nothing is hard.
*********

cpu time used = 0.05 seconds in translating directly to English
*********

Logicalform=((EVERY THING) (not (PRES HARD)))
*********

cpu time used = 0.06666666666666667 seconds in translating to logical form
*********


INPUT=(YES (((John LOVES X))))
Yes, John loves everything.
*********
```

```
cpu time used = 0.06666666666666667 seconds in translating directly to English
*********

Logicalform=(John (PRES (LOVES (EVERY THING))))
*********
cpu time used = 0.05 seconds in translating to logical form
*********


INPUT=(NO (((NOT (John LOVES X)))))
No, John loves nothing.
*********
cpu time used = 0.05 seconds in translating directly to English
*********

Logicalform=(John (PRES (LOVES (NO THING))))
*********
cpu time used = 0.05 seconds in translating to logical form
*********


INPUT=(YES (((John LOVES X) (NOT (X PRETTY)))))
Yes, John loves everything pretty.
*********
cpu time used = 0.06666666666666667 seconds in translating directly to English
*********

Logicalform=(John (PRES (LOVES (EVERY (L x ((x PRETTY) (x THING)))))))
*********
cpu time used = 0.06666666666666667 seconds in translating to logical form
*********


INPUT=(NO (((NOT (John LOVES X)) (NOT (X PRETTY)))))
No, John loves nothing pretty.
*********
cpu time used = 0.08333333333333333 seconds in translating directly to English
*********

Logicalform=(John (PRES (LOVES (NO (L x ((x PRETTY) (x THING)))))))
*********
cpu time used = 0.08333333333333333 seconds in translating to logical form
*********


INPUT=(YES (((John LOVES X) (NOT (X GIRL)))))
Yes, John loves every girl.
*********
cpu time used = 0.08333333333333333 seconds in translating directly to English
*********

Logicalform=(John (PRES (LOVES (EVERY GIRL))))
*********
```

```
cpu time used = 0.08333333333333333 seconds in translating to logical form
*********


INPUT=(NO (((NOT (John LOVES X)) (NOT (X GIRL)))))
No, John loves no girl.
*********
cpu time used = 0.1 seconds in translating directly to English
*********

Logicalform=(John (PRES (LOVES (NO GIRL))))
*********
cpu time used = 0.06666666666666667 seconds in translating to logical form
*********


INPUT=(YES (((X LOVES John) (NOT (X GIRL)))))
Yes, every girl loves John.
*********
cpu time used = 0.08333333333333333 seconds in translating directly to English
*********

Logicalform=((EVERY GIRL) (PRES (LOVES John)))
*********
cpu time used = 0.08333333333333333 seconds in translating to logical form
*********


INPUT=(NO (((NOT (X LOVES John)) (NOT (X GIRL)))))
No, no girl loves John.
*********
cpu time used = 0.08333333333333333 seconds in translating directly to English
*********

Logicalform=((NO GIRL) (PRES (LOVES John)))
*********
cpu time used = 0.06666666666666667 seconds in translating to logical form
*********


INPUT=(YES (((X SCARES John))))
Yes, everything scares John.
*********
cpu time used = 0.08333333333333333 seconds in translating directly to English
*********

Logicalform=((EVERY THING) (PRES (SCARES John)))
*********
cpu time used = 0.05 seconds in translating to logical form
*********


INPUT=(NO (((NOT (X SCARES John)))))
```

```
No, nothing scares John.
*********

cpu time used = 0.06666666666666667 seconds in translating directly to English
*********

Logicalform=((NO THING) (PRES (SCARES John)))
*********

cpu time used = 0.05 seconds in translating to logical form
*********


INPUT=(YES (((X IN BAS))))
Yes, everything is in the basket.
*********

cpu time used = 0.08333333333333333 seconds in translating directly to English
*********

Logicalform=((EVERY THING) (PRES (IN BAS)))
*********

cpu time used = 0.05 seconds in translating to logical form
*********


INPUT=(NO (((NOT (X IN BAS)))))
No, nothing is in the basket.
*********

cpu time used = 0.06666666666666667 seconds in translating directly to English
*********

Logicalform=((NO THING) (PRES (IN BAS)))
*********

cpu time used = 0.05 seconds in translating to logical form
*********


INPUT=(YES (((X TART) (NOT (X IN BAS)))))
Yes, every tart is in the basket.
*********

cpu time used = 0.1 seconds in translating directly to English
*********

Logicalform=((EVERY TART) (PRES (IN BAS)))
*********

cpu time used = 0.06666666666666667 seconds in translating to logical form
*********


INPUT=(NO (((NOT (X TART)) (NOT (X IN BAS)))))
No, no tart is in the basket.
*********

cpu time used = 0.08333333333333333 seconds in translating directly to English
*********
```

```
Logicalform=((NO TART) (PRES (IN BAS)))
*********
cpu time used = 0.75 seconds in translating to logical form
*********


INPUT=(YES (((X LEFT) (NOT (X RED)))))
Yes, everything red left
*********
cpu time used = 0.06666666666666667 seconds in translating directly to English
*********

Logicalform=((EVERY (L x ((x RED) (x THING)))) (PRES LEFT))
*********
cpu time used = 0.08333333333333333 seconds in translating to logical form
*********


INPUT=(NO (((NOT (X LEFT)) (NOT (X RED)))))
No, everything red did not leave .
*********
cpu time used = 0.06666666666666667 seconds in translating directly to English
*********

Logicalform=((EVERY (L x ((x RED) (x THING)))) (not (PRES LEFT)))
*********
cpu time used = 0.06666666666666667 seconds in translating to logical form
*********


INPUT=(YES (((X LEFT) (NOT (X GIRL)))))
Yes, every girl left.
*********
cpu time used = 0.06666666666666667 seconds in translating directly to English
*********

Logicalform=((EVERY GIRL) (PRES LEFT))
*********
cpu time used = 0.05 seconds in translating to logical form
*********


INPUT=(NO (((NOT (X LEFT)) (NOT (X GIRL)))))
No, every girl did not leave .
*********
cpu time used = 0.08333333333333333 seconds in translating directly to English
*********

Logicalform=((EVERY GIRL) (not (PRES LEFT)))
*********
cpu time used = 0.06666666666666667 seconds in translating to logical form
*********
```

```
INPUT=(YES (((X LEFT))))
Yes, everything left.
*********
cpu time used = 0.03333333333333333 seconds in translating directly to English
*********

Logicalform=((EVERY THING) (PRES LEFT))
*********
cpu time used = 0.05 seconds in translating to logical form
*********


INPUT=(NO (((NOT (X LEFT)))))
No, everything did not leave .
*********
cpu time used = 0.06666666666666667 seconds in translating directly to English
*********

Logicalform=((EVERY THING) (not (PRES LEFT)))
*********
cpu time used = 0.05 seconds in translating to logical form
*********

->
Goodbye
```