A PROLOG Lexical Phrase Computer Assisted Language Learning Module

Yuji Gene Hirayama
Portland State University

THESIS APPROVAL

The abstract and thesis of Yuji Gene Hirayama for the Master of Arts in TESOL were presented February 15, 1996, and accepted by the thesis committee and the department.

COMMITTEE APPROVALS: [redacted]

Beatrice Oshika, Chair

[redacted]

Jeanette S. DeCarrico

[redacted]

Suwako Watanabe
Representative of the Office of Graduate Studies

DEPARTMENTAL APPROVAL: [redacted]

Beatrice Oshika, Chair
Department of Applied Linguistics

* *

ACCEPTED FOR PORTLAND STATE UNIVERSITY BY THE LIBRARY

by [redacted] on 2 April 1996

ABSTRACT

An abstract of the thesis of Yuji Gene Hirayama for the Master of Arts in TESOL presented February, 15, 1996.

Title: A PROLOG Lexical Phrase Computer Assisted Language Learning Module

This thesis presents the design of a computer assisted language learning (CALL) program written in the computer language, PROLOG. It will provide a practice exercise to teach “lexical phrases” to second language learning students of English. Lexical phrases are lexico-grammatical "chunks" of words, which possess specific pragmatic functions within spoken discourse (Nattinger and DeCarrico, 1992). These form/function composites of varying length aid conversational fluency.

The program presents a scenario where the participants are college students who pass one another in the hallway. After they exchange initial greetings, the first participant (i.e., the computer) informs the other student that a test, for some unknown class, will be postponed till “next Tuesday”. The second student answers with a surprised remark. The first student responds that he/she could use the extra time for study, and then says, “good-bye”. The dialog ends with when the second participant types in “good-bye”.

The study sought to answer the following questions: 1) are lexical phrases adaptable for use in computer assisted language learning (CALL) programs?; 2) what problems arise when using lexical phrases on computers?; 3) is the dialog realistic and does it offer a communicative alternative to traditional drills?

The results are that lexical phrases can be easily implemented in computer assisted language learning (CALL) programs. Further, CALL programs using lexical phrases in a communicative language teaching mode provide a framework for realistic dialogs. It offers more interesting exercises compared to traditional language drills.

The only criteria for the computer-created dialog is its ability to produce realistic responses. This program produces a realistic dialog, although it is highly invariable. A major drawback to this study is its inability to implement any parsing capabilities into the program; thus, there are restrictions on the database representation of any contextual information.

Nevertheless, as computer software technology advances, the use of lexical phrases in CALL programs will provide an effective means to aid the communicative competence of second language learners of English.

A PROLOG LEXICAL PHRASE COMPUTER ASSISTED LANGUAGE LEARNING MODULE

by

YUJI GENE HIRAYAMA

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF ARTS
in
TESOL

Portland State University
1996

ACKNOWLEDGEMENTS

I would like to express my deep appreciation and gratitude to Dr. Beatrice Oshika for her guidance and her patience during my long thesis-writing process. I would also like to thank Dr. Jeanette DeCarrico for her clarifying the principles underlying lexical phrase theory.

Finally, this thesis is dedicated to the memory of Dr. James R. Nattinger, who with Dr. Jeanette DeCarrico, wrote “Lexical Phrases and Language Teaching”.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1. Statement of problem

With the rapid advances in computer technology, language educators are incorporating and developing software programs to assist students in their classes. There are two significant problems associated with language software that have resulted in relatively mixed showings. First, few theories exist to aid students in language competency. Second, computers cannot yet process spoken discourse at the human linguistic levels. The software complexity in developing intelligent human-machine interfaces is significant.

A valid theory of discourse is necessary to aid students in language competency. A dialogue is a sequence of conversation fragments, in which the participants share common concepts, ideas and presuppositions. Each fragment is a conversational move that has a particular function in the discourse (Reichman, 1985). Nattinger and DeCarrico (Nattinger, 1980; Nattinger and DeCarrico, 1992) posit that humans acquire and facilitate language functions through the use of lexical phrases. They define lexical phrases as form/function composites that fall on a continuum between the lexicon and syntax (Nattinger and DeCarrico, 1992, p. 36). These chunks of words, which are relatively frozen in form, are often "called up" by the speaker to aid in conversational fluency. They propose that language students learn lexical phrases to increase their communicative competence as well as performance. This hypothesis has yet to be implemented on a computer via a computer assisted-language learning (CALL) module.

The objective of the thesis is to create a short, simple, and somewhat realistic dialog with a human host and a computer by using lexical phrases for language generation. The vehicle to accomplish this task will be a simple computer assisted language learning (CALL) software module that will teach a lexical phrase to second language (L2) learners. The program will be a frame-based natural language processing (NLP) application written in PROLOG, a declarative computer language used in artificial intelligence (AI) applications.

1.2. Lexical phrases

Language researchers theorize that humans possess the ability to process language because of three innate factors: the ability to match patterns, the ability to model the world, and the ability to manipulate the environment to represent meaning (McClelland, 1989). One area of study is the role of language patterns in the generation and the processing of linguistic input. Language acquisition studies note that first language gestalt learners tend to reproduce whole segments as single utterances (Dore, 1975; Peters, 1977). Likewise, second language acquisition (L2) studies note that, at early stages of study, L2 learners use prepatterned, conventionalized forms of language to aid in communicative competence (Hakuta, 1974; Huang and Hatch, 1975; Yorio, 1980, Gatbonten and Segalowitz,1988). Peters (1983) states:

> A mature language user may find that certain expressions or variations on expressions are so useful that it would be convenient, as a device for conserving processing time and effort, to be able to retrieve them in as prefabricated a form as possible. Such prefabrications could be in either of two forms: fused and invariant units, and well-rehearsed (automatized) patterns that require minimum of processing (e.g., in the form of insertion of lexical items into a slot) in order to produce the desired utterance (p. 100).

Lexical phrases are lexico-grammatical "chunks" of words, which possess specific pragmatic functions within discourse (Nattinger and DeCarrico, 1992). These form/function composites of varying length aid conversational fluency by guaranteeing fast retrieval of phrases. Lexical phrase theory supports a "top-down processing" model of language. That is, speakers can focus on the larger structure of discourse, rather than on the individual words of each sentence.

Lexical phrases fall on a continuum between the lexicon and syntax (Nattinger and DeCarrico, 1992). At one end of the continuum (i.e., the lexicon side) are idioms and clichés, which are chunks of language that are relatively frozen in form. Idioms have meanings that can be derived from the phrase as a whole, such as "*raining cats and dogs*," "*step on the gas*," etc. Clichés, however, are larger phrases whose its meaning can be derived from the individual words, such as "*no doubt about it*," "*give 'em credit*," "*have a nice day,*" etc. Collocations are strings of lexical items that in some semantic and pragmatic way go together, such as "*false teeth,*" "*curry favor,*" etc. Collocations fall in the center of the lexicon/syntax continuum by also including idioms and clichés. At the end of the continuum (i.e., the syntax side) are lexical phrases. The difference between idioms/clichés and lexical phrases is that the former phrases perform no particular pragmatic function in discourse, whereas the latter phrases do. Lexical phrases are collocations that have specific discourse and pragmatic functions.

There are four types of lexical phrases: polywords, institutionalized expressions, phrasal constraints, and sentence builders (Nattinger and DeCarrico, 1992, p. 37). Polywords are short phrases that are non-variable and function like individual lexical items, such as canonical forms, "*for the most part,*" or "*in a nutshell.*" Noncanonical forms include, "*as it were,*" "*by and large.*"

Institutionalized expressions are proverbs, aphorisms, and formulas for social interaction that are used for quotation, allusion or direct use. Examples include "*how are you?*", "*long time no see.*"

Phrasal constraints are short-to-medium-length phrases that allow variation in lexical and phrasal categories. Like polywords, they perform a variety of functions, such as "*a ______ ago* ("*a long time ago*"/"*a while ago*") ", or "*as far as I* _____" ("*as far as I know*"/ "*as far as I am concerned*").

Sentence builders are lexical phrases that provide the framework for whole sentences. They contain slots or fillers that provide arguments for expressing an entire idea. "*I think (that)* I will go to the store," "*my point is that* TIME IS MONEY," "*MODAL + you + VP (for me)* (Could you open the door for me?)" Nattinger and DeCarrico (1992) believe that lexical phrases are part of the natural mechanism that aid in language generation.

	Sentence builders	Phrasal constraints	Institutionalized expressions	Polywords
Grammatical Level:	sentence	word	sentence	word
Canonical/ Noncanonical:	canonical	both	canonical	both
Variable/Fixed:	highly variable	somewhat variable	fixed	fixed

Table 1. Lexical phrase characteristics (Nattinger & DeCarrico, 1992, p. 45).

1.3. Natural language processing

Natural language processing (NLP) is a hybrid of two academic disciplines: artificial intelligence (AI) and linguistics (Grishman, R. 1986). AI researchers construct models for human intelligence and determine knowledge representation schemes (Obermeier; 1989). Linguists construct models of grammar formalisms.

NLP researchers have a threefold objective in designing their systems. First, they must adapt general linguistic theories for specific objectives, such as human-machine interfaces, machine translations systems, etc. Second, they must consider in their designs the degree of processing sophistication as well as the constraints involved in automated language generation. Third, they must consider other important design factors, such as system architecture, lexicon structures, and knowledge representations, which affect the semantic processing of a natural language.

Weizenbaum (1966), a pioneer of artificial intelligence (AI) research, simulated natural language discourse in his ELIZA program. The program's objective was to determine the appropriate modality of interaction between a human and a computer. Another objective was to demonstrate the potential emptiness of a system, which might appear like it “knows” a language. The discourse model was based upon the responses of a Rogerian psychotherapist to a patient undergoing a preliminary interview. Weizenbaum chose this mode of interaction for three reasons: it involved mimicking an actual psychiatric technique; the style of interaction helps lead the topic of discussion; and this form of dyadic communication enables one speaker (i.e., ELIZA) to assume the position of knowing nothing about the real world.

In actual human conversation, the listener will make initial assumptions about his/her partner prior to the dialogue (Weizenbaum, 1966). As long as the partner's responses are consistent with the listener's assumptions, the speaker's credibility of his/her partner remains "undamaged". When the responses become self-contradictory or incoherent, then the image of the conversational partner crumbles, and a negative image emerges. In human-machine communication, the partner's conversational "plausibility" is the key factor, not "credibility" as in human-human discourse (Weizenbaum, 1966). The use of conventionalized language structures in ELIZA gave evidence that prepatterned forms play a role in human natural language processing.

Technically, there are three main features of the ELIZA program. First, it has built-in questions, which could be posed in response to certain triggering words. Second, the program searches for particular keywords that triggers appropriate responses. Third, ELIZA generates non-committal sentences as responses when information is insufficient or when there are phrases that cannot be handled by the program. The program generated dialog as follows (Weizenbaum, 1966, p. 37):

1	H(uman):	Men are all alike.
2	E(LIZA):	IN WHAT WAY
3	H:	They're always bugging us about something or other.
4	E:	CAN YOU THINK OF A SPECIFIC EXAMPLE
5	H:	Well, my boyfriend made me come here.
6	E:	YOUR BOYFRIEND MADE YOU COME HERE
7	H:	He says I'm depressed much of the time.
8	E:	I AM SORRY TO HEAR YOU ARE DEPRESSED
9	H:	It's true. I am unhappy.
10	E:	DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY.
11	H:	I need some help, that much seems certain.
12	E:	WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP
13	H:	Perhaps I could learn to get along with my mother.
14	E:	TELL ME MORE ABOUT YOUR FAMILY ...

The interesting aspect of ELIZA is its use of "lexical phrases". The program uses templates, called decomposition rules, to decompose the sentence into various parts. The computer only recognizes those words that matched the template. Reassembly rules are then applied to form a response. For example, the sentence, "it seems that you hate me" can be decomposed and reassembled as follows (Weizenbaum, 1966, p. 39):

sentence:	It seems that you hate me.
decomposition rule:	X you Y me
	;where X and Y are an indefinite number of words
decomposed sentence:	(1) it seems that (2) you (3) hate (4) me
reassembly rule:	WHAT MAKES YOU THINK I (3) YOU
reassembled sentence:	WHAT MAKES YOU THINK I HATE YOU

ELIZA applies the decomposition rule to the input sentence and recognizes the keyword "you" and "me." The words preceding "you" are in group one, and the word following "you" is in group three. Next, ELIZA applies the reassembly rule to the decomposed sentence. Group three (i.e., in this case, one word) fills the empty slot of the sentence. Thus, ELIZA uses "sentence builder" lexical phrases for its language generation.

The program possesses two other features that creates realistic responses: transformation rules and lists. First, the program uses transformation rules that change the case of each pronoun (e.g. from "me" to "you") where necessary. Second, lists hold keywords and their associated rules. A system of ranking prioritizes the keyword in each text scan to form its response.

The program has two significant problems. One problem is the key-matching approach. ELIZA can "mimic" fluent conversation by using the pre-patterned phrases. However, it cannot extract any semantic information from the responses without parsing the input sentences. The test for understanding is not the ability to continue a conversation, but to draw conclusions from the content of the discussion (Weizenbaum, 1985). The second problem is that ELIZA's responses are not coherent in the overall dialog. After several conversational moves, ELIZA's responses would become repetitive. Weizenbaum (1985) outlines the difficulty with this program:

> ELIZA was a program consisting mainly of general methods for analyzing sentences and sentence fragments, locating so called key words in texts, assembling sentences from fragments, and so on. It had, in other words, no built-in contextual framework or universe of discourse. This was supplied to it by a 'script'. In a sense ELIZA was an actress who commanded a set of techniques but who had nothing of her own to way. The script, in turn, was a set of rules which permitted the actor to improvise on whatever resources it provided. (p. 120).

1.4. Computer assisted language learning (CALL)

Computer assisted language learning (CALL) involves the use of computers to teach language skills as either a self-managed system or as an auxiliary role to the language teacher. The debate centers on what the exact role of the computer should be within the language classroom, and by what method of language pedagogy the computer should be used in curriculums.

The four basic models of language learning are: the behaviorist model, the cognitive-code model, the communicative model, and the humanist model. The audiolingual approach, based on a behaviorist theory of learning, claims that language is learned through the acquisition of stimulus and response associations. The prime example of this model is the use of language drills. The teacher presents a sequence of teaching examples to the student. Through the use of repetitive drills, the student acquires a language skill or behavior. Errors are deemed detrimental to learning, since they prevent proper habit-formation. Thus, mechanical practice and the early elimination of any errors are the most important factors in this method.

The audiolingual model has had a strong influence on CALL programs in the 1980s for two reasons. The first reason is due to the historical connections to the "language lab". The language lab showed great promise during the 1960s and 1970s, but ultimately led to its own demise, because of its many limitations in language teaching. These limitations are that: 1) the student must correct himself or herself; 2) all responses are prefabricated; thus, there is no actual semblance to communication; 3) the student is a passive participant, rather than an active one; and 4) the program is inflexible.

The second reason is due to the nature of the computer itself. Early computer technology could only deal with natural language in a highly restricted form. It lends itself well to exercises, where the language is controlled and the range of possible responses for the student is limited. The main advantage of the computer over the language lab is that the computer could give instant feedback and analysis on any mistakes that are made during the drills.

The use of feedback led to the computer implementation of the cognitive code model of teaching. The cognitive code model is designed to give the student conscious knowledge of language rules. The student could apply these rules for later use. While the audiolingual model emphasizes mechanical, repetitive practice, the cognitive code model emphasizes careful attention to the rules of grammar. Conscious understanding of grammatical rules is the most important element in this learning mode.

The communicative model is based on the concept that people learn how to communicate in a target language through active spoken discourse. The main objective is to enable the students to experiment and to explore in the target language, so that they can grasp the forms needed to communicate (Cook, 1987).

The humanistic model is based on the requirement that the process of language learning will ultimately lead to the individual's own development. It should respond to the needs of each student in a truly student-centered program. This approach requires more research in terms of implementation on the computer.

Underwood (1984) outlines some points for a communicative approach to CALL software. He states that the program should:

1) aim at acquisition, not learned practice; thus no drills.
2) have grammar lessons that are implicitly taught, not explicitly taught.
3) encourage the student to generate original utterances rather than manipulate prefabricated language.
4) not judge or evaluate everything the student does.
5) avoid telling students that they are wrong, i.e., avoid wrong-try-again approach.
6) not reward students with congratulatory messages.
7) not be cute.
8) use target language exclusively.
9) be flexible, i.e., multiple responses.

10) allow the student to explore subject matter.
11) create an environment in which using the target language feels natural, both on and off screen.
12) never try to do anything that a book could do as well, i.e., no electronic page-turners.
13) be fun with no exams or quizzes.

The communicative model has more potential for success in CALL applications than the audiolingual and the cognitive code models, since many educators employ this method in their classrooms.

Major CALL programs have been developed at major universities, but the majority of exercises were drills based on the audiolingual model. Universities adopted the audiolingual method due to the limitations of early software technology and to the general inability of language teachers to program computers. With technological advances and the use of artificial intelligence (AI) methodology, communicative CALL software is producing promising results (Cook, 1988; Weischedel, Voge, and James, 1978; Last, 1989).

The objective of this study is to create a computer program using a communicative CALL model to teach lexical phrases to second language (L2) learners.

1.5. Research Questions

1. Are lexical phrases adaptable for use in computer assisted language learning (CALL) programs?

2. What problems arise when using lexical phrases on computers?

3. Is the dialog realistic and does it offer a communicative alternative to traditional drills?

1.6. Limitations

The dialogue will be short to curtail problems that occur in NLP interfaces (i.e., indirect answers, anaphoric reference over sentence boundaries, sentence fragments, conversational patterns). The rationale for a constrained context can be summed up by Ringle and Bruce (1982):

> A major obstacle to a free-form, user-initiated dialogue system lies in the infinite variability of belief-models that users bring to a conversation. One approach to this problem is to explore a domain in sufficient depth to allow for a comprehensive description of all probable goals and beliefs (including erroneous beliefs) that a user might have in a given situation. By anticipating all of the relevant goal and belief states, the system is able to accommodate a wide range of user-initiated utterances (or written inputs). The dimension of user sensitivity is reduced, in essence, to a parsing problem --that is, the problem of mapping surface strings to a small, hierarchical goal tree (p.215)

Issues on resolving anaphoric reference over sentence boundaries, indirect speech acts, and fragments are beyond the scope of this study. Solutions could not be addressed due to the complexity of each problem.

Further, a reliable parser was not available for this project. Parsing enables programs to extract structural information from a large set of responses. This program was not able to utilize a parser that could sufficiently parse a sentence.

Despite these limitations, the criteria for the computer-created dialog is its ability to produce realistic responses. Proper assessment of an NLP system is often vague. It will be left to others to come up with some standards to evaluate and to properly assess this NLP system.

CHAPTER II

REVIEW OF LITERATURE

2.1. Language acquisition studies

The role of prepatterned language forms and functions has played a prominent role in first and second language acquisition. Peters (1977) finds that some first language acquisition learners, termed *gestalt* learners, tend to reproduce whole segments as single utterances, such as "whatsat." These utterances have specific pragmatic functions. Huang and Hatch (1978) support this finding with the observation of a 5 year-old Taiwanese boy, Paul, who began muttering, "Get out of here" without knowing what the phrase meant. Paul uses other "memorized" utterances and uses them repeatedly in similar situations as well as in extended forms. Dore (1975) posits that, in first language acquisition, children move from one-word utterances to two-word utterances with a pause between words. Children, thus, tend to move from phrasal patterns to extended forms using pattern variations.

In second language (L2) acquisition studies, Yorio (1980) notes that ESL students who learn conventionalized forms of language can improve their communicative competence. He notes that the two major forms are idioms and routine formulas. Their function is to offer social support in awkward situations. Yorio defines an idiom as an expression whose meaning cannot be predicted from its constituent syntactic/morphemic structure. Routine formulas are conventionalized prepatterned expressions whose meaning is tied to standardized communication situations. Both these categories overlap at various times. For example, there are cases when idiomatic routine formulas include euphemisms. Yorio notes, "conventionalized forms make communication more orderly because they are regulatory in nature. They organize reactions and facilitate choices, thus reducing the complexity of communicative exchanges. They are group identifying" (1980, p. 438).

Hakuta (1974) theorizes that second language learners (L2) use acquisition strategies that focus on patterned segments of speech. The L2 learners do not understand the internal structure of these segments, but do understand the pragmatic contexts in which these segments are used. Hakuta makes the distinction between prefabricated patterns that incorporate slots and fillers, and prefabricated complete routines, such as, "How are you?".

2.2 Neurolinguistics

Krashen and Scarcella (1977) note that the child L2 learner is placed in peer or school situations that require interaction quickly before complete competence is attained. The child's advanced short term memory allows him to retain the necessary formulas for interaction. Krashen and Scarcella (p.285) note that, neurolinguistically, automatic speech --conventional greetings, overused expressions, certain idioms, swearing, etc. -- is localized in both hemispheres of the brain, whereas propositional language is lateralized to the left hemisphere. Research has shown that speech routines and patterns are often preserved in aphasia and hemispherectomy patients. However, contrary to Hakuta, they note that speech patterns and routines cannot lead to acquisition, since they are distinct from the creative construction process.

2.3 Cognitive Psychology

Studies in cognitive psychology (Stillings et al., 1987; Broadbent, D. A., 1975) have found that information "chunking" and prepatterned forms are related to the mind's information-processing capabilities. Theories state that the mind processes information in two ways: controlled and automatic.

Controlled-processing is limited by a person's working memory. Working memory is characterized by its small size, its rapid replacement by newly activated information, and by its quick dissipation of its activation energy. The load on working memory may consist of a three or four 'chunks' of information. The classic example is recalling a new telephone number. By rehearsing and repeating the numbers in chunks, the number can be recalled. This processing procedure, however, is rigidly sequenced in a hierarchical manner.

Automatic-processing, on the other hand, makes little demands on working memory. It is automatically triggered by patterns in currently activated information. They are often referred to as a *data-driven* or *pattern driven* processes. They are evoked by patterns in the informational data in the mind (Stillings et al., 1987, p. 52). Automatic-processing offers an advantage in that people can focus on other tasks during its processing. Similarly, lexical phrases or prepatterned forms can be processed automatically, while other discourse functions can be processed in a controlled-processing manner.

Peters (1983) remarks that the lexical redundancy leads to automatization by use of these prepatterned forms. She notes:

> A mature language user may find that certain expressions or variations on expressions are so useful that it would be convenient, as a device for conserving processing time and effort, to be able to retrieve them in as prefabricated a form as possible. Such prefabrications could be in either of two forms: fused and invariant units, and well-rehearsed (automatized) patterns that require minimum of processing (e.g., in the form of insertion of lexical items into a slot) in order to produce the desired utterance. (p. 87)

Research on the psychological effect on perceptual restoration has verified the efficiency of prefabricated language forms. Warren (1970) conducted a phoneme restoration experiment that took a phoneme [b] from the phrase "jump on the bandwagon." The phoneme [b] was extracted and replaced by the phoneme [s]. A second pass extraction of the phoneme [s] completed the task, so that the recording contained "jump on the ANDWAGON." The tape with the target sentence was played for some listeners. In every instance, the listeners heard the phoneme [b], even though it was omitted from the recording. It was concluded that the semantic context of an idiomatic phrase, such as "jump on the bandwagon" can allow for more efficient processing of any target phonemes. Thus, lexical phrases are structures that can be processed with a high degree of efficiency.

2.4. Frames

A central issue in AI research is how to structure and to retrieve information about the world (Chandrasekaran, 1990; Haugeland, 1985; Rosenbaum, Weisler, Baker-ward, 1987; Partridge & Wilks, 1990; Schank & Childers, 1984). This issue introduces problematic questions involving philosophical and epistemological concerns.

Early efforts toward knowledge representation research are reflected in the theory of "frames". Minsky (1975, 1982) proposes that human knowledge representation is in the form of frames, which consists of "slots" and "fillers" :

> We can think of a frame as a network of nodes and relations. The "top levels" of a frame are fixed, and represent things that are always true about the supposed situation. The lower levels have many terminals - "slots" that must be filled by specific instances or data. Each terminal can specify conditions its assignments must meet. (1975, p. 2)

Each frame is composed of a template-like network of nodes or slots that represent a stereotypical event or situation. The top level of the network is assigned values that are normally constant in a given situation. The bottom levels have terminals or slots that can be filled with exceptions or specific data. This is a powerful and flexible system, in which knowledge and rules are codified.

Frames, thus, are ideal for representing "chunks" of knowledge information that allow for top-down, expectation-driven processing (Stillings et al., p. 152). The use of frames utilizes *a priori* knowledge that provides a predictive component to its problem analyses (Sharples, Hogg, Hutchinson, Torrance and Young, 1984). For example, representation of a telephone number in Oregon begins with the expectation that the area code starts with "503" (or “541”). The remaining seven digits in the phone number fill the empty slots of the frame. Conventionalized, stereotypical information drives one's understanding of new events. This information can be applied to new situations, rather than built from scratch.

Hayes (1979) notes that Minsky's definition has two possible interpretations: a metaphysical and a heuristic view. The metaphysical interpretation states that, in using frames, one must make certain assumptions as to what entities exist in the representation of a symbol. For example, the entities that represent the term "football" are "inflated oval shaped ball", "brown color", "nicknamed 'pigskin'". The heuristic interpretation is that frames are a computational mechanism for organizing the processes of storage, retrieval, and inference of representations in computer memory.

An example of a heuristic frame (Bobrow et al., 1977) specifying a certain DATE is as follows:

FRAME	SLOT	FILLER
[DATE	MONTH	Name
	DAY	(BOUNDED INTEGER 1 31)
	YEAR Integer	(DEFAULT 1994)
	WEEKDAY	(MEMBER (Sunday Monday Tuesday Wednesday Thursday Friday Saturday)
		(TOFILL GETWEEKDAY))]

The frame DATE contains slots (i.e., MONTH, DAY, YEAR, WEEKDAY) that can be filled by fillers. Fillers comprise standard values (e.g. NAME), bounded values (e.g. DAY), default values (YEAR), and a list of values (e.g. WEEKDAY has a list called MEMBER comprising of the days of the week). The fillers can be procedures that require some type of action should the slot be filled. In the above example, TOFILL is a procedure that only is activated if the WEEKDAY information is needed. Procedures also can be automatically activated once the slot is filled with some value. The latter procedure is called a demon, while the former is called a servant (Bobrow et al; 1977; p. 15).

Minsky (1975; 1981) proposes that the concept of a frame is similar to the concept of a "schemata" or a "paradigm". However, frames possess four distinctive features from schema. First, frames contain procedures that can be triggered and executed automatically or on demand. Second, frames are conceptually related, allowing objects to be "inherited" from objects higher up in a hierarchical structure. Third, frames are organized in a clearly modular fashion. Finally, Minsky's proposal proposes a frame-based system, rather than a unit memory structure (i.e., schema).

There are, however, fundamental weaknesses with frames from an AI perspective. First, frames are rigidly domain-specific and cannot be extended easily to other contexts. In the previous example, the entities "inflated oval ball", "brown color", "nicknamed 'pigskin'", are valid only for those who interpret "football" as the ball used in the American sport. "Football" in England refers to an American "soccer-ball". Second, the philosophical issue is whether it is possible to depict an everyday activity or entity by a set of primitive values. Many philosophers pointed out that it was impossible to do so (Dreyfus, 1981). As a result, early AI research in symbolic processing suffered from its inability to extend a particular context knowledge to the everyday world knowledge. Third, its use in symbolic processing systems could not solve the "common sense knowledge" problem. Commonsense background knowledge could not be fully encoded with facts or rules, since it is so difficult to define these parameters. Thus, frames simply could not take advantage of this background knowledge (Dreyfus and Dreyfus, 1988). Fourth, frames may not be fully predictable as guaranteed in theory, since a frame could activate different frames (Brown & Yule, 1983). For example, the words, "base," "strike," "pitcher," "ball" are all syntactically unconnected, and could potentially activate four different frames when only one frame, i.e., baseball, is necessary (Charniak, 1982).

Despite the early enthusiasm with frame technology (Bobrow and Winograd, 1977; Charniak, 1978; Hayes, 1979; Riesbeck, 1982, Waltz, 1982), many AI scientists rejected symbolic processing techniques incorporating frame systems in favor of parallel connectionist systems, called neural net architectures. AI researchers, therefore, conclude that the human mind does not operate by manipulating small pre-determined sets of primitives as exemplified by frame systems.

Frames, however, offer an excellent structural representation for sentence-builder lexical phrases (Nattinger & DeCarrico, 1992), providing a framework that can be implemented in English lesson plans. Nattinger and DeCarrico note that "students can begin with a few basic ones (e.g. sentence-builder lexical phrases), together with their functions, then practice several alternatives based in the same slot-and-filler frames" (1992, p. 122). The actual use of a frames may, in fact, be useful for studying lexical phrase generation in human language.

2.5. NLP Frame-based systems

Computational linguistics and NLP systems have resulted in functional systems that account for prepatterned forms in human language production. These systems require the lexicon to account for any semantic or pragmatic interpretations for any group of lexical items. Thus, lexicons will tend to include phrasal patterns, such as idioms or collocations, in addition to feature sets of individual lexical entries.

Bobrow et al. (1977) designed a frame-based dialog system (GUS, Genial Understander System), that assumes the role of a travel agent. GUS's objective was to generate **realistic** --versus **real--** dialog within a human-machine interface.

Bobrow (1977) notes several problems that NLP system designers must address in their projects. First, in natural dialog, each participant occasionally assumes topic initiative during the course of a conversation. NLP researchers must determine if the human or the machine or both have control of the topic initiative. GUS assumed control throughout the dialogue. Second, indirect speech acts must be taken into account in any dialog system. Speech acts look at utterances as performative acts (Austin, 1962). In direct speech acts, the primary act is closely tied to the literal meaning of the utterance. In indirect speech acts, one illocutionary act is performed indirectly by way of performing another speech act. For example, "Can you reach the salt?" has the indirect speech act meaning of "can you pass me the salt?" GUS possessed limited inferential power to interpret indirect answers to direct questions, but was sufficient for the project (Bobrow et al., 1977, p. 157). Third, the issue of anaphoric reference over sentence boundaries is a central topic of research in NLP studies, and again arises in any dialog system. GUS was able to handle this problem in a simple manner by using frames, but it still could not handle pronominal reference (Bobrow et al., 1975, p. 171). Fourth, sentence fragments often occur in natural conversation, and a NLP system must be able to process these structures. GUS assumed that fragments could function as fillers in a frame, so that the reasoning component of the system was able to continue making the proper inferences. Fifth, Bobrow notes that conversations conform to patterns that are used in special circumstances. Thus, GUS used the specialized language of ticket agents (Bobrow et al., 1977, p. 158).

Bobrow (1977) uses frames to direct the course of a conversation for a flight scheduler. GUS assumes that the conversation will adhere to typical discourse structure for making trip arrangements. The system first creates an "instance" (i.e., a working frame) of the dialog frame and proceeds to fill the slots based on specifications in the prototype. When the slots are filled by a different instance of another frame, called a subframe, then the slots of this subframe are filled. Bobrow notes that control moves in a depth-first, recursive process that completes work on a slot before continuing on to another slot. From the previous example from Bobrow (1977, p. 15):

FRAME	SLOT	FILLER
[DATE	MONTH	Name
	DAY	(BOUNDED INTEGER 1 31)
	YEAR Integer	(DEFAULT 1994)]

An **instance** of the prototype is:

[ISA DATE	MONTH	May
	DAY	28]

ISA indicates that it is an instance of the frame, whose name follows (i.e., DATE). It is not always necessary to have the slots filled, but only relevant information is necessary for its particular processing step. When a slot is filled by a new instance of a frame, the slots of that instance are filled in the same way. This procedure operates in a depth-first, recursive manner. The slots may occasionally be filled out of sequence either through information given by the human or by procedures attached to previous slots (Bobrow ,1977, p. 166).

2.6. Pragmatics and Discourse

Pragmatics is defined as the linguistic dimension of social interaction (Mey, 1993). Other definitions are "the sustained production of chains of mutually-dependent acts, constructed by two or more agents each monitoring and building on the actions of the other" (Levinson, 1983), and pragmatics as "the study of how utterances have meanings in situations" across sentence boundaries (Leech, 1983).

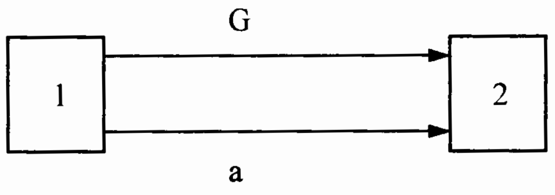
Historically, linguistic researchers assigned pragmatics as a general "waste-basket" for contextual anomalies in their work (Mey, 1983). The Chomskian revolution in syntax gave little importance to pragmatic studies. The modeling of grammatical competence, the native speaker's knowledge of the language, was the focus of syntactic study. The study of the use of language --pragmatics-- was relegated to a secondary position in linguistics.

Austin (1962) noted the limitations of world-view truth conditional propositions of semantic theory, which claim that declarative sentences are true or false, if they contain a proposition about the world. However, some utterances, such as, "good luck," cannot possess a true or false condition, since they are not propositions. Austin stated they are 'words that do things' or perform a certain act (or speech acts). Mey (1993) notes that speech acts induce a change in the existing world.

Speech acts have different aspects or "forces" according to Austin. Levinson (1983) notes that locutionary force is an utterance with determinate sense and reference. The perlocutionary force deals with affecting change on the audience by means of uttering a sentence. Illocutionary force occurs by "virtue of the conventional force associated with the utterance, such as naming a statement, offer or promise" (Levinson, 1983; p. 236).

Searle (1975) noted that some utterances have a primary illocutionary act, but its secondary meaning (i.e., illocutionary act) is not literal. For example, smokers are familiar with the utterance, "do you have a light?" The uttered question is not asking specifically if the listener has a match, but is requesting a match from the listener. In this sense, the literal meaning is not accepted, but the secondary meaning is understood. The force behind such utterances are called indirect speech acts.

Leech (1983) proposes that pragmatics, including speech acts, involve problem-solving from both the speaker's and listener's perspective. In the case of speech acts, the speaker must produce an utterance that can affect change in the listener. The listener must interpret the speaker's intention.



1 = initial state
2 = final state
G = goal of attaining state 2
a = action by means of a speech act

Figure 1. problem solving using speech acts (Leech, 1983, p.36)

Pragmatics offers some principles that describe any discourse process. They are: the cooperative principle, four maxims that make up this principle, and implicature that can result from these maxims. Grice(1975) proposed maxims that guide cooperative conversational behavior: the maxims of quality, quantity, relation and manner. The maxim of quality requires that one try to make a truthful contribution. One should not say what one believes to be false unless adequate evidence is provided by the speaker. The maxim of quantity requires that your contribution be as informative as possible without being excessively detailed. The maxim of relation requires that a contribution should be relevant when presented in the conversation. The maxim of manner requires that expressions be clear; not obscure or ambiguous. These four maxims, which make up the cooperation principle, have been widely accepted as a model for human-machine dialogs (Gal et al., 1992).

Sperber and Wilson (1986) offer an alternative to Grice's Cooperative principle, which they called the Relevance principle. This principle states that in any given context, what people say has some relevance. The goal of conversation is to achieve successful communication. Here the speaker is recognized as one who has something of relevance to the topical framework of the conversation.

The utterance conveys the speaker's intention. Mey (1993) notes that Grice's cooperative principle makes claims about "common purposes or set of purposes." Thus, relevance theory states that the purpose of communication is to "enlarge mutual cognitive environments" (Sperber & Wilson, 1986; p. 193) and not to "duplicate thoughts."

Grice (1975) notes that when one of the maxims of the cooperative principle is violated, the utterance acquires a new meaning that can be inferred from the context Such an utterance is called a conversational implicature. Some utterances are conventionalized expressions of requests, invitations, and offers. These have implicit meaning, and are called conventional implicatures or conventionalized indirect speech acts (Searle, 1975).

Ethnomethodologists, a subgroup of anthropology, studied conversation and found that it was highly structured and coherent. They devised a transcription method that accounted for linguistic as well as non-linguistic phenomena, such as laughter (Mey, 1993). Their aim was to determine what "rules" or principles were followed in a conversation through a process called conversation analysis (CA). This contrasted with discourse analysis (DA), which Mey (1983; p. 194) associates with traditional field linguistic methods. Mey notes that discourse analysis operates deductively and is "rule or grammar" driven, whereas, conversation analysis uses inductive methods and is "data-driven."

In CA, the basic unit of conversation is the "turn". Sacks (1975) defines a turn as a shift in the speaking flow of a normal conversation. Mey (1983) notes, "yielding the right to speak, or the 'floor', as it is often called, to the next speaker thus constitutes a turn" (p. 217). Schegloff (1992) posits that a turn is constructed out of building blocks called "turn constructional units". These units are made up of words, phrases, and sentences that, upon possible completion, can be treated as the end of a turn.

Turns occur at points in a conversation that are called "transition relevance places"(TRP) (Sacks; 1974). Examples of TRPs are pauses for breath, silence, and ending one's contribution. The speaker can also bypass the TRP by moving ahead with an utterance; creating an unnatural break --in midsentence-- and thus indicate the desire to move ahead.

The hearer can support the conversation by "back-channeling." This involves using short utterances for agreement to indicate that the hearer is involved in the conversation. Forms of back-channeling include short phrases, such as, "*I see*," "*right*," "*uh-huh*," etc.). Languages, such as Japanese "aizuchi", use extensive back-channeling in their conversations.

Searle (1992) regards the concept Schegloff's "turns" as lacking the explanatory power to address the key issues in spoken discourse.

> "Schegloff and I agree that units of speech in conversation come in chunks. I think these chunks have to be defined intentionalistically, but the boundaries of the chunks are not necessarily the boundaries of single speech acts. ..these chunks are what he is calling 'turn construction units,' and the boundaries of the chunks, he calls 'transition relevant places.'..But what we now need to know is what is the explanatory status of the description of the patterns? If the description of a pattern specifies the intentional content of a rule that the agent is following then the description has some explanatory force. But if the description just identifies some regularity in behavior then so far no explanation has been given." (p.146).

Studies have found that there exists speech/language mechanisms that aid in discourse comprehension. Reichman (1985) defines one of these mechanisms as "discourse expectations," which speakers use to make predictions of future discourse, based on surface linguistic forms. Reichman notes that certain surface linguistic forms, called *clue words*, act as discourse expectations. They signal simultaneously that a discourse item is in focus and that a conversational move (or "turn") has taken place (Reichman, p.30). Conversation failure arises when clue words are not interpreted correctly as a result of differing belief systems of the speaker and the listener (Ringle and Bruce, 1982). Shared background knowledge aids speakers in their mutual understanding.

Reichman's clue words are utterances that "pre-sequence" to other utterances. Mey (1993) notes that these presequences offer an initial exchange, after which the main purpose of the dialog is expressed. For example, common presequences in shopping are (Mey; p. 222):

> Excuse me.
> I wonder if you have any X...
> Do you by any chance have X...
> Does your store carry X...
> where X is the item in question.

Pre-sequences offer a method to structure conversations. It is predictable in that the adjoining response can be quickly ascertained. It signals what the utterance means and what the utterance's function is. When two utterances by two different speakers are adjacent to one another in a conversational exchange, these utterances are called adjacency pairs (Schegloff & Sacks, 1973). They capture recurrent patterns in conversation and are descriptors of conversational organization (Tsui, 1991).

Common adjacency pairs include greeting-greeting, order-compliance, request for information-fulfilling the request, etc. The first part of an adjacency pair is a presequence that predicts a likely response from the listener. Thus, given the first part of the pair, the second part is immediately relevant and predictable.

Levinson (1983) argues that one cannot formulate any sequencing rules based on adjacency pairs due to the unpredictability of the second utterance. Tsui (1991) challenges Levinson's view by emphasizing that the descriptive power of an adjacency pair predicts what the second exchange utterance might be, rather than what it actually is (p. 117).

The coherence of a conversation means that the speaker's utterances fit into the topical framework and makes sense to the listener. Coherent utterances are cohesively linked, lexically, grammatically, and interactionally with the immediate conversation (Stenstrom, 1994). The coherence principle states that, "an utterance must be related to either the illocutionary intention or the pragmatic presuppositions of the preceding utterance or it will fail a coherent sequence," (Tsui; 1991, p.123).

CHAPTER III

METHOD

Lexical phrases are lexico-grammatical "chunks" of words, which possess specific pragmatic functions within discourse (Nattinger and DeCarrico, 1992). These form/function composites of varying length aid conversational fluency. Nattinger and Decarrico (1992) further emphasize that lexical phrase theory accords with discourse analysis and speech act theories (p.59).

This thesis presents the design of a computer assisted language learning (CALL) program written in the computer language, PROLOG. It will provide a practice exercise to teach a sentence-builder lexical phrase to second language learning students of English. While the input will primarily be via the terminal, the purpose is to provide some practice in improving conversational fluency. Nattinger and DeCarrico (1992) stress that lexical phrases be practiced with sufficient variation. Thus, the program should support sufficiently random variation. Once students have mastered these phrases, they can focus on grammar lessons concerning any individual lexical units in a lexical phrase.

The program presents a scenario where the participants are college students who pass one another in the hallway. After they exchange greetings, the first participant (i.e., the computer) informs the other student that a test, for some unknown class, will be postponed till "next Tuesday". The second student answers with a surprised remark. The first student responds that he/she could use the extra time for study, and then says, "good-bye". The dialog ends with when the second participant says "good-bye". At this time, the program asks if the human participant would like to continue again or not. If yes, then a new dialog is generated; if no, the program ends.

The scenario assumes that both participant have common knowledge that 1) they are classmates in a particular course, 2) they originally have a test in that course in the near future, 3) the test is postponed to the next Tuesday, and 4) the test will be difficult for at least one of the students (i.e., the first participant). This background knowledge is not known to us, but can be inferred from the conversation.

3.1. Limitations

This program had three main limitations: 1) parsing, 2) syntax, and 3) evaluation. Parsing involves analyzing a sentence into its constituent parts. This enables programs to extract categorical information from any responses, and to thereby exhibit a form of "intelligence". A reliable parser, however, was not available for this project.

Second, issues of resolving anaphoric reference over sentence boundaries, indirect speech acts, and fragments are beyond the scope of this study. Solutions could not be addressed due to the complexity of each problem. Therefore, forms such as sentence fragments, indirect speech acts, anaphoric reference, were not used in the program's generated dialogue.

Proper assessment of an NLP system is often vague. It will be left to others to come up with some standards to evaluate and to properly assess this NLP system.

3.2. Conversation structure

Conversational patterns vary from person to person and place to place. Nattinger and DeCarrico (1992) note that conversations are made of collection^{s} of discourse patterns. Both participants contribute to the conversation. Once the participants are finished, they close the dialogue (Nattinger and DeCarrico, 1992, p.71). The basic structure, however, can be characterized as an opening, followed by a message, and ending on a closing (Stenstrom, 1994).

The discourse strategies vary in face-to-face conversations (Stenstrom, 1994). Typically, these strategies have the following tendencies:

1). opening and closing sections may be lacking.

2). openings and closing are affected by the degree of formality.

3). topic changes, shifts and drifts are common.

4). body language plays an important role.

5). extralinguistic details play an important role (i.e., environment).

In openings, the two parties greet each other. Phatic talk involves general questions about weather, health, personal matters, and polite phrases (Stenstrom, 1994, p. 150). Phatic talk tends to precede the main topic.

Topical strategies include introduction and termination of a theme; changing, shifting and drifting from a topic; and digressing and resuming a given topic. Closings involve the adjacency pair formats for pre-closings (*it's getting late*, *right*, *all right, OK)*, closings, thanks, and good-byes.

In conversation settings, lexical phrases of social interaction and necessary topics play a significant role with the dialogue. Lexical phrases of social interaction fall under two groups: conversational maintenance, conversational purpose. Conversational maintenance involves how conversations begin, continue, and end (Nattinger and DeCarrico, 1992, p. 60-61). Necessary topics are lexical phrases that are commonly asked questions for obtaining information. Discourse devices are lexical phrases that connect the meaning and structure of the discourse and play a lesser role in conversation (Nattinger and DeCarrico, 1992, p.71). Examples of these three types of lexical phrases are seen in table 2, 3 and 4.

summoning	*excuse/pardon me*	checking	*all right?*
	how are you?	comprehension	*understand (me)?*
responding to	*how are you doing?*	shifting a topic	*by the way..*
summons	*what's going on?*		*oh that reminds me of X*
nominating a topic	*what's X*	shifting turns	*so OK*
	have you heard about X?		*excuse/pardon me*
clarifying:	*what did you mean by X?*	closing	*I must be going*
(1) audience	*excuse/pardon me?*		*I've got to run*
(2) speaker	*what I mean is X*	parting	*good-bye*
	how shall I put it?		*see you later*

Table 2. Lexical phrases for conversational maintenance
(Nattinger and DeCarrico, 1992, p. 62-63).

expressing	*thanks (very much)*	questioning	*do you X?*
politeness			*is/are there X?*
requesting	*Modal+Pro+VP?*	answering	*yes, (there/it/they is/are X.*
	May I X?		
offering	*Modal+Pro+VP?*	complying	*of course*
			I'd be glad to
refusing	*of course not*	complimenting	*NP+BE/LOOK+intensifier+ Adj*
	I'd rather you X		
asserting	*it is (a fact that) X*	responding	*(yeah) I know*
	word has it that X	(1) accepting	*(oh) I see, no kidding*
	it seems X		

Table 3. Lexical phrases for conversational purpose
(Nattinger and DeCarrico, 1992, p. 62-63).

logical connectors	*as a result (of X)*	temporal connectors	*and then,*
			after X then
spatial connectors	*around here*	fluency devices	*you know*
	over there		*it seems (to me) that X*
exemplifiers	*in other words*	relators	*the thing X is Y*
	it's like X		*not only X but also Y*
qualifiers	*it depends on X*	evaluators	*as far as I know/can tell*
	the catch is X		*there's no doubt about X*
summarizer	*to make a long story short*		
	my point is that X		

Table 4. Lexical phrases for discourse devices
(Nattinger and DeCarrico, 1992, p. 64-65).

3.3. Exchange structures

Exchange structures, or adjacency pairs, are short dialogs involving two turns that present expected sets of utterances. Common exchange structures include the following (Nattinger and DeCarrico, 1992, p.119-120):

1) summons-response

Hi, how are you (doing)? - (I'm) fine, thanks (and you)?

Good morning/afternoon/evening. - Good morning/afternoon/evening.

2) asserting-accepting

Word has it that *X*. - No kidding.

It seems (to me) that *X*. - I see.
where *X* is a declarative sentence.

3) closing-parting

(It's been) nice talking to you. - (Well), so long (for now).

It's been nice talking to you, but I must be going. - Good-bye.

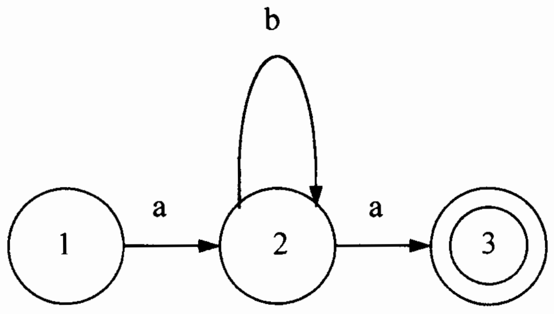
The program will use the above three exchanges to simulate a short dialogue between the human subject and the computer. The conversation structure will process a regular opening-main message-closing dialogue. The computer will assume the role of the first speaker and the human subject will assume the role of second speaker.

The context is a university environment between two classmates who meet in passing in the hallways. They exchange a brief greeting, then the first speaker brings up the topic of the exchange, "Word has it that *X*," where X represents any declarative sentence. In this project, X equals "the test is postponed till Tuesday." The human subject then responds with an answer appropriate for the exchange structure. Finally, a closing sequence will end the dialog.

3.4. Finite Automaton

Computer models began with general finite automata theory. A finite automaton (FA) is a state machine which recognizes well-formed strings of a regular language. An FA operates by having a finite number of states that transition to another state depending on a particular input. It has three main components: a finite set of states, an alphabet of possible input letters, and a finite set of transitions arcs. Minimally, an FA must have an initial state and a final state with a transition arc connecting the two states.

In Figure 2, a three state FA is shown. State 1 is the initial state, and state 3 is the final or terminal state, which is indicated by the double circles. State 1 and state 2 are nonfinal states and are indicated with a single circle. A successful transition occurs when the next symbol of the input strings matches the symbol on the arc. Each symbol is read one letter at a time to recognize a string.



(a)

	a	b
1	2	0
2	3	2
3	0	0

(b)

Figure 2. A simple finite automaton. (a) a transition diagram, (b) a transition table.

The transition diagram begins at state 1 and reads a symbol “a”. Since it matches the transition arc, it moves to state 2. At state 2, the second input symbol is read. If it is a “b”, then it loops back to state 2. If it is an “a”, it moves to state 3. Since state 3 is the final state and there are no more input symbols, the process has completed. In more technical terms, the regular language L consists of an alphabet “a” and “b” and the grammar $\{ab^{*}a\}$, where the asterisk or Kleene star indicates any number of “b’s” (including zero b’s). The language recognizes “well-formed” strings, such as “aa”, “aba”, “abba”, etc.

An alternative representation of a finite automaton is called a state transition table. The rows represent the different states of the FA. The columns represent the arcs from one state to another. The symbols for each arc is placed at the head of each column. A transition operates as follows (figure 2b): the FA begins at state 1 and reads the first symbol in the input string, say “a”. If the symbol matches a labeled arc, then it will transition to state 2. When the input is read and matched against an arc symbol “a”, then the state will transition to state 2. The FA continues to read the input string symbol until all of the symbols have been read. When all the symbols are read and the FA is in final state (state 3), the FA is said to accept the string. If the FA is not in state 3 and all the input has been read, then the string is rejected and the FA is said to have failed. The zero in the transition refers to a trap state, which indicates that an transition to this state will reject the string. In an FA, this state is equivalent to having no arc.

The simple model of a finite automaton is said to be **deterministic** in that it will transition from one state to another given a single input. Thus, given a single input, the FA will make a single transition to a different state in a one to one correspondence of input to state. There are no choices as to which state to transition to, and there is no history of what transitions occurred prior to the current state. An FA is called a “machine,” because it processes an input, then moves from one state to another state depending if it matches a transition arc. The language that an FA can recognize is said to be the language defined by the finite automaton. In the example in figure 2, the language accepted by the FA has strings ”ab, aab,abb,abab,” etc. The input string “aaa” is invalid and will be rejected by the FA.

3.5. Moore Machine

A slightly more sophisticated model of a finite automaton is a Moore machine. A Moore machine is a finite automaton with two additional components: a set of output characters and an output table. Finite automata possess an alphabet of possible input letters. A Moore machine has an alphabet of possible input letters, similar to an FA, and an additional set of possible output characters. Further, a Moore machine has an output table that displays what character is printed by each state that is entered. Figure 3 illustrates a simple Moore machine.

By normal convention, the first symbol printed, "X", is specified in the start state. The operation is similar to an FA, except that as each state is entered an output character is printed. If the first symbol in the input string matches the symbol on the transition arc ("a" in figure 3), then the machine goes to state 2. For example, as the Moore machine enters state 2 after recognizing and matching the input letter "a", it will output the character "X". Moore machines differ from finite automata in that they do not define a language of accepted words, since every input string generates an output string (Cohen, 1981).

Further, Moore machines do not have a final state; processing terminates when the input letter is read and the last character is printed, regardless of the state it has entered.

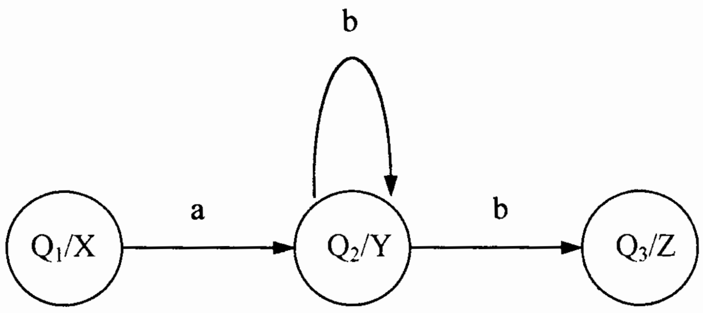


Figure 3. A Moore machine.

Just as FA's had state transition tables, Moore machines also can have its operation characterized in a state transition table (Figure 4). Note again the use of zero trap states which move the state machine into a state that can neither be entered or exited. This is analogous to having no arcs for a certain input character.

Old State	Transition Table New State: After input "a"	After input "b"	Output Table (character printed in the old state)
Q_1	Q_2	0	X
Q_2	Q_3	Q_2	Y
Q_3	Q_3	0	Z

Table 5. State transition table for a Moore machine.

As mentioned previously, finite automata, like the Moore machine, are said to be deterministic. A state machine is deterministic if its future path is predicted by the current state that it is in. Accordingly, there is one course of action and one result. A nondeterministic system has choices that make the final outcome uncertain. The Moore machine illustrated previously is deterministic.

3.6. PROLOG

PROLOG (PROgramming in LOGic) is a computer programming language that evaluates relationships between objects. It was developed in the early 1970s for natural language processing (NLP) applications and has been used extensively for artificial intelligence projects. It is a declarative programming language that requires the programmer to define the logical relationships between various objects.

Most programming languages are procedural in scope. That is, the problem, the relationships between objects and the procedure (i.e., "how") to solve the problem must be specified. The mechanisms behind Prolog are quite complex and are incidental to this study. Therefore, the fundamental principles of Prolog will be presented in this chapter.

Based on predicate logic, a Prolog program consists of a set of **facts, rules,** and **questions** (Clocksin and Mellish, 1994). A **fact** expresses a relationship between one or more objects as an unconditional truth. For example, the declarative sentence, "Mary is female" may be expressed as a unary or one-place relation.

```
female(mary).
```

A unary relation can only be used to determine yes or no, if "Mary is female," (Bratko, 1990). Since, a fact is always unconditionally true, then the program will determine that "Mary is female" is a true condition.

A binary relation expresses relations between pairs of objects. For example, the sentence "Tom is the parent of Mary" can be expressed as the following binary relation. Similar to a unary relation, the binary relation verifies the truth of "Tom is the parent of Mary".

```
parent(tom, mary).
```

The objects in brackets are called *arguments*. The name of the relationship that defines the arguments is called the *predicate*. Thus, the predicate *parent* has two arguments, *tom* and *mary*. Prolog programs have usually store a collection of facts, which is called a *database*.

Rules are expressions that are either true or false depending on the truth conditions of its elements. It is a general statement about the objects and their relationships (Clocksin and Mellish, 1994). For example, the rule "p is true if q1 is true and q2 is true" is written as follows:

```
p :- q1, q2.
```

P represents the **head**, or **proposition**, of the rule, and q1 and q2 form the **body** of the rule. The body of a rule are **conditions** that determine, whether the head of a rule is true or false.

A comma (",") separating the conditions indicates a logical AND relationship. For a logical AND, both elements in the body must be true for the head to be true; otherwise, the head is false. A semicolon (";") represents a logical OR relationship. That is, one or both of the elements in the body must be true for the head to be true. If these conditions are not met, then the head is false. The truth table for these relationships are as follows:

AND	q1	q1	Result: p
	F	F	F
	F	T	F
	T	F	F
	T	T	T

(a)

OR	q1	q2	Result: p
	F	F	F
	F	T	T
	T	F	T
	T	T	T

(b)

Table 6. Truth condition tables for (a) logical AND, (b) logical OR. F=False, T=True.

The head and body of a rule are represented in Prolog as a set of **predicates** and its **arguments**. A predicate expresses its relationship with its arguments. For example (Bratko, 1990):

```
sister(mary,john):-
        parent(tom, mary),
        parent(tom, john),
        female(mary).
```

The rule expresses the declarative sentence “Mary is the sister of John, if Tom is the parent of Mary and Tom is the parent of John and Mary is a female.” If any of the objects in the body is false, then “Mary is the sister of John” is false. The predicates are *sister*, *parent*, and *female*; the arguments are *mary*, *john*, and *tom*. Rules and facts are grouped into **clauses**.

Bratko(1990, p. 13) uses state diagrams to represent these relationships. Each node or state in the graph represents the objects or arguments in the rule. The arcs between each node represents the binary relations. The arrows point from the first argument to the second argument. The dashed arrow represents the head of the rule. If the “solid line” conditions are true, then the relation shown by the dashed arc will also be true.

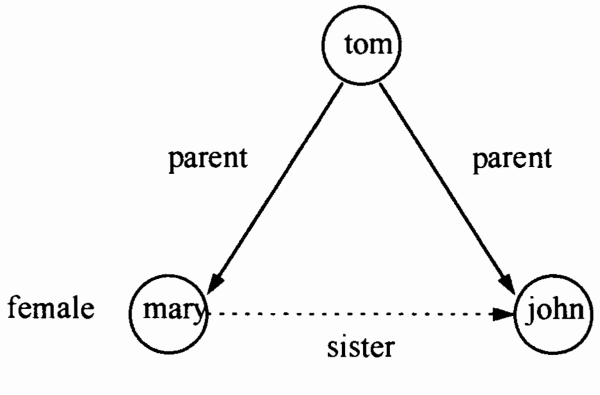


Figure 4. Definition of the sister relationship.

Prolog, in many ways, is a conversational computer programming language. Once the set of facts and rules has been established, then **questions** about these relationships may be asked. When a question, or query, has been posed, it will search through the database for any possible matches. Two facts match if their predicates and arguments are the same. From a prior example, the question "is Tom the parent of Mary?" can be asked.

```
database:     parent(tom, mary).              parent(tom, mary).

query:        ?- parent(tom, mary).           ?- parent(tom, bill).
              yes                             no
                    (a)                             (b)
```

Since the question (a) matches the fact, the program answers, "yes". In question (b), the program answers "no," since a match could not be found. Technically, **no** does not mean a given relation is false from a truth conditional perspective. It means that a match could not be found; and thus, the question is not *provable* (Clocksin and Mellish, 1994).

Prolog can use names that stand for certain objects. These objects are determined within the program and are called **variables**. When Prolog uses a variable, it is *instantiated* (or not *instantiated*) to a particular object. Prolog distinguishes variables from objects through capitalization.

```
database:        parent(tom, mary).

query:           ?- parent(tom, X).
                 X = mary.
```

The query asks, "Tom is the parent of who?" In this case, X represents a variable that has no particular value before the question is posed. When the program processes the question, it searches the database for a matching predicate and arguments. Since *parent* and the argument, *tom,* match, then the variable X gets instantiated with the object, *mary.* Once X is instantiated, it has the value *mary.* The answer to the query responds with *X is mary.*

Prolog's control structure is based on a procedure called **unification**. It matches the data structures with any free variables. A variable can assume any type of data that is assigned to it. (Prolog variables can only assume one value however.) An more complicated example of unification is given in the following simple program.

```
likes(ellen,tennis).
likes(john,football).
likes(tom,baseball).

likes(bill,Activity) :- likes(tom, Activity).
```

The program has a rule and three facts. The rule states that, "Bill likes some general unknown activity if tom likes some general unknown activity. Note that *Activity* begins with a capitalized letter, which indicates that it is a variable with no initial value. The program must search the database to arrive at an instantiated value for *Activity.*

A query searches for a solution to the question "does Bill like baseball?", and can be entered as:

```
query:      likes(bill, baseball).
answer:     yes.
```

The program will attempt to match the predicates, "likes(bill,baseball)" to "likes(bill, Activity)," where the capitalized object, *Activity*, is a variable. Since the predicate, "likes(bill, Activity)," is a rule, the computer processes the condition, "likes(tom, Activity)" by matching to a similar predicate. It first attempts a match with "likes(ellen, tennis)". Since the first arguments in each predicate is different (i.e., *ellen* and *tom*), then the match fails.

The computer moves on to the next fact to get a successful match. Once again, *tom* and *john* do not match, and the match fails. Finally, the computer matches "likes(tom, Activity)" to "likes(tom, baseball)". Since Activity represents a variable (i.e., it has no value until assigned one), it instantiates *Activity* with *baseball*. That is, *Activity* has the argument *baseball*. Once a successful match has occurred, then the condition is true, and control moves back to "likes(bill, Activity)." Since Activity now holds the argument *baseball*, the head of the rule is now, "likes(bill, baseball)". This matches with the query and results in a positive truth condition and a match. Thus, the computer prints out a *yes* to confirm a successful match. Prolog programs operate in this fashion of matching predicates with other predicates.

Table 6 indicates the steps the program used to solve for the query:

?- likes(bill,baseball).

Step	Predicate	Action
1.	likes(bill,Activity),	-*Activity* is instantiated to *baseball.*
2.	likes(tom,Activity),	-*Activity* has value of *baseball.* attempts a match for *likes(tom, baseball).*
3.	likes(ellen, tennis),	-match fails. backtrack & try next predicate.
4.	likes(john,football),	-match fails. backtrack & try next predicate.
5.	likes(tom, baseball),	-**yes**. match is successful.

Table 6. Steps in solving for ?-*likes(bill, baseball).*

Prolog also has two other main mechanisms that help process its code: backtracking and recursion. Prolog will attempt to find a successful match in its clause. If it is processing the head of a rule, then it must evaluate all of the conditions for a truth. If it finds a truth, then it advances in the body of the rule. If it fails, then it backtracks to the last successful match.

Backtracking is important because it can offer a variety of results in a non-deterministic manner (i.e., every input has a choice of which path to take). Whereas without backtracking, the program must operate in a deterministic fashion (i.e., every input has a single corresponding output), where each predicate matches successfully. An example of backtracking was seen in table 6. The search for the predicate *likes(tom,baseball)* failed on its first reading of *likes(ellen,tennis).* The program backtracked to the last successful match, which was likes(tom,baseball). Next, it likes(john,football), which also failed. The program backtracked again, and, on its next attempt, made a successful match.

Recursion means that a rule can call itself within its own body. Recursion is an important property of natural language. It enables the language to generate strings of infinite length. Most procedural computer languages are not able to implement recursion in their rules, since it causes difficulties in executing instructions. Prolog, however, uses recursion extensively within its programming structure. Thus, the previous rule is recursive since it calls its own predicate "likes".

```
likes(bill, Activity):- likes(tom, Activity).
```

3.7. Lexical phrase CALL program

The lexical phrase computer assisted language learning (CALL) module is designed for practicing communicative competence using these form/function composites. It is written in the declarative programming language, PROLOG. The dialog's control structure is a Moore model finite automaton with thirteen states.

The scenario is between two university classmates, who meet in passing in the hallways. After an initial greeting, the first student mentions that an upcoming test will be postponed till next Tuesday. The second student expresses some form of surprise as he/she was not aware of the test's postponement. After this episode, both close the meeting and depart. The dialogue consists of three components: the opening, the main message, and the closing (figure 5).

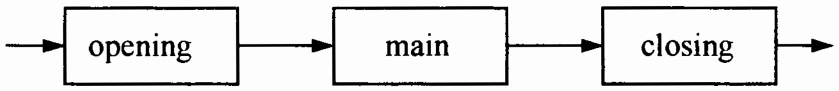


Figure 5. The structure of the dialogue.

The opening uses eight states. In this automaton, states S0, S1, and S2 are all potential start states. The program will generate a random number between zero to two to determine which state to enter. State S1 will output *hello*. State S2 will output *how are you doing?*. State S0 will output the union of states one and two, and will print, *hello, how are you doing?*.

The possible input responses are *good, I'm fine, o.k., busy*, and *hello*. From these inputs, the automaton will go directly to the main message state. In this mode, it will retain the initiative (i.e., single initiative mode) in the dialog before completing the turn. A turn is defined as any utterance speaker A says before speaker B takes over (Stenstrom, 1994).

An additional set of possible responses may be used in place of the first level responses. That is, when the initial phrase is *hello* (state 1), an alternative acceptable response is *hi*, instead of *hello*. Each first level response has an alternative second level response (Table 7).

	STATES		
	S1	S2	S0
initial phrases	hello.	how are you doing?	hello, how are you doing?
possible responses 1. first level 2. second level	 hello. hi.	 1) hello., 2) good., 3) I'm fine., 4) o.k., 5) busy. 1) hi, 2) great., 3) fine., 4) so so, 4a) fair. 5) swamped.	
main message	1) word has it that the test is postponed till Tuesday. 2) it seems that the test is postponed till Tuesday.		

Table 7. Possible openings in single-initiative mode.

The opening component also handles mixed-initiative mode. With the output *how are you doing?*, the valid responses include, *how are you?* or *fine. how are you*? The human subject responds by reciprocating the greeting, and is expecting some reply to his greeting. The computer must regain the initiative in the dialog by making a counter response, and then by bringing up the main message of the dialog (Table 8). An alternative response is a reply with the addition of *and you?*. The response can be, for example, *good. and you?* The initial participant must reply, and then present the main message. In summary, the *how are you doing*? responses are grouped into response A; the *good, and you*? responses are grouped into response B. The main message is presented in response group C. If the possible response is from group A, then proceed to response group C. If the possible response is from group B, then proceed to response group C.

	STATES		
	S1	S2	S0
initial phrases	hello.	how are you doing?	hello, how are you doing?
Note: if A, then C; else B, then C. A) possible response 1. first level 2. second level	1) how are you doing? 2a) how are you?, 2a) how's it going?		
B) possible response 1. first level 2. second level	hello. hi.	[1) hello., 2) good., 3) I'm fine., 4) o.k., 5) busy.]+**AND YOU?** [1) hi, 2) great., 3) fine., 4) so so, 4a) fair. 5) swamped.]+**AND YOU?**	
C)counter responses	[1) hello., 2) good., 3) I'm fine., 4) o.k., 5) busy] + main message		
main message	1) word has it that the test is postponed till Tuesday. 2) it seems that the test is postponed till Tuesday.		

Table 8. Possible responses in mixed-initiative mode.

The main message section uses the sentence builder lexical phrases, *word has it that*__, and *it seems that*___. The program generates a random number between 0 and 1. If the number is zero, then the lexical phrase, *word has it that* ___ is outputted first. If the number is one, then *it seems that*___ is printed. The declarative sentence that fills the slot is "the test is postponed till Tuesday." The acceptable responses are *I see* and *no kidding*. Both input sentences have variants that are also acceptable responses. These are: *no way*, *is that so?*.

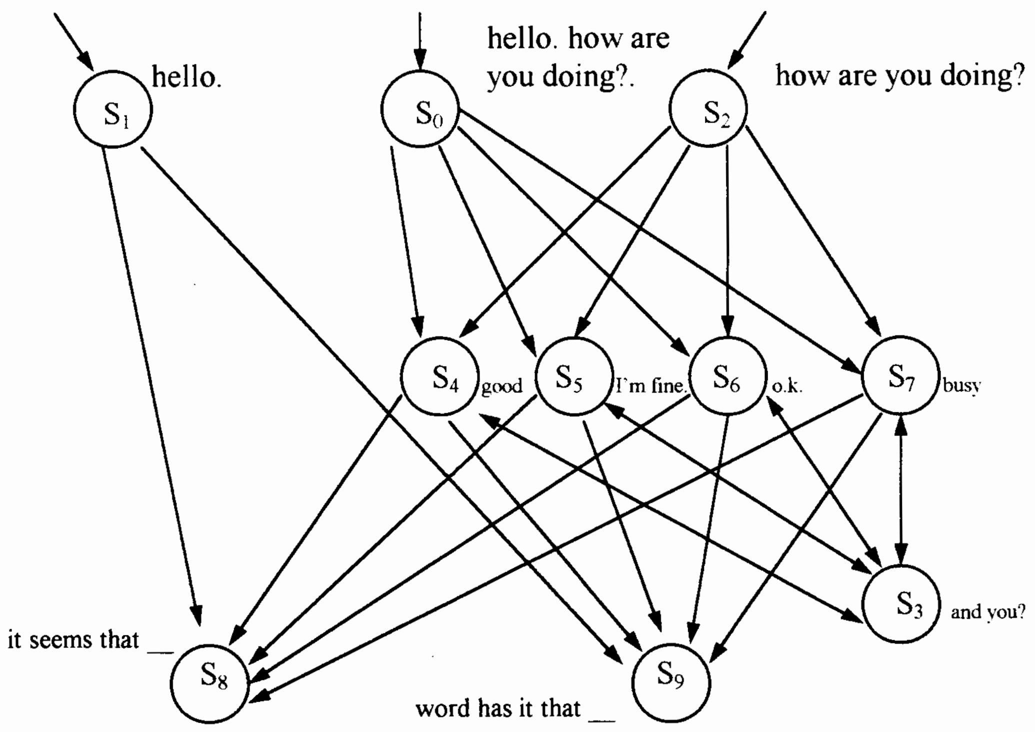


Figure 6. State transition diagram outlining the opening and main message sections.

The closing section ends with the first participant outputting *I could use the extra time...Gotta run. See you later.* The acceptable responses are: *See you later*, and *good-bye*. The variants are: *so long*, and *bye*. When the final response of *good-bye* is read by the program, then it completes the dialogue and returns to the beginning.

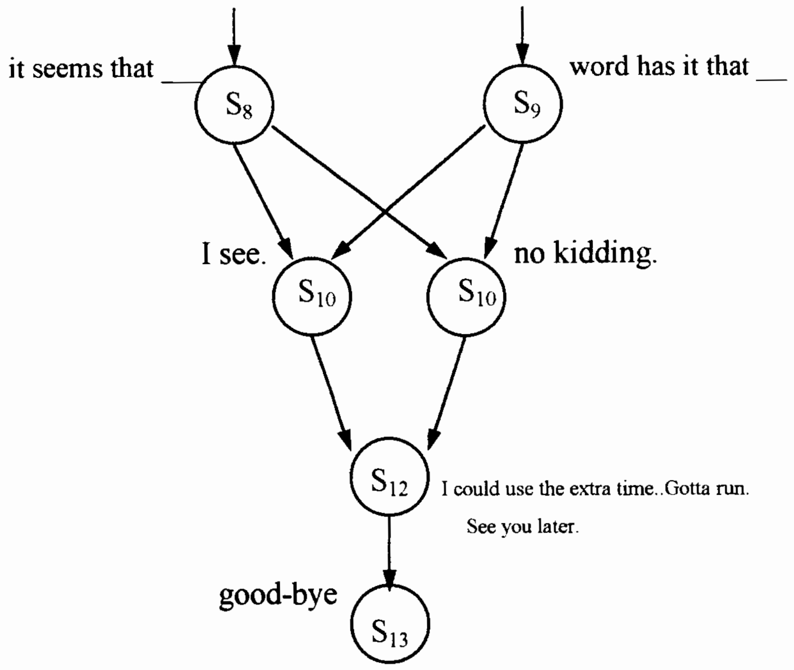


Figure 7. The main message and closing sections

3.8. Program details

The program consists of subroutines that call other modules (figure 8). The first module is the main control section of software that initializes all states and flags as well as creates the windowing environment. From the main control module, the dialog reads the responses from the terminal. The generation section prints out the output associated with each state. The analysis section determines the state of the response. The transition section enables the move to the next state. It is equivalent to the arc transitions in a finite automaton. The final section is the lexicon, which holds all of the lexical phrase data. The generation, analysis, and transition sections all access the lexicon.

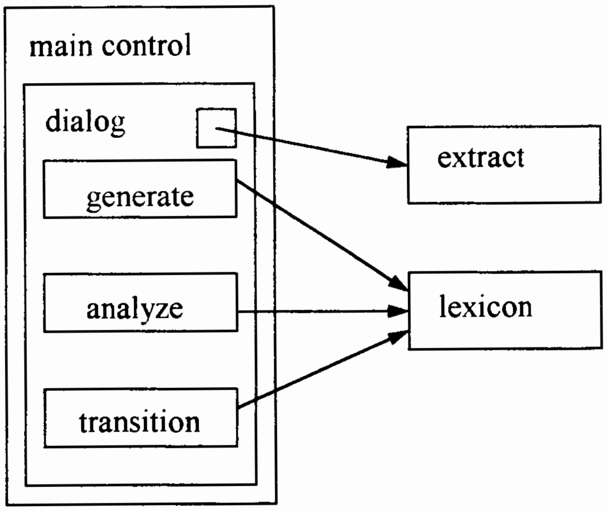


Figure 8. Structure of the program.

The main control module first initializes the state counter to zero. The state counter keeps track of the current state the program is in. This is important so that the proper sequencing through the state machine can be achieved.

The windows for the scenario area and the dialogue are set up. The scenario window explains the contextual background of the dialogue. In principle, the program could be set up for any number of contexts. Thus, an initial explanation of the context would be needed for the users of this program. The dialogue window is used for aesthetic purposes only, and is not necessary in program.

Next, a random number generator determines a number from zero to two, which will be used as the start state of dialogue. The random number generators offers some variety in the dialogue. As mentioned previously, If the number is a one, then the output is *hello*. If the number is two, then the program outputs *how are you doing?*. If the first random number is a zero, then the union of states 1 and 2 is outputted, i.e., *hello, how are you doing?.*

Once a dialogue has ended, the program asks the participant if he/she would like to start over. If the answer is yes, then the program loops to the beginning. If no, the program quits.

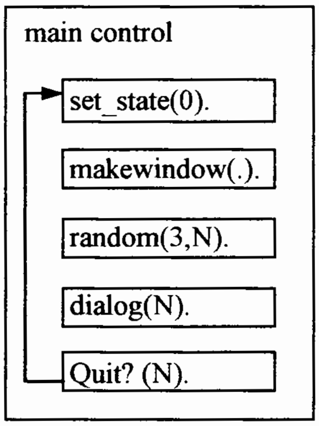


Figure 9. Organization of the main control section.

The dialog section first checks if the current state is the final state (i.e., state 13). If the current state is not the final state, the program will generate the lexical phrase, which is associated with the present state of the program. The lexicon is always accessed by the current state of the program. Once the lexical phrase is outputted, then the state counter is updated to the new state. The code for generating a lexical phrase is as follows:

```
generate(CurrentState):-              /* generates the lexical phrase */
  data(Frame,_,_,CurrentState),       /* access the lexicon */
  write(":| ", Frame, "\n"),          /* output the lexical phrase */
  set_state(CurrentState).            /* update the state counter  */
```

When the lexical phrase has been outputted to the terminal, the program reads the response that is typed in by the participant. Each response must be terminated with a period. The program uses the "extract" module to determine if each response is properly terminated with a period. This is particularly important for stacked lexical phrases, such as *I'm fine. and you?* and (in response to *how are you doing?*) *good. how are you?* The extract module splits up the lexical phrases, so that the lexicon can be accessed successfully with one lexical phrase. (The code for the extract module is in appendix B.)

Once the response has been read from the terminal, the program analyzes the input by comparing it to the lexicon. The analyze section takes the lexical phrase and matches it to a corresponding predicate in the lexicon. When there is a successful match, the state of the response is read, then used to transition to another state. The code for this module is as follows:

```
analyze(Response):-                  /* analyze the typed in response */
  data(Response,_,_,NewState),       /* access lexicon and get new state */
  transition(NewState).              /* transition to the new state */
```

The transition module uses the present state of the response to jump to the next state. The transitions are essentially jump arcs that move from one state to another state. In order to get variation in the program, the random number generator is used to access the next jump state. That is, a random number from zero to one is generated. If the number is zero, then the current state is set to S8. If the number is one, then the match

N=0 fails, and the program must backtrack and try the next clause that matches with *transition (1)*. Since there is another clause, the predicate is accessed, and the state is set to S9.

```
transition(1):-          /* transition for state S1  */
  random(2,N),           /* generate a random number from zero to one */
  N=0,                   /* if N=0 continue; N=1 then match fails. backtrack
                                & try again. */
  set_state(8).          /* set the current state to state S8. */

transition(1):-          /* since N=1 use this transition */
  set_state(9).          /* set the current state to state S9 */
```

The last part of the dialog module gets the current state of the program, after the transition has been made to a new state. The program then loops again to the beginning of the dialog module until the final state has been reached. Once the current state equals the final state (i.e., state 13), then the dialog module returns to the main control module. At this point, the state counter is cleared, so that the program can begin again.

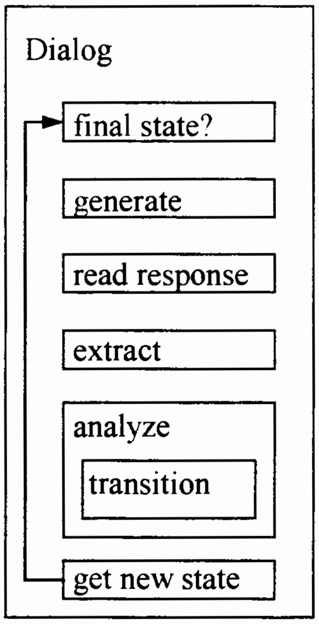


Figure 10. Dialog module.

Finally, the lexicon contains all of the lexical phrases for use in this program. There are two main lexical phrase structures in the lexicon.

1) data(lexical phrase, a_kind_of, phrase type, state).

2) data(lexical phrase, an_instance_of, original lexical phrase, state).

The main lexical phrases have the label *a_kind_of*. There is additional information, such as *phrase type* that ranges from greetings, response, understanding, etc. These parameters are not important to the program itself. The important parameters are the lexical phrase, and its state. That is, the number in the last column refers to the state that the lexical phrase is associated with.

Any alternatives of the main lexical phrases are designated with *an_instance_of* label. When the program reads a lexical phrase, it makes a match with the lexicon. If the initial search fails, it will try again to make a match with another lexical phrase. For example, given the input *no way*, the program will first search for any response with the corresponding labels. It will first find the lexical phrase *I see*, and attempt to match to it. Since this does not match the input, it will backtrack again for a corresponding match. On the second attempt, it matches successfully with the lexicon entry for *no way*.

Examples of the lexicon are as follows:

```
data("hello.", a_kind_of", "greeting", 1).
data("how are you doing?", a _kind_of, "greeting", 2).
data("good", a_kind_of, "response", 4).
data("it seems that", a_kind_of, "lexical phrase", 8).
data("the test is postponed till Tuesday.", a_kind_of, "filler", 8).
data("I see.", a_kind_of, "understanding", 10).
data("hi.", an_instance_of, "hello", 1).
data("how's it going?", an_instance_of, "how are you doing?", 2).
```

The lexicon has only one lexical phrase in each entry. This is designed to keep the lexicon small. In cases that require stacked lexical phrases, the program splits up any stacked phrases using the extract module. Once the lexical phrase has been split up, then each individual phrase can be used to match a corresponding predicate in the lexicon.

The context for this module can be changed by replacing the lexicon with any alternative phrases appropriate to that particular setting. This program can provide the L2 students of English the opportunity to practice lexical phrases in varying contexts with a variety of possible responses.

CHAPTER IV

DISCUSSION

Lexical phrases permit communicative fluency through efficient retrieval from memory. These prefabricated language structures have specific pragmatic functions. They enable speakers to focus on the converation in a top-down manner, rather than focusing on discrete lexical items. Lexical phrases are ideal for second language students of English to aid their conversation fluency (Nattinger and DeCarrico, 1992). The pedagogical application of lexical phrases for computers is the topic of this study.

The major problems associated with this program were: the constrained context or scenario; the lack of parsing capabilities; the mixed-initiative mode was implemented in only one section; the overall robustness of the program.

The dialog was kept short to curtail the problems that occur in NLP dialogue interfaces (i.e., indirect answers, anaphoric reference over sentence boundaries, sentence fragments, conversational patterns). Issues on resolving anaphoric reference over sentence boundaries, indirect speech acts, and fragments are beyond the scope of this project.

Although constrained contexts lead to stilted and repetitive conversations, the rational for it involves the difficulty in computionally modeling the belief systems and knowledge of the participants. That is, the background knowledge needed to generated dialogue in a complex environment is extensive. This will place a computational burden on the size of the lexicon and the processing time of a natural language processing system. Ringle and Bruce (1982) summarize the problems associated with constrained contexts.

> A major obstacle to a free-form, user-initiated dialogue system lies in the infinite variability of belief-models that users bring to a conversation. One approach to this problem is to explore a domain in sufficient depth to allow for a comprehensive description of all probable goals and beliefs (including erroneous beliefs) that a user might have in a given situation. By anticipating all of the relevant goal and belief states, the system is able to accommodate a wide range of user-initiated utterances (or written inputs). The dimension of user sensitivity is reduced, in essence, to a parsing problem --that is, the problem of mapping surface strings to a small, hierarchical goal tree (Ringle and Bruce, 1982, p. 215).

The only criteria for the computer-created dialog is its ability to produce realistic responses. This program does produce realistic dialog, but it is highly invariable. One major drawback to this study was the inability to implement any parsing capabilities into the program. Parsers extract information from sentences by breaking them up into its constituent parts. Thus, programs with parsing capabilities can achieve a semblance of "intelligence".

Since no parser was used, the program could not adequately accept any slot fillers by the human subject. For example, this program could not switch roles: the human subject is speaker A, the computer is speaker B. If the human subject filled the slot of a lexical phrase with "Word has it that Prof. Henry postponed the test till next week," the program should have been able to accept such an input provided that parsing capabilities existed. Without a parser, the lexicon must contain any and all anticipated responses by the human subject, which would be nearly impossible to accomplish. Ideally, the use of lexical phrases coupled with parsing capabilities would make an efficient system.

The first section of the program, the opening, involved mixed-initiative mode. That is, speaker A does not retain the initiative in the conversation, but alternates with speaker B. Natural language involves mixed-initiative mode since the participants are negotiating a conversation (Stenstrom, 1994). The complexity of mixed-initiative lies in its variability, which places an added computational load on the program. The program must process a greater number of choices and options. Thus, the software must sufficiently handle these options.

Conversation is far more complex than its representation as a collection of linear exchange structures. However, for its purposes, the program offers somewhat realistic dialogs for practice. (Appendix B contains one lesson plan to be used with this program.)

The difficulty in creating the lexicon was the infinite variability of the interaction. For example, the following dialogs were created by the program.

(1) A: hello.
B: hi.
A: word has it that the test is postponed till Tuesday.
B: no kidding.
A: we could use the extra time!...Gotta run. See you later.
B: good-bye.

(2) A: hello.
B: how are you?
A: I'm fine.... it seems that the test is postponed till Tuesday.
B: no way.
A: we could use the extra time!... Gotta run. See you later.
B: bye.

(3) A: hello. how are you doing?
B: good. and you?
A: okay.... word has it that the test is postponed till Tuesday.
B: I see.
A: we could use the extra time!... Gotta run. See you later.
B: see ya.

In terms of the responses by the computer, there is high degree of variability in the opening module . The main message offers only two lexical phrase choices: "word has it that ___" and "it seems that ____". The closing module has no variability.

The program could be improved in terms of the variability of its responses by programming more choices into the system. The degree of difficulty ranges from a single choice (e.g. as seen in the main section) to a mixed-initiative mode. The human subject has a high degree of variability in terms of typing in responses. The process of recognizing phrases is considerably easier than generating these phrases.

This program has many flaws, but with advanced use of parsers, faster processors, speech synthesis and recognition systems for computers, the prospects for improved CALL programs is exciting. Naturally, this can be extended to improving lexical phrase teaching programs. Further, the program itself is not very robust. That is, the program must accept any lexical phrase that matches exactly with the lexicon. Any accidentally carriage return or the pressing of any key would cause an error.

Future study could include several topics related to lexical phrases. A natural language processing study can be made based on a program that incorporates a parser with lexical phrases. The testing of the program can be studied in an actual classroom environment with more extensive contexts and scenarios.

In summary, the use of lexical phrases are adaptable for use in computer assisted language learning (CALL) programs. The communicative language teaching model would provide realistic dialogs. It offers more interesting exercises compared to traditional language drills. The problems of encoding contextual information with effective parsers in computer programs are still in a rudementary stage of research. As computer technology advances, the use of lexical phrases in CALL programs will provide an effective means to aid the communicative competence of second language learners of English.

BIBLIOGRAPHY

Austin, J. L. (1962). **How to do things with words**. Oxford: Clarendon Press.

Becker, J. (1975). The phrasal lexicon. In B. Webber-Nash & R. Schank (Eds.), **Theoretical issues in natural language processing 1.** Cambridge: Bolt, Beranek, and Newman.

Bobrow, D. G, Kaplan, R. M., Kay, M., Norman, D. A., Thompson, H., & Winograd, T. (1977) Gus, a frame-driven dialog system. **Artificial Intelligence, 8**, 155-173.

Bobrow, D. G., & Winograd, T. (1977). An overview of KRL, a knowledge representation language. **Cognitive Science, 1**(1), 3-46.

Bratko, I. (1991). **PROLOG: Programming for artificial intelligence** (2nd ed.). Reading: Addison-Wesley.

Brown, G., & Yule, G. (1983). **Discourse analysis**. Cambridge: Cambridge University Press.

Chandrasekaran, B. (1990). What kind of information processing is intelligence? In Partridge, D. & Y. Wilks (Eds.), **The foundations of Artificial Intelligence**. Cambridge: Cambridge University Press, .

Charniak, E. (1978). On the use of framed knowledge in language comprehension. **Artificial Intelligence, 11**, 225-265.

Charniak, E. (1982). Context recognition in language comprehension. In W. G. Lehnert & M. H. Ringle (Eds.), **Strategies for natural language processing** (pp. 435-454). Hillsdale:Erlbaum.

Cohen, P. R., Perrault, C. R., & Allen, J. F. (1982). Beyond question answering. In W. G. Lehnert & M. H. Ringle (Eds.), **Strategies for natural language processing** (pp. 245-274). Hillsdale:Erlbaum.

Cook, V. J. (1987). Designing CALL programs for communicative teaching. **ELT Journal, 42**(2),262-271.

Clocksin, W. F. & Mellish, C. S. (1994). **Programming in Prolog** (4th ed.). Berlin:Springer-Verlag.

Dore, J. (1975). Holophrases, speech acts and language universals. **Journal of Child Language, 2**, 21-40.

Dreyfus, H. L. (1981). From micro-worlds to knowledge representation: AI at an impasse. In J. Haugeland (Ed.), **Mind Design** (pp.161-204). Cambridge: MIT Press.

Dreyfus, H. L. (1988). Making a mind versus modelling the brain: Artificial intelligence back at a branchpoint. **Daedalus, 117(1)**, 15-43.

Gatbonton, E., & Segalowitz, N. (1988). Creative automatization: Principles for promoting fluency within a communicative framework. **TESOL Quarterly, 22** (3), 473-492.

Gal, A., Lapalme, G., Saint-Dizier & P., Somers, H. (1991). **Prolog for natural language processing**, Chichester: John Wiley & Sons.

Grice, H. P. (1975). Logic and conversation. In P. Cole & J. Morgan (eds.) **Syntax and Semantics** Vol. 3. New York: Academic Press.

Grishman, R. (1986). **Computational Linguistics**. Cambridge Univ. Press, London.

Hakuta, K. (1974). Prefabricated patterns and the emergence of structure in second language acquisition. **Language Learning 24**, 287-97.

Haugeland, J. (1985). **Artificial Intelligence: The Very Idea**. Cambridge: MIT Press.

Hayes, P. (1979) The logic of frames. In D. Metzing (Ed.), **Frame Conceptions and Text Understanding** (pp. 46-61). Berlin: Gruyter and Co.

Hobbs, J. R. (1982). Towards an understanding of coherence in discourse. In W. G. Lehnert & M. H. Ringle (Eds.)**, Strategies for natural language processing** (pp. 223-243). Hillsdale:Erlbaum.

Huang, J., & Hatch, E. M. (1978). A Chinese child's acquisition of English. In E.M. Hatch (Ed.), **Second language acquisition: a book of readings** (pp. 118-131). Rowley, MA: Newbury House.

Krashen, S. D. & Scarcella, R. (1978). On routines and patterns in language acquisition and performance. **Language Learning 28**: 283-300.

Last, R. W. (1989). **Artificial intelligence techniques in language learning**. Chichester: Ellis Harwood.

Leech, G. N. (1983). **Principles of Pragmatics**. London: Longman.

Levinson, S. (1983). **Pragmatics**. Cambridge: Cambridge University Press.

Lehnert, W. G., & Ringle, M. H. (1982). **Strategies for natural language processing**. Hillsdale: Erlbaum.

McClelland, J. L. (1989). **Explorations in parallel distributed processing: a handbook of models, programs and exercises.** Cambridge: MIT Press.

Mey, J. L. (1993). **Pragmatics: an introduction**. Cambridge: Blackwell.

Minsky, M. (1975). A framework for representing knowledge. In P. H. Winston (Ed.), **The Psychology of Computer Vision**. New York: McGraw-Hill.

Minsky, M. (1981). A frame for representing knowledge. In J. Haugeland (Ed.), **Mind Design** (pp. 95-128), Cambridge: MIT Press.

Nattinger, J. R. (1980). A lexical phrase grammar for ESL. **TESOL Quarterly, 19**(3), 337-344.

Nattinger, J. R., & DeCarrico, J. S. (1992). **Lexical Phrases and Language Teaching**. Oxford: Oxford University Press.

Obermeier, K. K. (1989). **Natural Language Processing Technologies in Artificial Intelligence: The Science and Industry Perspective**. Chichester: Ellis Horwood.

Parret, H. & Verschueren, J. (1992). **(On) Searle on conversation**. Philadelphia:John Benjamins.

Partridge, D., & Wilks, Y. (Eds.). (1990). **The foundations of artificial intelligence: A sourcebook**. Cambridge: Cambridge University Press.

Pereira, F.C.N., & Shieber, S. M. (1987). **Prolog and natural language analysis**. Menlo Park: CSLI.

Peters, A. M. (1977). Language learning strategies: Does the whole equal the sum of the parts? **Language, 53**, 560-573.

Peters, A. M. (1983). **The units of language acquisition**. Cambridge: Cambridge University Press.

Reichman, R. (1985). **Getting computers to talk like you and me**. Cambridge: MIT Press.

Riesbeck, C. K. (1982). Realistic language comprehension. In W. G. Lehnert & M. H.Ringle (Eds.), **Strategies of natural language processing** (pp. 37-54). Hillsdale: Erlbaum.

Ringle, M. H., Bruce, B. C. (1982). Conversation failure. In W. G. Lehnert & M. H. Ringle (Eds.), **Strategies for natural language processing** (pp. 203-221). Hillsdale: Erlbaum.

Rosenbaum, D. A., Weisler, S. E., & Baker-ward, L. (1987). **Cognitive Science**. Cambridge: MIT Press.

Sacks, H., Schegloff, E. & Jefferson, G. (1974). A simplest systematics for the organizatoin of turn-taking for conversation. **Language 50**: 696-735.

Searle, J. (1975). Indirect speech acts. In P. Cole and J. Morgan (Eds.), **Syntax and Semantics, Volume 3: Speech Acts**. New York: Academic Press.

Searle, J. (1992). Searle on Searle. In H. Parret and J. Verschueren (Eds.) **(On) Searle on conversation**. (pp. 50-64). Philadelphia: John Benjamins.

Schank, R. C. & Childers, P. G. (1984). **The cognitive computer**. Reading: Addison-Wesley.

Schegloff, E. (1992). Conversational analysis. In H. Parret and J. Verschueren (Eds.) **(On) Searle on conversation**. (pp. 32-48). Philadelphia: John Benjamins.

Sharples, M., Hogg, D., Hutchinson, C., Torrance, S., & Young, D. (1984). **Computers and thought: a practical introduction to artificial intelligence**. Cambridge:MIT Press.

Sperber, D. & Wilson, D. (1979). **Relevance: communication and cognition**. Cambridge: Harvard University Press.

Stenstrom, A. (1994). **An Introduction to Spoken Interaction**. London:Longman.

Sterling, L., & Shapiro, E. (1986). **The art of Prolog**. Cambridge: MIT Press.

Stillings, N. A., Feinstein, M. H., Garfield, J. L., Rissland, E. L., Rosenbaum, D. A., Weisler, S. E., & Baker-Ward, L. (1992). **Cognitive science: An introduction.** Cambridge: MIT Press.

Tsui, A. B. M. (1991). Sequencing rules and coherence in discourse. **Journal of Pragmatics, 15**, 111-129.

Underwood, J. H. (1984). **Linguistics, computers and the language teacher**. Rowley: Newbury House.

Waltz, D. L. (1982). The state of the art in natural language understanding. In W. G. Lehnert & M. H. Ringle (Eds.), **Strategies for natural language processing** (pp. 3-32). Hillsdale, N.J.: Erlbaum.

Warren, D. (1970). Phonemic restoration. **Scientific American, 7**, 45-65.

Weischedel, R. M., Voge, W. M., & James, M. (1978). An artificial intelligence approach to language instruction. **Artificial Intelligence, 10**, 225-240.

Weiskamp, K., & Hengl, T. (1988). **Artificial intelligence programming with Turbo Prolog**. New York: John Wiley.

Weizenbaum, J. (1966). Eliza--a computer for the study of natural language communication between man and machine. **Communications of the Association of Computing Machinery, *9***(1),36-45.

Weizenbaum, J. (1976). **Computer power and human reason: from judgement to calculation.** San Francisco: W.H. Freeman.

Yorio, C. A. (1980). Conventionalized language forms and the development of communicative competence. **TESOL Quarterly,14**(4), 433-442.

APPENDIX A

INSTRUCTIONS ON RUNNING THE PROGRAM

The program runs on any version of DOS operating system. To start the program, insert the disk into drive A, then enter:

```
A> lexical <hit enter>
```

To end the program at any time, enter "bye" as a response.

The possible responses for the program are as follows:

I.) Openings:

1.	hello.
1a.	hi.
2.	how are you doing?
2a.	how's it going?
2b.	how are you?
3.	good.
3a.	great.
4.	I'm fine.
4a.	fine.
5.	o.k.
5a.	so so.
5c.	fair.

6.	busy.
6a.	swamped.

II) Responses to the main message

-main message: it seems that the test is postponed till Tuesday.

word has it that the test is postponed till Tuesday.

-response:

7.	I see.
7a.	no kidding.
7b.	is that so?
7c.	you don't say.
7d.	no way.

III) Closing responses:

8.	good-bye.
8a.	bye.
8b.	so long.
8c.	see you later.
8d.	see ya.
8e.	later.

APPENDIX B

EXAMPLE LESSON PLAN

PURPOSE

This exercise helps learners to use lexical phrases in conversational settings. This is designed for first year second language learners of English.

MATERIALS

1) IBM PC or compatible with DOS 5.0 operating system.
2) Picture of two people in a hallway.

TIME

Ten minutes.

INSTRUCTIONS

1. The instructor should present the lexical phrase to the class. The pragmatic functions of the phrase as well as its meaning should be explained with a few examples.

2.. For homework, the students should work on the computer to practice the lexical phrase. Each student should be teamed with another student in pairs. Then, they should read the instructions and the list of lexical phrases.

3. Run the lexical phrase tutor by inserting the disk and by entering at the DOS prompt: **lexical**.

4. The student should read the scenario, then push the space bar when he/she is ready to begin.

5. The student should type in the lexical phrase responses that fit the situation.

6. When the dialogue has finished, the students should write down the script. For homework, they should practice the script and present it at the next class period with their partner. The student should try to figure out some alternative sentences that could be used with **word has it that __** or **it seems that __** and perform these in class. In other words, they should try to fill in the blanks with some other examples.

APPENDIX C

THE PROGRAM

```
/*      LEXICAL.PRO by G. Hirayama
        This program helps second language learners of English practice lexical phrases.
Lexical phrases are prepatterned phrases that aid communicative competency and helps
the student of English gain speaking fluency.

        The program is written in PDC Prolog version 3.30 and can be run on any DOS
based system.

        To run the program, enter at the prompt: lexical <enter>.    */

% ---------------------------------- THE PROGRAM -------------------------------------
nowarnings

/* Define the domains and initialize the database */
domains
 frame=string
 slot=symbol
 type=string
 datum=data(frame,slot,type,states)

database
 current_state(states)
 current_flag(states)
 data(frame,slot,type,states)

include "a:extract.pro"

/* This section defines the predicates in the program. PDC Prolog requires that all
predicates be defined at the start of the program. */

predicates
 run
 clear_state
 clear_flag
 set_state(states)
 get_state(states)
 get_flag(states)
 final_state(states)
 analyze(string)
 transition(states)
 generate(states)
 repeat
 dialog(integer)
 get_end(char)
/* The main control module */

clauses
 run:-
  set_state(0),                                          % initial the state counter to zero
  set_flag(0),                                           % initial the flag to zero
  repeat,
  makewindow(1,7,7,"LEXICAL PHRASE",1,1,22,78),          % make the outside window
  makewindow(2,48,7, "   PRACTICE   ", 5,5,10,70),        % make the dialogue window
  makewindow(3,48,7, "   The Scenario   ",16,10,5,60),    % make the scenario window
  write(" You are a college student. You are walking to a class in","\n",
  " Neuberger Hall. A classmate from another class walks by","\n",
  " and greets you briefly."),
  readchar(_),
  removewindow,
  random(3,N),
  dialog(N),                          % call the dialog module
  write("\n\n", "Quit? [Y/N]"),       % loop if N, quit if Y.
  readchar(C),
  get_end(C),
  clearwindow,
  removewindow,
  removewindow,
  run.

 get_end('y'):- !,
  exit.

 get_end('n'):- !.

/* predicates to check for final state and for current state  */
 final_state(13):- current_state(13),!.
 set_state(Sn):- asserta(current_state(Sn)).
 get_state(Sn):- current_state(Sn),!.

 clear_state:- retract(current_state(_)),fail.
 clear_state.
 set_flag(Sn):- asserta(current_flag(Sn)).
 get_flag(Sn):- current_flag(Sn),!.

 clear_flag:- retract(current_flag(_)), fail.
 clear_flag.

 repeat.
 repeat:- repeat.

/* The dialog module */
 dialog(N):-
  N<13,
  generate(N),
  write(":| "),
  readln(Response),
  extract(Response,NewS),
  analyze(NewS),
  get_state(N2),
  dialog(N2).


dialog(13):- !,          % if the program is at the final state, clear counter and flag.
 clear_flag,
 clear_state.

/* These predicates generate the responses in the program.  */
 generate(0):- !,
  data(Frame,_,"greeting",1),
  write(":| ", Frame),
  data(Frame2,_,"greeting",2),
  write(" ", Frame2, "\n"),
  set_state(2).

 generate(3):-
  current_flag(1),
  random(4,N),
  N1=N+4,
  data(Frame,_,_,N1),
  write(":| ",Frame,"..."),
  random(2,X),
  N2=X+8,
  data(Frame2,_,_,N2),
  data(Frame3,_,"filler",_),
 write(Frame2, Frame3,"\n"),
 set_state(N2).

generate(3):-
 current_flag(1),
 random(4,N),
 N1=N+4,
 data(Frame,_,_,N1),
 write(":| ",Frame, "..."),
 data(Frame2,_,_,9),
 data(Frame3,_,"filler",_),
 write(Frame2,Frame3),
 set_state(9).

generate(8):- !,
 data(Frame,_,_,8),
 write(":| ", Frame),
 data(Frame2,_,"filler",_),
 write(Frame2, "\n"),
 set_state(8).

generate(9):- !,
 data(Frame,_,_,9),
 write(":| ", Frame),
 data(Frame2,_,"filler",_),
 write(Frame2, "\n"),
 set_state(9).

generate(N):- !,
 data(Frame,_,_,N),
 write(":| ", Frame, "\n"),
 set_state(N).

/* These predicate analyzes the typed in responses by the student and determines the next
        state. */
analyze(S):-
 data(S,_,_,N),
 set_state(N),
 final_state(N).

analyze(S):-
 data(S,_,_,Curr_state),!,
 transition(Curr_state).
```

```
analyze(_):-
 write("Sorry...I can't understand a word you're saying..!"),nl.

/* TRANSITIONS: These transitions move from one state to another.  */

transition(1):-
 random(2,N),
 N=0,
 set_state(8).

transition(1):-
 set_state(9).

transition(2):-
 get_flag(Sn),
 Sn=1,
 set_state(3).

transition(2):-
 random(4,N),
 N1=N+3,
 set_state(N1),
 transition(N1).

transition(3):- !.

transition(N):-
 N<8,
 random(2,X),
 X=0,
 set_state(8).

transition(N):-
 N<8,
 set_state(9).

transition(N):-
 N=8; N=9,
 random(2,X),
 X=0,
 set_state(10).
```

```
transition(N):-
 N=8;N=9,
 set_state(11).

transition(10):-
 set_state(12).

transition(11):-
 set_state(12).


/* LEXICON: The lexicon for all the phrases used in the program. */

 data("hello.", a_kind_of,"greeting",1).
 data("how are you doing?", a_kind_of, "greeting", 2).
 data("and you?",a_kind_of,"greeting",3).

 data("good.", a_kind_of, "response", 4).
 data("I'm fine.", a_kind_of, "response", 5).
 data("okay.", a_kind_of, "response", 6).
 data("busy.", a_kind_of, "response", 7).

/* the main lexical phrases. The slot is filled by the declarative sentence "the test..." */
 data("it seems that ",a_kind_of, "lexical phrase", 8).
 data("word has it that ",a_kind_of, "lexical phrase", 9).
 data("the test is postponed to Tuesday.", a_kind_of, "filler", 8).

 data("I see.", a_kind_of, "understanding", 10).
 data("no kidding.", a_kind_of, "exclamation", 11).
 data("We could use the extra time!...Gotta run. See you later.",a_kind_of,
   "closing", 12).
 data("goodbye.", a_kind_of, "closing", 13).

 data("hi.", an_instance_of, "hello", 1).
 data("how's it going?", an_instance_of, "hello",1).
 data("how are you?", an_instance_of, "how are you doing", 2).
 data("great.", an_instance_of, "good", 4).
 data("fine.", an_instance_of, "I'm fine.", 5).
 data("so so.", an_instance_of, "okay", 6).
 data("fair.", an_instance_of, "okay", 6).
 data("swamped", an_instance_of, "busy", 7).

 data("is that so?", an_instance_of, "I see", 10).
data("you don't say.", an_instance_of, "I see", 10).
data("no way.", an_instance_of,"no kidding",11).

data("bye.", an_instance_of,"closing",13).
data("see ya.",an_instance_of,"closing",13).
data("so long.",an_instance_of,"closing",13).
data("see you later.",an_instance_of,"closing",13).
data("later.", an_instance_of, "closing", 13).


/* ------------------------------------ extract.pro------------------------------------------------
   This program separates stacked lexical phrases so that they can be properly accessed
        from the database.      */

domains
 tokenlist=symbol*
 states=integer

predicates
 extract(string,string)
 search(string,char,integer,integer)
 findc(string,char,integer,integer)
 check_target(string,string,string)
 rightx(string,string,integer,integer)
 setparam(integer,integer,integer)
 set_flag(states)

clauses
 extract(S,S2):-
  data(S,_,_,N),
  N=2,
  concat(".",S,S1),
  set_flag(1),
  search(S1,'.',Pos,Size),
  rightx(S1,Target,Pos,Size),
  check_target(S1,S2,Target).

 extract(S,S1):-
  search(S,'.',Pos,Size),
  rightx(S,Target,Pos,Size),
  check_target(S,S1,Target).

check_target(S,S1,""):-
```

```
 S1=S,!.
check_target(_,S1,T):-
 search(T,' ',1,Size),
 rightx(T,T2,1,Size),
 S1=T2,
 set_flag(1).

check_target(_,S1,T):-
 S1=T.

search(Src,C,Pos,Size):-
 str_len(Src,Size),
 findc(Src,C,Pos,Size),!.

findc(Str,_,_,_):-
 str_len(Str,0),fail.

findc(Str,C,Pos,Size):-
 frontchar(Str,Fc,Rest),
 C=Fc,
 str_len(Rest,Subl),
 Pos=Size-Subl.

findc(Str,C,Pos,Size):-
 frontchar(Str,_,Rest),
 findc(Rest,C,Pos,Size).

rightx(Src,Trg,N_pos,Size):-
 setparam(N_Pos,Size,C_pos),
 frontstr(C_Pos,Src,_,Trg).

setparam(N_pos,Size,C_Pos):-
 N_pos <= Size,
 N_pos >= 1,
 C_pos=N_pos,!.

setparam(N_pos,Size,C_pos):-
 N_pos>Size,
 C_pos=Size-1,!.

setparam(N_pos,_,C_pos):-
 N_pos<1,
 C_pos=0.
```