The Correction of Ill-Formed Input using History-Based Expectation with Applications to Speech Understanding

Pamela K. Fink

Southwest Research Institute
6220 Culebra Road
San Antonio, TX 78284

Alan W. Biermann

Department of Computer Science
Duke University
Durham, NC 27706

A method for error correction of ill-formed input is described that acquires dialogue patterns in typical usage and uses these patterns to predict new inputs. Error correction is done by strongly biasing parsing toward expected meanings unless clear evidence from the input shows the current sentence is not expected. A dialogue acquisition and tracking algorithm is presented along with a description of its implementation in a voice interactive system. A series of tests are described that show the power of the error correction methodology when stereotypic dialogue occurs.

This material is based upon work supported by The National Science Foundation under Grant number MCS 7904120 and Grant number MCS 8113491 and by the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under Grant 81-0221.

1 Introduction

In an environment where stereotypic discourse commonly occurs, the repetitiveness and predictability of the interactions may enable a machine to effectively anticipate some inputs. For a speech understanding system, such anticipation can greatly enhance the processor's capabilities for error correction so that proper action will take place despite inaccuracies at the voice recognition phase. This paper is concerned with the automatic construction of a model of user behaviors in typical interactions and the use of such a model in the correction of misrecognition errors.

It is assumed that a user approaches the machine in a typical application with a problem to be solved. He or she inputs a series of sentences requesting action or information that will lead to a solution and then leaves when the task is complete. In the early examples of such an interaction, the machine will have little or no expectation and will be dependent on its basic capabilities for understanding and carrying out commands. However, if repetitive behaviors occur, the processor will effectively use them to anticipate inputs and correct errors. This will enable the user to speak less precisely and more quickly while still achieving reliable performance.

Such repetitive behaviors may occur within a single dialogue where a user may utter sentences with similar meanings again and again (as in "Is there a plane on Thursday? What time does it leave? Is there one on Friday? When does it leave?"). They may also occur when a given dialogue resembles earlier ones. The expectation system will thus continuously monitor inputs, looking for repetition. If no repetitious behavior occurs, the natural language processor is allowed to proceed without intervention in handling a dialogue. However, if repetitiveness is detected, the expectation system will supply the processor with anticipated behaviors which can be used to help remove uncertainties in sentence recognition when they occur.

In the following sections, an overview of the history-based expectation system is given. Then a representation for user behaviors is described, followed by an algorithm for creating and tracking such models along with a meth-

0362-613X/86/010013-36$03.00

od for using them in error correction. Finally, an implementation of this methodology is described in the domain of speech recognition and results from a series of tests investigating the system's performance in various situations are presented.

2 An Overview of the History-Based Expectation System

The general goal of the history-based expectation system is to merge a series of dialogues, each of which consists of a sequence of sentences, into a more general dialogue that reflects the patterns that exist between and within the separate dialogues. Thus, the expectation system must:

- save incoming dialogues,
- find patterns between and within these dialogues so that they can be merged into a more general dialogue which becomes a formula for a more general situation, and
- use this information to help predict what will be said by a user in a given situation.

This ability to predict what might be said by a user can help error correct what is input to the natural language system through errorful means, such as a voice recognizer. We will call this ability **expectation.** Figure 1 shows an overview of the structure of the history-based expectation system. Expectation is acquired at two levels, the sentence level and the dialogue level. A special parser, called the expectation parser, is used to analyze at the sentence level. The expected dialogue is a data structure used to store the history-based expectation that is acquired using an expectation acquisition algorithm. This constitutes the dialogue level.

As each sentence is entered into the system, such as through a speech recognition device, it is parsed and a meaning representation is produced and saved by an expectation acquisition algorithm in the expectation module (see 1 in Figure 1). The parse is also output for use in the next step in the system's processing of the sentence. This process builds a sequence of sentence meanings, which are then incorporated into an expected dialogue (see 2 in Figure 1). After an expected dialogue is partially or completely built, the expectation module attempts to determine where the user is in a given dialogue using information from the expected dialogue and the current parsed sentence (see 1 and 3 in Figure 1). If it succeeds, it creates and transmits (see 4 in Figure 1) an expected sentence set to the expectation parser. The expectation parser will then use this information to improve its ability to recognize the next incoming sentence.

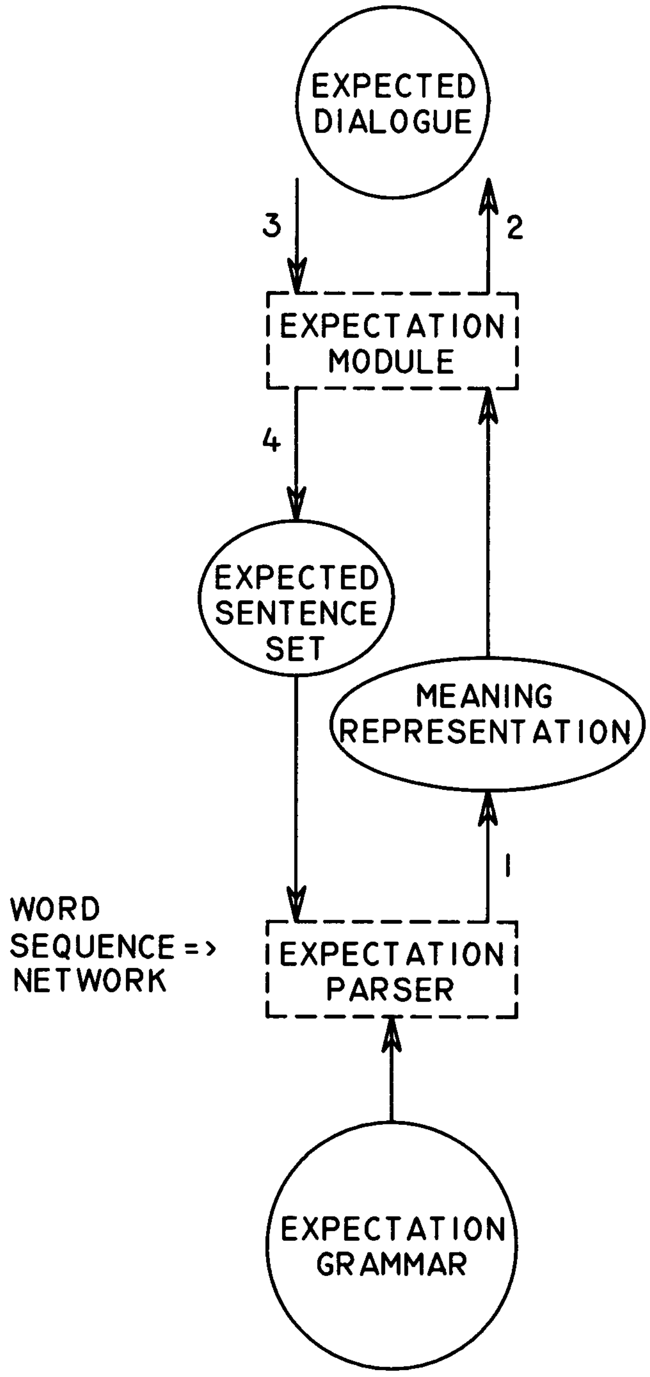

Figure 1. Overview of the history-based expectation system.

3 A Representation for User Behaviors

Suppose a user inputs the following sequence:

Sentence	*Label*
Display my mail summary for today.	S1
Show me this letter. (with touch input)	S2
(the letter appears on the screen)	
Remove this letter.	S3
Display the letter from JA.	S4
(letter appears on the screen)	
Delete it.	S5
Log off.	S6

We denote the meaning of each sentence Si with the notation M(Si). The exact form of M(Si) need not be discussed at this point; it could be a conceptual dependence graph (Schank and Abelson 1977), a deep parse of Si, or some other representation. A user behavior is represented by a network, or directed graph, of such meanings. At the beginning of a task, the state of the interaction is represented by the start state of the graph. The immediate successors of this state are the typical opening meaning structures for this user, and succeeding states represent, historically, paths that have been followed by this user.

It is important that if two sentences, Si and Sj, have approximately the same meaning this should be clear in the representations M(Si) and M(Sj). Our algorithm, described below, merges two meanings M(Si) and M(Sj) into a single node in the behavior representation if they
– are sufficiently similar, and
– appear in similar contexts.
Thus, in the above example it would appear that M(S3) and M(S5) play similar roles and could be represented by one structure: after a letter is read, one might expect to see it deleted.

Often, two commands will be similar except for the instantiation of certain constituents. This is the case in sentences S2 and S4, which request the display of, respectively, the message indicated by a touch and the letter from JA. Again, it is desired to represent such similar meanings in a behavior graph with a single node if they appear in similar environments. Thus, a routine will be needed to find a generalization of two such sentences that can represent their common meaning. In the example, the generalization of S2 and S4 might be "display ⟨LETTER⟩" where "⟨LETTER⟩" is a noun group referring to a letter.

In tracking a dialogue, we may arrive at a node in the behavior graph with meaning M1. This means a command is expected with meaning M2 that is either identical to, or a special case of, M1. If such an M2 is input at this time, we will say that M1 predicts M2 and define the predicate:

Predicts(M1, M2) = true if and only if meaning M1 is identical or similar to M2.

It is quite possible, as with M(S2) and M(S4) above, that a common generalization can be found for two sentences that appear in similar contexts. Then one will be able to merge them into a single node in the behavior graph. Thus, it is necessary to have a predicate to check whether these conditions hold and a function to find the desired generalization. The following two routines do this:

Mergeable(M1, M2) = true if and only if an M can be found such that Predicts(M, M1) and Predicts(M, M2).

Merge(M1, M2) yields a meaning M that is identical to, or a generalization of, M1 and M2.

A user behavior is represented as a network of sentence meanings with transitions from one meaning to another that indicate traversals observed in actual dialogues and their frequencies. For example, the above six-sentence sequence could be represented as shown in Figure 2. Each node i has a meaning Mi and a count Ci, which gives the number of times in observed dialogues this node has been visited. The integer on each transition gives the number of times it has been traversed in observed dialogues.

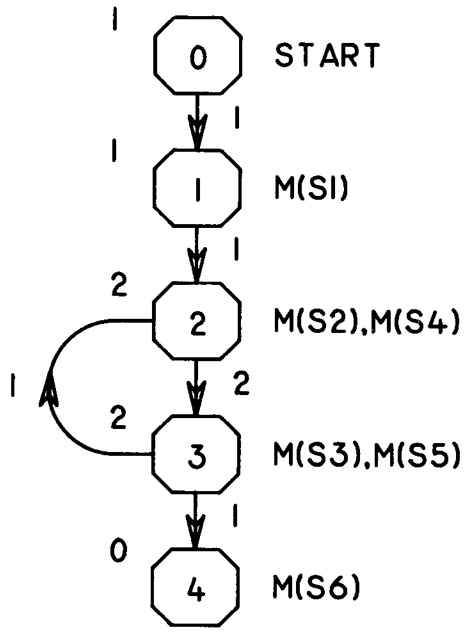

Figure 2. Modelling the user's behavior.

More formally, a behavior graph B will consist of a set of nodes named 0, 1, 2, 3, ..., bsize-1. Each node i will have its associated Mi and Ci and the first node will have a special meaning M0 = 'start'. The transitions will be represented as triples (i, j, k) where the traversal is from node i to node k and has been observed j times. The example six-command sequence would be represented by the nodes 0 through 4 with Mi's and Ci's as shown and with the triples

{(0,1,1) (1,1,2) (2,2,3) (3,1,2) (3,1,4)}.

Notice that the observed probability of crossing transition (i, j, k) is j/Ci, a fact that is used by the expectation parser.

4 The Expectation Model Building and Tracking Algorithm

It is desired to have an algorithm to monitor the discourse, collect the history of inputs, and invoke expectation when any kind of repetition occurs. Such an algorithm is described below. To do so, however, some additional notation is needed:

current = an integer giving the state number in B corresponding to the most recently recognized sentence.

bsize = the total number of states in B.

E(i) = {k | j > 0 (i, j, k) is in B}, the set of successor states to state i, also called the expected sentence set of i.

P(S, E(current)) = The result of the expectation parser with input S and E(current), where S is the current input sentence which may have errors, and E(current) is a set of expected meanings in B, the successors of node current. The result or output of the parse of sentence S is its meaning M(S).

The behavior graph B begins with one state numbered "0" and with M0 = start, C(0) = 0. Thus, the size of the graph is bsize = 1 and the most recently recognized sentence is assumed to be this start state, current = 0.

Suppose that the first sentence in the above sample dialogue is read:

S1 = "Display my mail summary for today."

Then the processor will begin with no expectation since E(0) is currently the empty set, and find

M(S1) = P(S1, {}).

This will result in the creation of a second state in B with the following statements:

```
Create a NEW NODE:
Put(current, 1, bsize) into B;
                              (a transition to the new state is created)
C(current) := C(current) + 1;  (state 0's count is incremented)
current := bsize;          (the new state is now the current state)
M(current) := M(S);         (the new state's meaning is recorded)
C(current) := 0;  (this state has not yet been visited and exited)
bsize := bsize +1;              (the size of graph B is incremented)
```

Thus, the first two states shown in Figure 2 will exist with the single transition (0, 1, 1). Sentence S2 and S3 result in similar processing, the addition of states 2 and 3, and the creation of transitions (1, 1, 2) and (2, 1, 3) as shown in Figure 3.

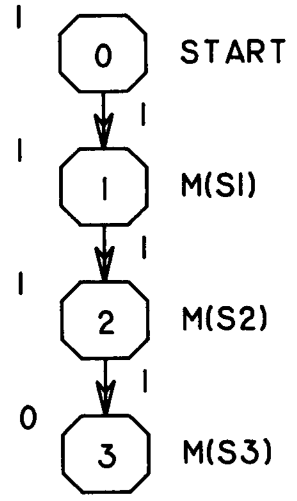

Figure 3. Constructing the behavior graph.

The input sentence will yield a different action, however, if its meaning M(S) is determined to be mergeable with the meaning of an existing node Mk on the graph. While the details of mergeability have not yet been discussed, let us assume for the current example that M(S4) is mergeable with M(S2). Then a new meaning will appear in the graph that is a generalization of these two, Merge(M(S2), M(S4)), and a graph transition will be built to this new meaning. Transfer to the existing meaning Mk would proceed as follows:

```
C(current) := C(current) + 1;
Mk := Merge(Mk, M(S));
Put(current, 1, k) into B;
current := k;
```

Figure 4 shows the updated graph. At this point, current = 2, and the expectation set, E(2), is non-empty for the first time. So, now we compute P(S5, {M3}), meaning that S5 is read with the expectation that its meaning will be "remove this one". Given this expectation, the parser will prefer any transitions down paths that lead to some paraphrase of this sentence and, unless the system clearly recognizes that something else has been said, a sentence meaning "remove this one" should be recognized. If it is, then current will be advanced to this expected node. In general, there may be several expected sentence meanings, and the processor will select the one most similar to the incoming utterance unless that sentence is clearly not any member of the expected set.

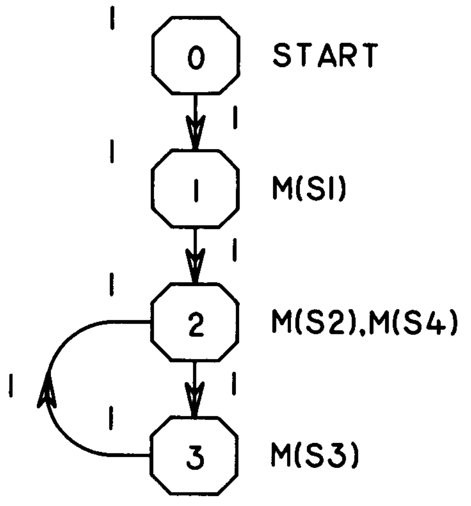

Figure 4. Merging M(S2) and M(S4).

Thus, if a successor k to the current state predicts the incoming sentence, we track that successor. Tracking the expected meaning Mk would proceed as follows:

```
C(current) := C(current) + 1;
Mk := Merge(Mk, M(S));
Increment r in (current, r, k);
current := k;
```

Figure 5 shows the result.

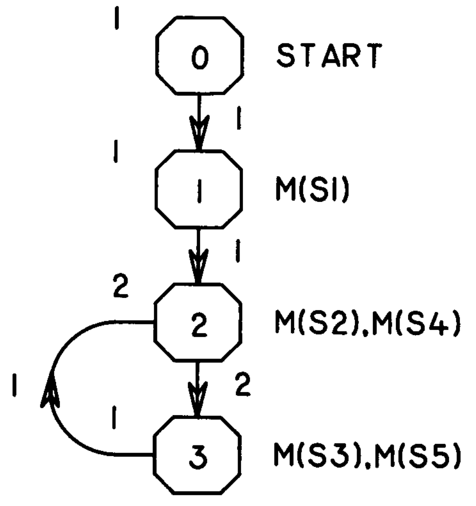

Figure 5. Merging M(S3) and M(S5).

The final sentence S6 in the dialogue will cause the creation of a termination state and complete the graph of Figure 2. The behavior graph creation and tracking algorithm is thus the collection of the above code segments:

```
if no behavior graph B exists then
    begin
    bsize := 1;
    M0 := start;
    C0 := 0;
    end;
else
    load B;
current := 0;
repeat
    begin
    read input sentence S;
    M(S) := P(S, "Mk | k in E(current)");
    if Predicts(Mk, M(S)) where k in E(current) then
        begin
        C(current) := C(current) + 1;
        Mk := Merge(Mk, M(S));
        Increment r in (current, r, k);
        current := k;
        end;
    else
        if Mergeable("Mk | k = 1 and/or 2 and/or ...
            bsize-1", M(S)) then
            begin
            C(current) := C(current) + 1;
            Mk := Merge("Mk | k = 1 and/or 2
                and/or ... bsize-1", M(S));
            Put(current, 1, k) into B;
            current := k;
            end;
        else
            create a NEW NODE;
    end;
until M(S) is a dialogue termination.
```

This code creates a finite state model of the dialogue based on equivalence or similarity classes defined by the functions Predicts, Mergeable, and Merge. As will be discussed in the next section, similarity classes are based not only on the similarity of the sentences themselves, but also on the environment in which they occur. Thus, there is only one state for each such similarity class in the finite state model created.

When the user enters the system again, this algorithm can be reinvoked using the existing B graph. If the next dialogue is very similar to a previous one, then the expectation dialogue will powerfully support error correction. If the next dialogue has little resemblance to previous ones, then no expectation will be available, and the user will be dependent on basic processor recognition capabilities.

This section has given an overview of the approach to history-based expectation processing. The details of the method are dependent on how the functions P, Predicts, Mergeable, and Merge are implemented. The following sections describe our implementation, which was used to investigate the viability of this approach and the performance it can achieve.

5 An Implementation

The usefulness of the methodology described above was tested in the implementation of a connected speech understanding system. An off-the-shelf speech recognition device, a Nippon Electric Corporation DP-200, was added to an existing natural language processing system, the Natural Language Computer (NLC) (Ballard 1979, Biermann and Ballard 1980). The expectation system provided the intermediate processing between the errorful output of the speech recognizer and the deep semantics of NLC. The resulting speech understanding system is called the Voice Natural Language Computer with Expectation (VNLCE, Fink 1983). [The current system should be distinguished from an earlier voice system (VNLC, Biermann et al. 1985), which had no expectation and which handled discrete speech where a 300 millisecond pause must follow each word.]

It should be emphasized, of course, that the central issue here is the study of expectation mechanisms and the details of the design decisions could have been made in rather different ways. Thus one could have implemented expectation error correction with a typed input system or with a speech input system that integrates voice signal processing with higher level functions in a way not possible with a commercial recognizer. This implementation shows only one way in which the functions P, Predicts, Mergeable, and Merge can be constructed to achieve expectation capabilities. The conclusion of this paper is that in this particular situation substantial error correction is achieved, and thus one may suspect that similar results can be achieved in other applications.

The implementation, as in the overview of the general system presented in section 2, consists of two major parts, an expectation parser and an expectation module, and their respective data structures. The expectation parser embodies the function P, while the major functions of the expectation module are Predicts, Mergeable, and Merge. An expected sentence set, E(current), along with the most recent input sentence S, are inputs to the expectation parser P. The expectation parser P uses these two inputs to determine the meaning M(S) of the input sentence S. Thus, M(S) is a deep parse of S. The function Predicts determines if one of the sentences in E(current) predicts M(S). If so, then M(S) is merged with this sentence meaning and dialogue tracking is begun from that point. Otherwise the function Mergeable determines how "similar" M(S) is to any other sentences in the expected dialogue. In this implementation, the function Mergeable is actually much more cautious about determining whether or not a set of sentences should be merged. For the implementation, if Mergeable determines that certain nodes in the expected dialogue are mergeable with M(S), then it adds the successors of these nodes to E, creating an expanded expected sentence set. Then, if the next sentence input is predicted by one or more of these sentences, they are merged through the action of Predicts and Merge.

5.1 The Expectation Parser

The purpose of the expectation parser in this implementation of a speech understanding system is to take input from the scanner and the expectation module, and use this information to determine what was said by the user. Thus, during the parsing process, the expectation parser must reconcile the sequence of words input from the scanner with the expected sentence set from the expectation module, or determine that the scanner input is not like anything that was expected and, thus, ignore expectation. In this way, the expectation parser parses from two inputs. It is constantly trying to maintain an equilibrium between the input from the scanner and the input from the expectation module. This balancing is kept in line by a set of rating factors that are used during the parsing procedure to help guide the search for a reasonable sentence structure. These rating factors, at times, will be referred to as probabilities in the following discussion. However, in reality, the ratings are one thousand times the values of the logarithms of numbers between 0 and 1. Thus, the ratings span the values −999 to 0, where 0 is equivalent to a probability of one. These ratings are computed this way because they remain integral and still fairly accurately represent the correct values. Also, they can simply be added and subtracted rather than multiplied and divided in the hundreds of calculations required for a single sentence parse.

The expectation parser uses an ATN-like representation for its grammar (Woods 1970). Its strategy is top-down. The types of sentences accepted are essentially those accepted by the original NLC grammar, imperative sentences with nested noun groups and conjunctions (Ballard 1979). An attempt has been made to build as deep a parse as possible so that sentences with the same meaning result in identical parses. Sentences have the same "meaning" if they result in identical tasks being performed. The various sentence structures that have the same meaning we call **paraphrases.** We have studied the following types of paraphrasing:

1) WORD <=> WORD
'entry' <=> 'number'

2) ADJ NOUN <=> NOUN QUALIFIER
'positive entries' <=> 'entries which are positive'

3) NOUN NUMBER <=> DET ORDINAL NOUN
'row 2' <=> 'the second row'

4) CLASSIFIER NOUN <=> NOUN of/in CLASSIFIER
'the row 1 entries' <=> 'the entries in row 1'

5) EQUIVALENT SETS
'row 1' <=> 'entries in row 1'

6) QUANTIFIERS
'all (of) (the) entries' <=> 'the entries'

7) CONJUNCTION OF NOUNS
'double rows one and two' <=> 'double row one and row two'

8) DEFAULT CONTEXT
'the rows' <=> 'the rows in matrix 1'

9) NAMES
'column 1' <=> 'testA'

10) PRONOUNS
'it' <=> 'row 1'

11) ORDINAL NOUN <=> NOUN X in COLUMN or ROW Y
NOUN NUMBER <=> NOUN X in COLUMN or ROW Y
'sixth entry' <=> 'entry 2 in column 3'
'entry 6' <=> 'entry 3 in row 2'

12) NUMBER <=> ENTRY X
'9.75' <=> 'entry 3'

13) WORD <=> {WORDS}
'double' <=> 'multiply by two'

14) CONJUNCTION OF VERBS
'double row two and zero matrix one.'
<=> 'double row two. zero matrix one.'

It is obvious from this list that there are varying levels of paraphrasing. Some arise at the vocabulary level (number 1), some at the syntactic level (numbers 2, 3, 4, 5, 6, and 7), some at the semantic level (numbers 8, 9, and 10), some at the current world level (numbers 11 and 12), and some at a combination of levels (numbers 13 and 14). Some are domain dependent, especially at the vocabulary level such as entry <=> number. Others are not, such as ADJ NOUN <=> NOUN QUALIFIER. Those that only require knowledge of the vocabulary or of the grammar are implemented in the current history-based expectation system. This means that paraphrases one through seven are handled currently as part of the parsing process itself. The last seven may be dealt with at some future date. However, they are somewhat more complicated because they require temporal-type knowledge such as the current referent of a pronoun or the current size of a matrix. The lexical and grammatical paraphrases, on the other hand, will always have the same meaning, regardless of the current state of the world. By handling the seven lexical and syntactic paraphrases, a stored parse can aid in recognizing many sentences with the same "meaning" but different surface structures.

To simplify representation of the parser output we have developed a special notation to indicate the deep parse of a sentence. For example, the parse of the sentences:

Double the positive row 1 entries.
Double the positive entries in row 1.
Double the row 1 entries which are positive.
Double the entries in row 1 which are positive.

is notated as:

Double (entries (positive) (r1))

The mechanism for using the expectation information during parsing is built into the ATN-like network. The parser receives from the scanner a sequence of **word slots.** These word slots are defined by the speech recognition system based on the sequence of words it recognized. Thus, there could be missing or extra word slots due to errors made during speech recognition. To each word slot the scanner adds other possible words based on what words the system tends to confuse. The scanner also rates the possibilities for each word slot by the same scale discussed previously. During parsing, the parser creates a template that represents the parse of the sentence input. This template contains slots that represent the parts of a sentence such as verb, adjective, and headnoun. At each point in the parse of a sentence, when the expectation parser is trying to determine what the role of the current word slot is in the sentence, five different attempts are made to use the current word slot as needed to fill the template slot at the current point in the grammar network. These are:

- ADV (advance): Find a word in the current word slot from the scanner output that will fit the needs at this node in the grammar. If such a word cannot be found, try choice 2.
- EXPADV (expectation advance): Look at the parse of the current expected sentence to see if the template slot that the parser is currently trying to fill is filled in the expected sentence. If so, copy the value in the template slot from the expected sentence to the current parse, ignoring the word slot from the scanner. Otherwise, try choice 3.
- SKIPWORD: Skip the current word slot from the scanner output, filling the corresponding parser template slot, when appropriate, with a NIL value to indicate that a word has been skipped and that it was assumed to have the function associated with the template slot. If the parse fails later on, and the parser backs up to this point, try choice 4.
- EXTRAWS (extra word slot): Assume that the word slot from the scanner is an extra one due to an error in recognition. Skip this word slot and again try choice 1. If failure occurs, try choice 2. Finally, if failure again occurs, try choice 5.
- LOSTWS (lost word slot): Assume that the needed word slot from the scanner is lost due to an error in recognition. Without advancing to the next scanner word slot, try step 2 again. If this fails, then fill the parser template slot, when appropriate, with a NIL value to indicate that a word has been lost and that it was assumed to have the function associated with that template slot. Remain at the current scanner word slot so that it can again be evaluated for a different function.

An example piece of the parser network is shown in Figure 6. The five kinds of error correction were hand coded into each network so that the special character-

```
 formatted routine FILLADJ:

4257   START
4267      ADV
4272      |   CHEK PART ADJ
4279      |      FILLSLOT ADJECTIVE QUOTE
4556      |         EXPCOMP ADJ
4286      |            RET
4551      EXPADV
5214      |   EXPCHEK ADJECTIVE
4885      |      COPYSLOT ADJECTIVE
4750      |         COPYWORD ADJECTIVE
          |            goto 4556
          goto 4556
4756      SKIPWORD LOG7
4762      |   FILLSLOT ADJECTIVE NIL
4769      |      COPYWORD QUOTE NIL
          |         goto 4556
5036      EXTRAWS LOG7
          |   goto 4267
          |   goto 4551
5043      LOSTWS LOG7
              goto 5214
              goto 4762
```

Figure 6. An example parse net.

istics of each grammatical structure could be accounted for individually. Thus in some cases, certain error correction alternatives were checked immediately while in others it was wiser to determine whether normal processing would fail at deeper levels before attempting those same corrections. The network represents a tree structure which is searched by the expectation parser. Succession in the network is represented by the parent-child relationship, which is indicated in Figure 6 by indentation. Thus, the node containing the command ADV is the parent of the node containing the command CHEK PART ADJ, and so is succeeded by it. Should a command fail, the parser backs up to the parent node of the node that has just failed. Thus, if a check for an adjective in CHEK PART ADJ fails, control will back up to the node containing ADV. Choice is represented by the sibling relationship which is indicated in Figure 6 by the vertical lines connecting nodes. Thus, ADV, EXPADV, SKIPWORD, EXTRAWS, and LOSTWS are all siblings in the tree network and are choices that the parser can make when parsing a sentence. Note that, in this case, these five choices represent the five possible attempts that are made in trying to parse a word slot that were discussed above. A choice is made by picking the siblings in the order in which they appear in the network. Thus, when the CHEK PART ADJ fails and control backs up to ADV, the expectation parser will back up to the START node and then take the second choice, EXPADV, and attempt to proceed down that chain of commands.

The scoring mechanism within the parser serves to aid in the evaluation of the alternative paths during the parse process and the pruning of improbable choices. A typical spoken input to the system is

"add row one to row two"

and the speech recognition machine will often return such errorful output as

"and row * to row".

The asterisk indicates that the device guesses the existence of a word but has failed to identify it.

The parser must be able to extract the user's original intent and its operation is guided by rating factors which evaluate the quality of the path through the parser, the word selection, the level of agreement with expectation, and the self consistency (or compatibility) of the sentence. These individual ratings work as follows:

1) The Transition Value
Every time the parser moves over a SKIPWORD, EXTRAWS, or LOSTWS command a charge is made to the value of the transition. Normally, a transition does not cost anything, but each SKIPWORD, EXTRAWS, and LOSTWS executed results in a lowering of the transition's value. This charge is made for the rest of the parse unless the SKIPWORD, EXTRAWS, or LOSTWS is backed over. This charge can be seen in the sample grammar net appearing in Figure 6 after the words SKIPWORD, EXTRAWS, and LOSTWS. The charge in this example for each of the three commands is 1000*log[0.7] = −35.

2) The Word Value
We define the **synophones** of a given vocabulary word to be the words a user might speak that could possibly be recognized as that word. Because of the nature of the dynamic programming algorithm in the NEC machine, it yields only one guess at each word slot. So it is necessary for our software to provide the set of synophones for each guessed word. This, in effect, simulates the situation where the speech recognition device provides a larger number of possible matches. Thus, in the case of the above recognizer output, the following synophones would be produced to represent the sequence of possible words spoken:

word slot	word	rating
0	and	1000*log[1.0] = 0
	add	1000*log[0.8] = −22
1	row	1000*log[1.0] = 0
	rows	1000*log[0.8] = −22
2	*	1000*log[1.0] = 0
3	to	1000*log[1.0] = 0
	two	1000*log[1.0] = 0
	into	1000*log[0.8] = −22

4	row	1000*log[1.0]	=	0
	rows	1000*log[0.8]	=	−22
5		1000*log!1.0!	=	0

Each alternative word is given a rating. The words selected by the recognizer are given maximum ratings and alternatives are given lower values. If two words have the same pronunciation as with *to* and *two*, they are given the same values.

3) The Expectation Value
This value is based on whether or not there is an expected sentence, how well the current parse is matching the current expected sentence from the expected sentence set, and how much the current parse is using this expected sentence. Whenever a slot is filled by the parser, it is compared with the corresponding slot in the expected sentence. If they do not match, the expectation value decreases, otherwise the expectation value remains the same.

4) The Compatibility Value
This value differs from the other three in that it is simply true or false. Verb-operand, noungroup-noungroup, and expectation are checks made during the parse. If compatibility fails, then the expectation parser backs up, otherwise it continues forward.

Each of these components has a value assessed at each word slot in the incoming sentence as well as one for the entire sentence. The word slot values are assumed to have a top rating until the parser reaches that word slot. Thus, the parser is always examining a best case situation based on what it has already done. For example, all word slot transition values are assumed, initially, to have the value 1000*log[1] = 0. The transition value at a word slot is only lowered if it is necessary for the parser to execute a SKIPWORD, EXTRAWS, or LOSTWS command in parsing that word slot. The charge made is according to the value indicated at the particular command in the grammar network. The average of the current values of all word slot transition values creates the sentence transition rating for the parse so far. The word slot and sentence values for the expectation and word values are computed similarly. The compatibility value differs, however, since it does not have degrees of ratings but rather indicates acceptability or lack thereof. Thus, it is not included in the formula for determining a rating for the parse. Rather, if it fails, then parsing automatically backs up. If it succeeds, then parsing continues forward.

The values of the transition, word, and expectation components are used to determine two sentence parse ratings. At each word slot, the values of the three factors are averaged together to produce a general word slot parse rating. Also, the sentence values for the three components are averaged together to obtain a general sentence parse rating. Thus, we have the following equations that define the various rating values, where n is the number of word slots in the sentence:

1) The Transition Value

word slot transition value:
ws__transition[x] = value of SKIPWORD, EXTRAWS, or LOSTWS at word slot x

sentence transition value:
$$\text{transition\_\_confidence} = \sum_{i=0}^{n} \text{ws\_\_transition}[i]/n$$

2) The Word Value

word slot word value:
ws__word[x] = value of the word chosen from the scanner input for word slot x

sentence word value:
$$\text{word\_\_confidence} = \sum_{i=0}^{n} \text{ws\_\_word}[i]/n$$

3) The Expectation Value

word slot expectation value:
ws__expectation[x] = match of word slot x in current parse with slot x in the expected sentence

sentence expectation value:
$$\text{expectation\_\_confidence} = \sum_{i=0}^{n} \text{ws\_\_expectation}[i]/n$$

4) The Parse Values

word slot parse value:
word__slot__factor[x] = (ws__transition[x] + ws__word[x] + ws__expectation[x])/3

sentence parse value:
sentence__factor = (transition__confidence + word__confidence + expectation__confidence)/3

The transition__confidence, word__confidence and expectation__confidence provide an average overall value for the ws__transition, ws__word, and ws__expectation ratings, respectively. These average values provide a best case rating at any point during the parse because they assume perfect ratings for all word slots not yet parsed. The overall parse values, word__slot__factor and sentence__factor, are calculated simply from the average of the other three rating values. This is done so that each factor has equivalent power in controlling the parse. If it is desirable to allow one factor to have more control over the parse than the other two, then this can be accomplished by manipulating the particular minimum rating values discussed below. In order to control the expectation parsing, search is cut-off if rating values fall below certain levels. Currently, these levels are:

1. Minimum word slot transition value (−52)
   Minimum sentence transition value (−12)
2. Minimum word slot word value (−150)
   Minimum sentence word value (−60)
3. Minimum word slot expectation value (−23)
   Minimum sentence expectation value (−7)
4. Minimum word slot parse value (−190)
   Minimum sentence parse value (−65)

If any one of the rating factors drops below its corresponding minimum value, the current search path is cut-off and a different route through the grammar nets is attempted. In this way, there is a control over the extent of the search. By setting all the minimum ratings to

−999, for example, all possibilities in the grammar are checked. On the other hand, setting all the minimum ratings to 0 results in the expectation parser behaving like a normal parser since this essentially turns off the use of the SKIPWORD, EXTRAWS, and LOSTWS commands, the use of synophones, and expectation.

In theory, the parsing algorithm is admissible. That is, it is capable of finding the best possible parse. The various rating factors can initially be set high and gradually lowered until a parse is found. This parse would have the highest rating possible. However, this is impractical in practice due to the amount of time required to repeatedly search a growing space. Thus, minimum rating values are set and the search is conducted once. In this way, the first parse found is the "best" parse in the sense that it is the first one found whose rating was higher than the minimum pre-set value.

5.2 Routines of the Expectation Module

The task of the expectation module is to acquire a general dialogue from a series of dialogues spoken by a user. The dialogues essentially contain examples of how to go about solving a particular kind of problem. In acquiring these dialogues and merging them into one generalized dialogue, the expectation system learns how to solve this particular kind of problem through examples. In a sense, by building this generalized dialogue the expectation system is creating a procedure that can solve a particular subset of problems. This is a future goal of the project. However, the current application is for the generalized dialogue to be used as an aid in the voice recognition process by offering predictions about what might be said next.

The types of problems that can be learned by the existing history-based expectation system include linear algebra applications such as matrix multiplication, simultaneous linear equations, and Gaussian elimination. Non-linear algebra problems that require matrix-type representations can also be learned, such as gradebook maintenance and invoice manipulation. Though the implemented system is limited to matrix-oriented problems, the theoretical system is capable of learning a wide range of problem types. The only requirement on the problem or situation is that it can be entered into the expectation system in the form of examples. Thus, for example, it can acquire a "script" such as the one for going to a restaurant as defined in Schank and Abelson (1977).

The expectation module takes two inputs and produces two outputs. The inputs are

- the user behavior graph discussed earlier, called the expected dialogue D, and
- the meaning of the most recently input sentence, M(S).

Its outputs are a new expected dialogue D modified according to the latest input sentence M(S) and an expected sentence set E. These outputs are produced based upon the inputs and the functions Predicts, Mergeable, and Merge.

The role of the predicate Predicts can be best understood by recalling the function of the parser P. P uses the set of expected sentences E(current) to try to error correct the incoming sentence S. P may do this by discovering that some Mk in E(current) is quite similar to M(S). If P does select such an Mk and uses it to help parse S, then Predicts (Mk, M(S)) is true. Otherwise, Predicts (Mk, M(S)) is false. Thus the function of Predicts is to select the Mk which the parser used in parsing S. If the parser did not use expectation, then Predicts always is false.

If the incoming sentence was not predicted by existing transitions in D, perhaps it can be found to be similar to some node Mk in D and a new transition could be added to that node. The routine Mergeable has the job of finding one or more such Mk's into which the current sentence meaning M(S) can be merged. The question of similarity of two sentences is determined by the meanings of the sentences themselves and the "environment" in which they occur in the dialogue. Sentence "meanings" are based on the sentence deep parses produced by the expectation parser, while a sentence "environment" is based on the meanings of the sentences preceding and following it in the expected dialogue.

Similarity is based on the notion of "distance". Currently two sentences are considered *similar* in meaning if their parses differ in only one slot in the noun group template. This means that their noun group distance cannot be greater than one to be considered similar. For example, the following two sentences are similar:

M("double the first row") = double (r1)

M("double row 2") = double (r2)

The environment of one sentence matches that of another if the sentence meanings preceding the two sentences being compared are identical and/or the sentence meanings following them are identical. Clearly, these definitions are quite arbitrary and many other strategies could be tried. However, for the purposes of this study, they were quite satisfactory.

Based on the question of how well the environment and the sentence itself matches previously seen environments and sentences, five different matches are possible between the current incoming sentence and the elements of the expected dialogue:

1) The sentence matches a sentence meaning in the expected dialogue *exactly,* but there is no match of their environments.
2) The sentence matches a sentence meaning in the expected dialogue *similarly,* but there is no match of their environments.
3) The sentence matches a sentence meaning in the expected sentence set *exactly,* which implies that their environments also match.

4) The sentence matches a sentence meaning in the expected sentence set *similarly,* which implies that their environments also match.
5) There is no match between the sentence and any sentence meaning in the expected dialogue.

In cases 1, 2, and 5, the sentence is determined to be new and unique to the expected dialogue. Therefore, Mk and M(S) are not mergeable. In such cases, M(S) is added as a new entry in the expected dialogue D. In the other two cases, numbers 3 and 4, the incoming sentence is determined to be the same as or similar to one already seen previously in an exact or similar situation. Thus, Mk is mergeable with M(S). In case 3 the sentence is automatically merged with the one that it matches exactly in the expected sentence set. In case 4, the sentence is merged with the one that it matches similarly in the expected sentence set only after it has passed an argument creation algorithm test to be discussed below. Otherwise it is also considered new and unique and added to the expected dialogue as in cases 1, 2, and 5. The actual argument creation occurs in the function Merge.

The notion of creating an argument is associated with the problem of when to merge a set of similar sentences in an expected dialogue into one sentence with a special flag in the slot where the sentences differ. This is determined by the function Mergeable. As an example, at a certain point in a dialogue, one may have an expected sentence set E(i) such as the following:

double (r1)	.33
double (r2)	.33
double (r3)	.33

The numbers indicate the probability levels, derived from j/Ci, as discussed at the end of section 3.

In such a situation, the user's intentions may be reflected more correctly by the following expected sentence set:

double (rARG) 1.0

which signifies that any row may be referred to. However, though this simplified expected sentence set may be a good generalization of the pattern observed, it has ramifications for error correction. Specifically, it will be unable to fill in a row number should that value be missing in the incoming sentence. The first option also has its drawbacks. In this case, should the row number be missing in the sentence, the expectation parser will error correct the sentence to the most probable value, or the first one in the set if the probabilities are equal, here the value one for row 1. Thus, both options are imperfect in terms of the error correction capabilities that they can provide. The comparison that must be made to determine which option is better in a given situation is how often the first will error correct incorrectly as opposed to how much error correcting power we will lose by using the second. How it is done is beyond the scope of this paper but is explained in detail in Fink (1983).

The Merge function takes two inputs, M1 and M2, which have been determined by the Mergeable function to be similar in some way by considering their respective environments and meanings. Based upon how similar the two meanings are, Merge creates a meaning M that is a generalization of M1 and M2, sometimes employing an argument. Thus, there are only two possible kinds of matches at this point between an input sentence and a member of the expected sentence set, an exact match or a similar match. In the case of an exact match M = M1 = M2 and M replaces M1 in the expected dialogue. In the case of a similar match, the meanings only differ by one slot in the noun group of their deep parse representation, so a generalization of that slot to "ARG" is made, meaning an argument is created. The function appears as follows:

```
Merge (M1, M2)
    begin
    for each slot x in M1 and M2 do
        if x(M1) != x(M2) then
            x(M) := ARG;
        else
            x(M) := x(M1);
    end;
```

Thus, if the sentences "Double (r1)" and "Double (r2)" are inputs to Merge, the output would be "Double (rARG)".

6 Experimental Results

An experiment was run using VNLCE to test the error correction capabilities in different situations. These situations were simulated by making the test subjects perform certain tasks on the system that resulted in different dialogue structures, or schemas. The four tests made on VNLCE in this experiment are considered to be representative of the possible schemas that can be produced by different dialogues in different situations. All possible dialogue schemas actually produce a continuum of patterns from totally-ordered to totally-unordered. The tests described below are simply points on this continuum.

I) Totally-Ordered Schema
This type of schema occurs whenever the system has at most two sentences at a time in its expected sentence set and one of these always has a probability rating over 80%.

II) Partially-Ordered Schema
In this case, there is a general order to the sentences being spoken, but there is not usually just one highly probable sentence in the expected sentence set at a time, but several with varying degrees of probability.

III) Totally-Unordered Schema
This occurs when there is no over-all order to the sentences being spoken. Essentially any sentence in

the expectation dialogue has a probability of being spoken next.

IV) Totally-Ordered Schema with Arguments
This test is an example of a totally-ordered schema, but the system does not know exactly what will be said all the time because one or more of the expected sentences contain an argument.

Each of the four tests was run on three different test subjects to acquire data concerning how fast a user speaks, what types of errors are produced by the voice recognizer, and how well the expectation system acquires and uses the expected dialogue to help error correct the input.

To begin the experiment session, the subject trained the voice recognizer, a NEC DP-200, on a specific vocabulary of 49 different words in connected speech mode. The DP-200 can handle only 150 word slots in connected speech mode, so 49 allowed for some repetitive training. The subject was then given a brief tutorial that lead him/her through a few features of the VNLCE system and gave him/her some practice in talking to the NEC device. This training session usually took a total of about 45 minutes. The subject was then given one or more of the test sheets representing the problems to be solved. The number was based on the amount of time that the subject was willing to donate to the effort.

Each test dialogue had a similar over-all structure in that it required a certain amount of repetition, thus creating a loop structure in the expected dialogue. In all tests, except test II, the subject was provided with the specific sequence of sentences to be spoken. This guaranteed that the desired level of repetition was actually achieved. How much repetition there was in each dialogue depended on the expected dialogue schema being imitated. In test I, which was done to demonstrate a totally-ordered schema, the test subject had to repeat an identical sequence of six sentences nine times in a row except for the seventh time when four new sentences were inserted into the loop. A sample schema can be seen in Figure 7. In test II, the user had much more freedom, since its purpose was to demonstrate a partially-ordered schema. Here the subject had to solve six sets of simultaneous linear equations with two equations and two unknowns and he/she spoke whatever sentences that seemed appropriate. A sample schema is shown in Figure 8. Notice that in one case an argument was created. The third test was done to show how well error correction works when the dialogue seems random, creating a totally-unordered schema. To create such an environment, the user was asked to repeat four sentences in random order eight times. An example expected dialogue schema that resulted from this test is shown in Figure 9. In the last test, test IV, the subject was asked to repeat a sequence of four sentences six times, each time through changing the value of the row number spoken. This demonstrates the argument creation facility in a totally-ordered dialogue schema. The expected dialogue generated from this test appears in Figure 10.

Each test has associated with it three charts indicating the results. The first graph represents the average sentence error and correction rates, the second shows the average word error and correction rates, while the third illustrates the average rate-of-speech in words-per-second spoken by the subject while doing the experiment.

The charts indicating the average error and correction rates of the four tests reflect the loop structure of the dialogues. Each chart is a series of bar graphs, each bar graph representing the average error and correction rates over the sentences spoken by the subjects in a particular loop of the dialogue. The highest point on each of these bars represents the raw error rate of the voice recognizer. The different markings within the bars themselves represent the percentage of the errors that were corrected by a particular facility of the expectation system. The horizontal design associated with "loosening" indicates the percentage of the errors that were corrected by the use of the flexible parsing techniques, such features as the synophones and the parser commands SKIPWORD, EXTRAWS, and LOSTWS. The vertical design associated with expectation indicates the percentage of the errors that were corrected by use of the expected sentence set alone. The blank area indicates the percentage of the errors that were corrected by using both of the above facilities. Finally, the dot design shows the percentage of the errors that were not corrected. Thus, for example, in the top chart in Figure 11, the eighth loop of the dialogue had an 85% sentence error rate from the voice recognizer. Of those errors, 6% were corrected using the facilities associated with loosening the search, while 25% were corrected by using only expectation. Another 63% were corrected using features from both categories. Only 6% could not be corrected.

Test I, using a totally-ordered dialogue schema, was done to show how well the expectation system can error correct errorful input when it can predict exactly what will be said next. As can be seen from the graphs in Figure 11, as the ability to predict what will be said next increases, so does the ability to error correct. In loop seven of the dialogue, we deliberately had each user add four extra sentences between the fourth and fifth sentences of the loop. This was done to show that the expectation system had not become a complete automaton, but that it was still capable of dealing with unexpected input. However, as can be seen from these graphs (Figure 11), the expectation system's error correcting power decreases in that particular loop of the dialogue since there is no expectation at certain points to help it.

Test II, creating a partially-ordered dialogue schema, was done to show how the expectation acquisition algorithm dealt with dialogues containing some pattern and to see how well error correction could work when expectation was not perfect. The results are shown in Figure 12.

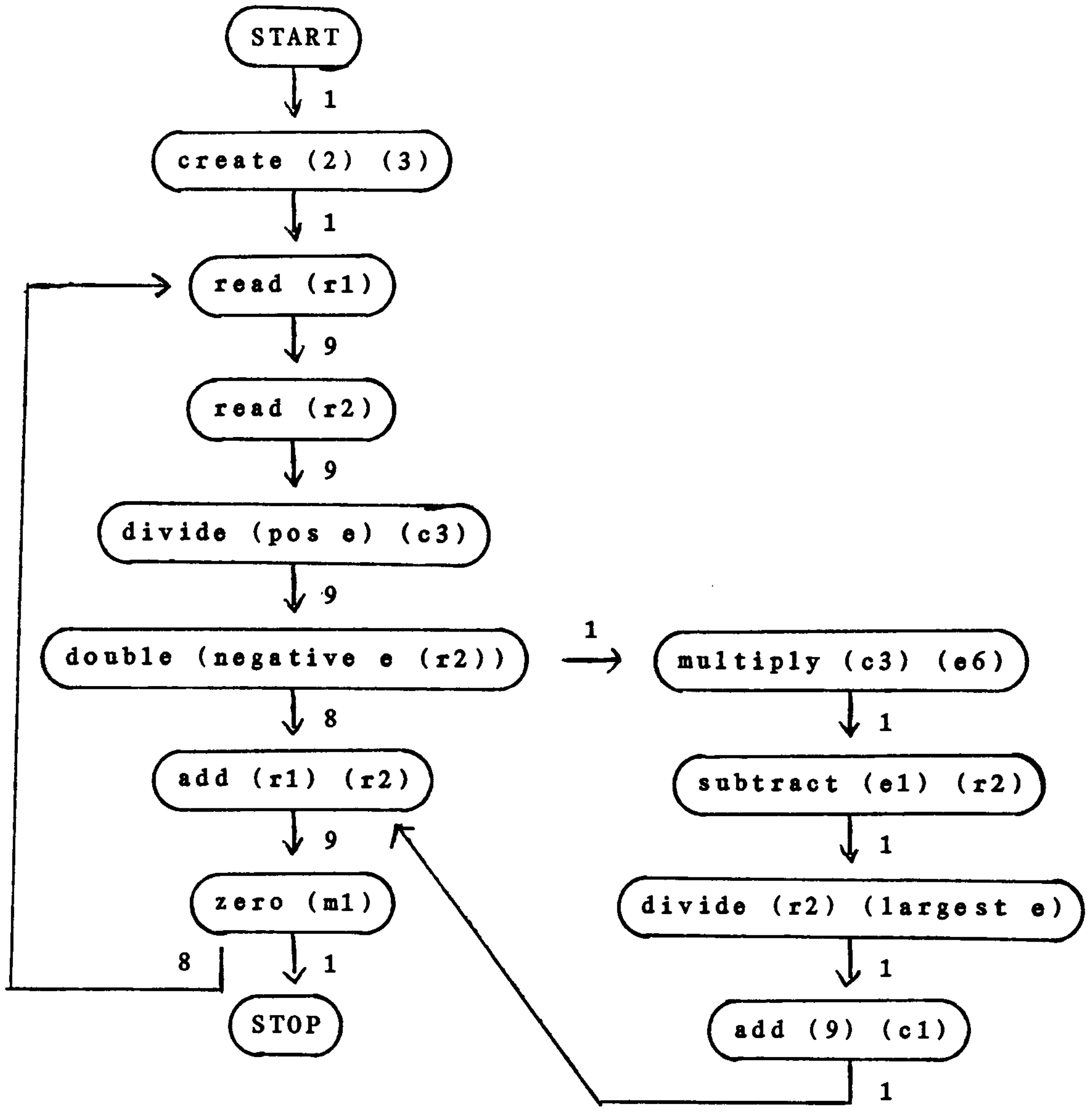

Figure 7. Expected dialogue schema for Test I.

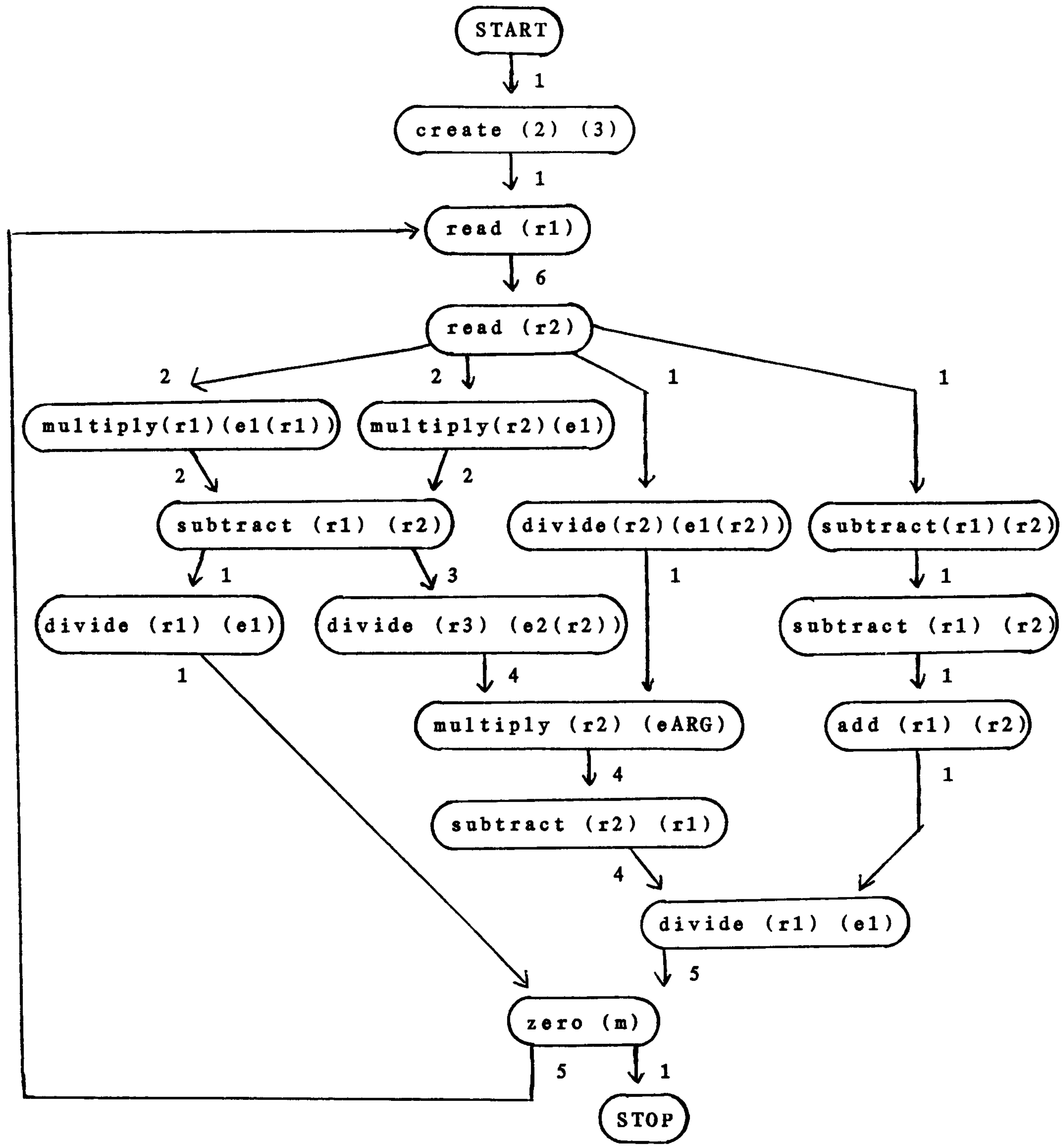

Figure 8. Expected dialogue schema for Test II.

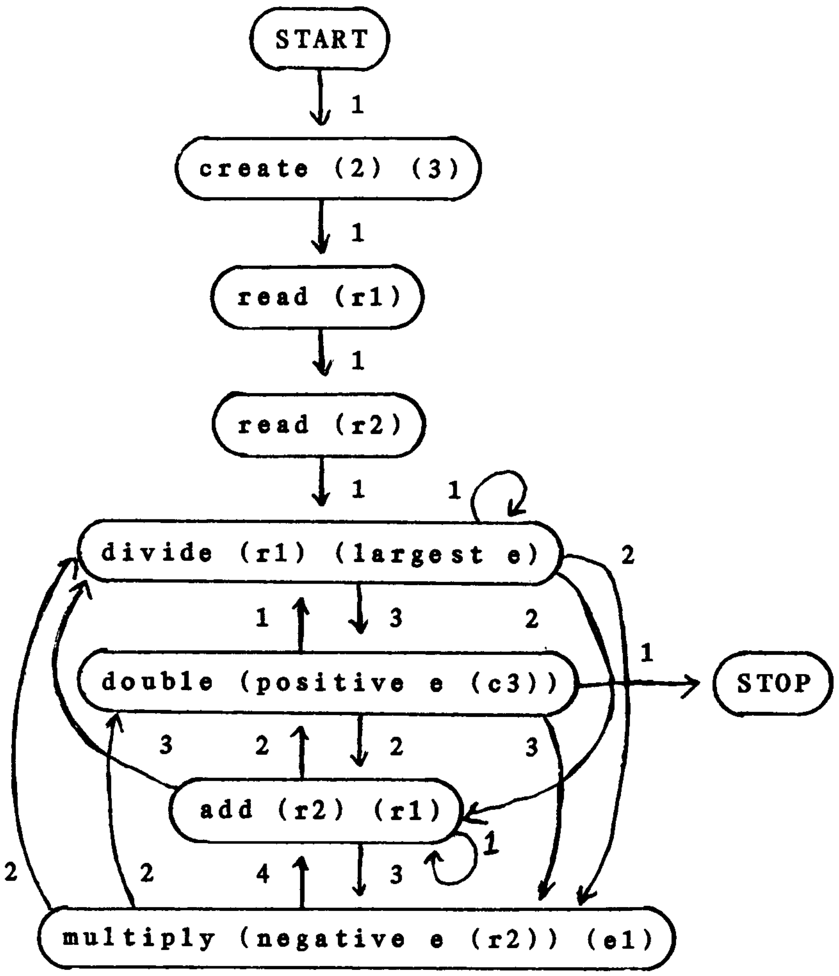

Figure 9. Expected dialogue schema for Test III.

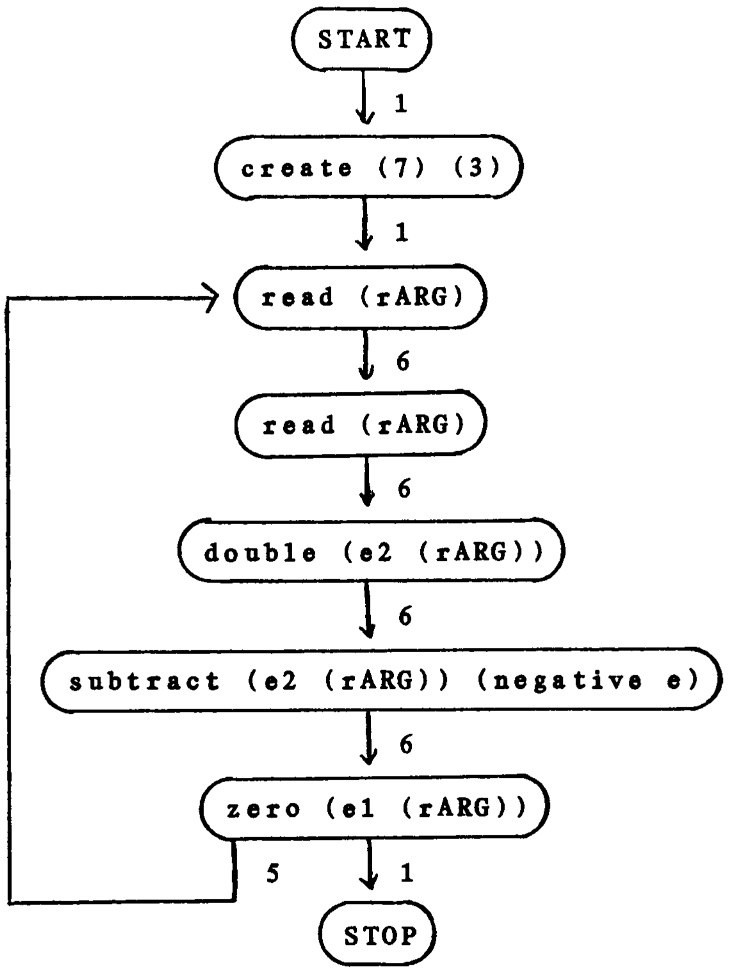

Figure 10. Expected dialogue schema for Test IV.

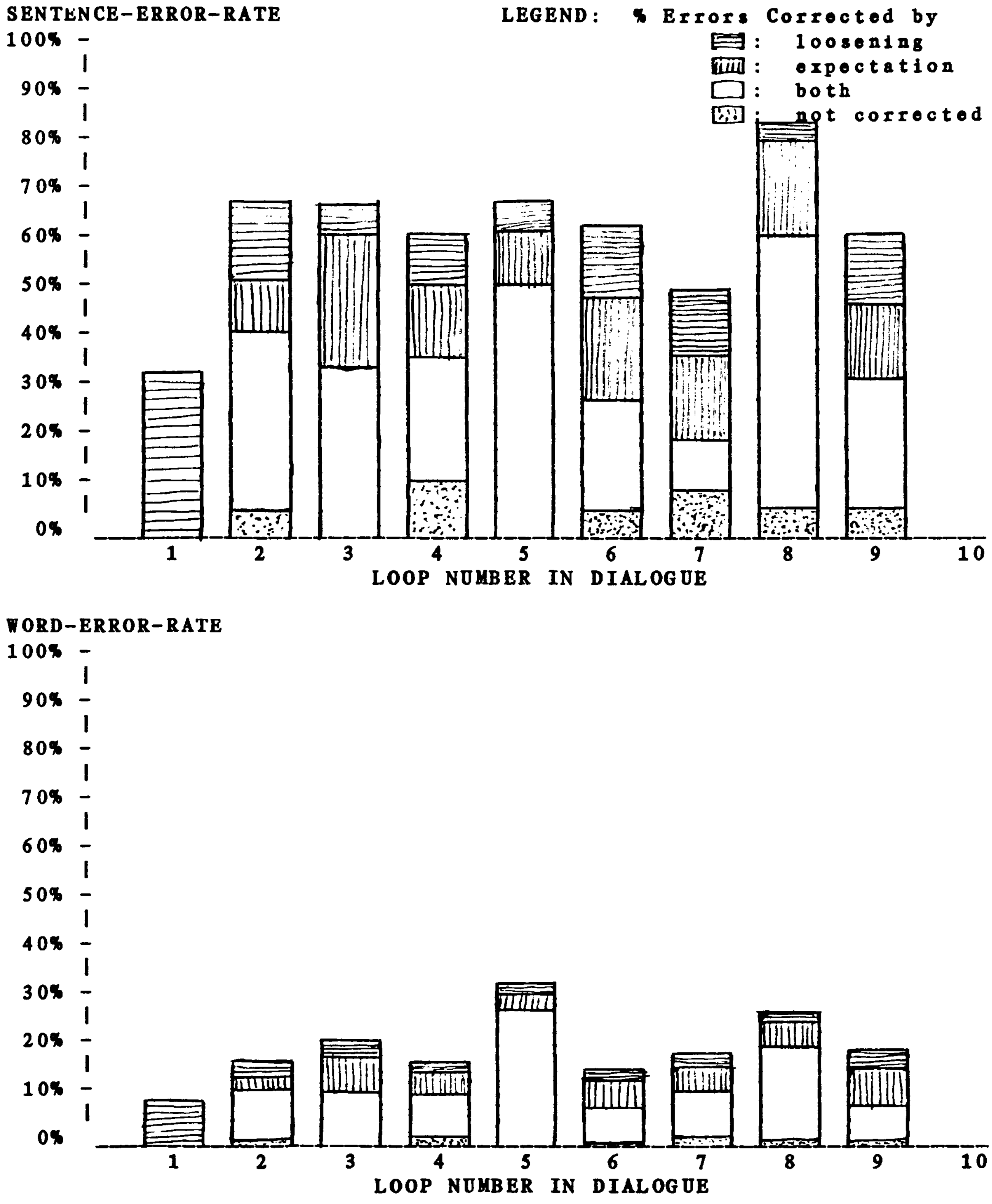

Figure 11. Error and correction rates for Test I.

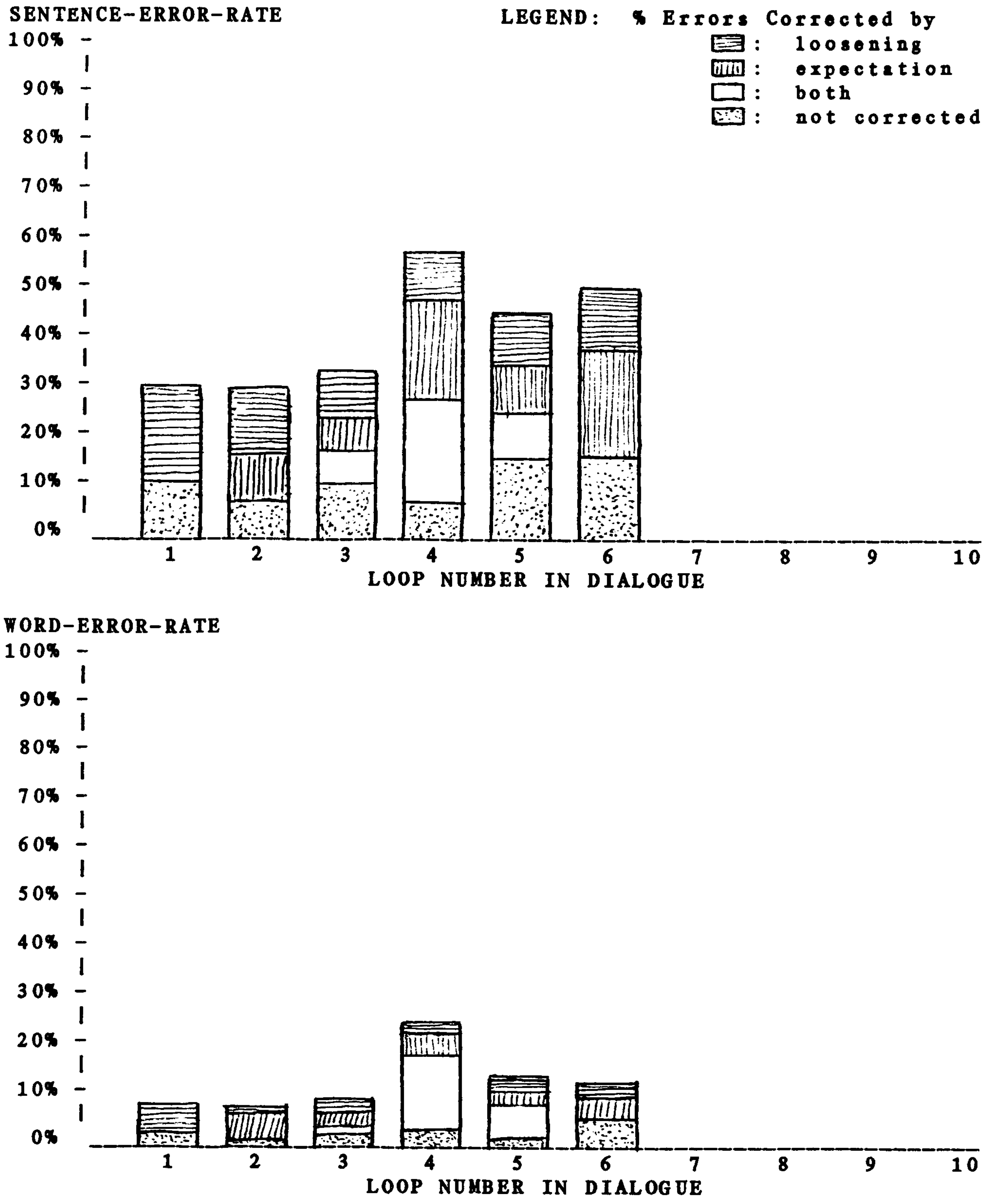

Figure 12. Error and correction rates for Test II.

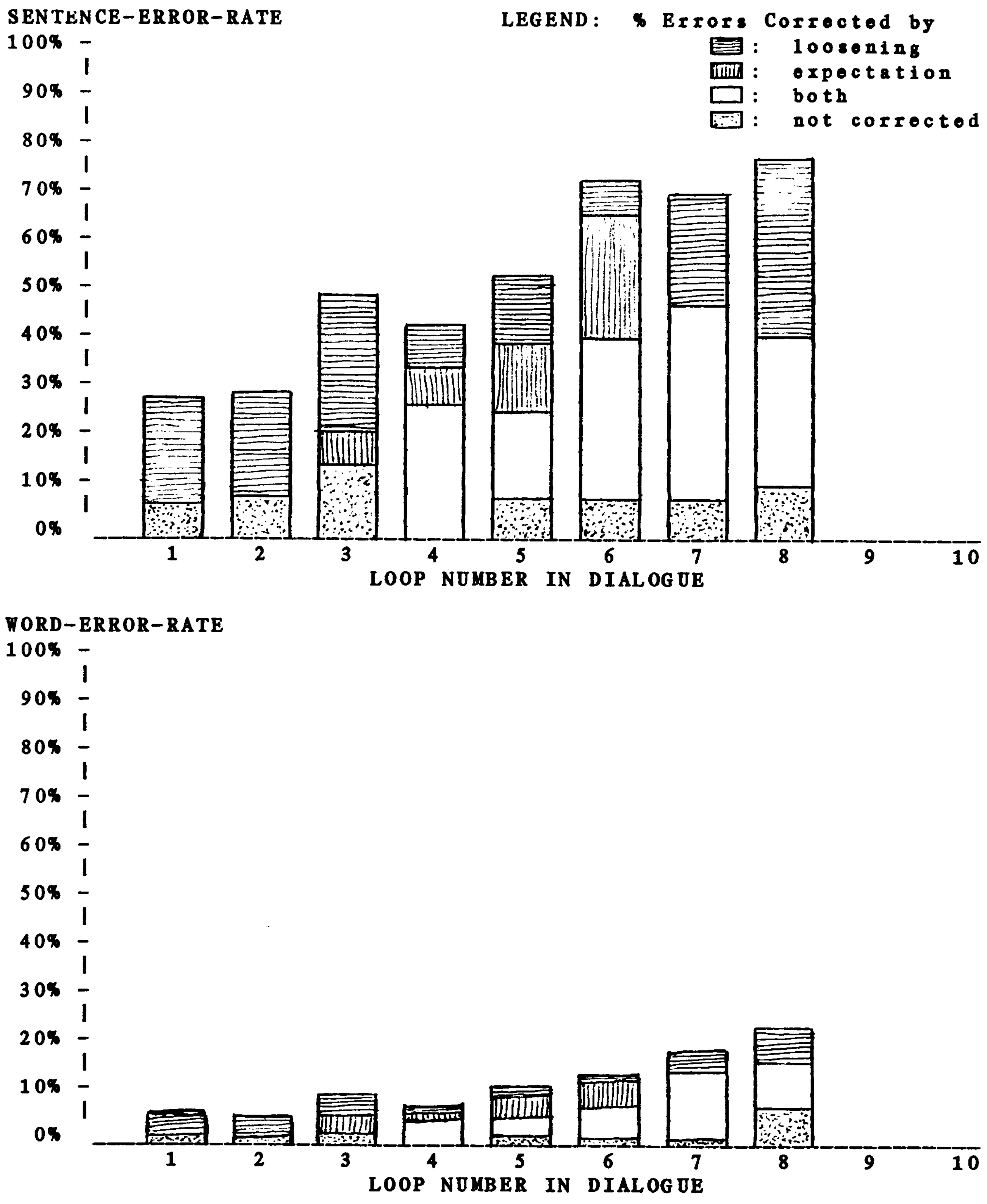

Figure 13. Error and correction rates for Test III.

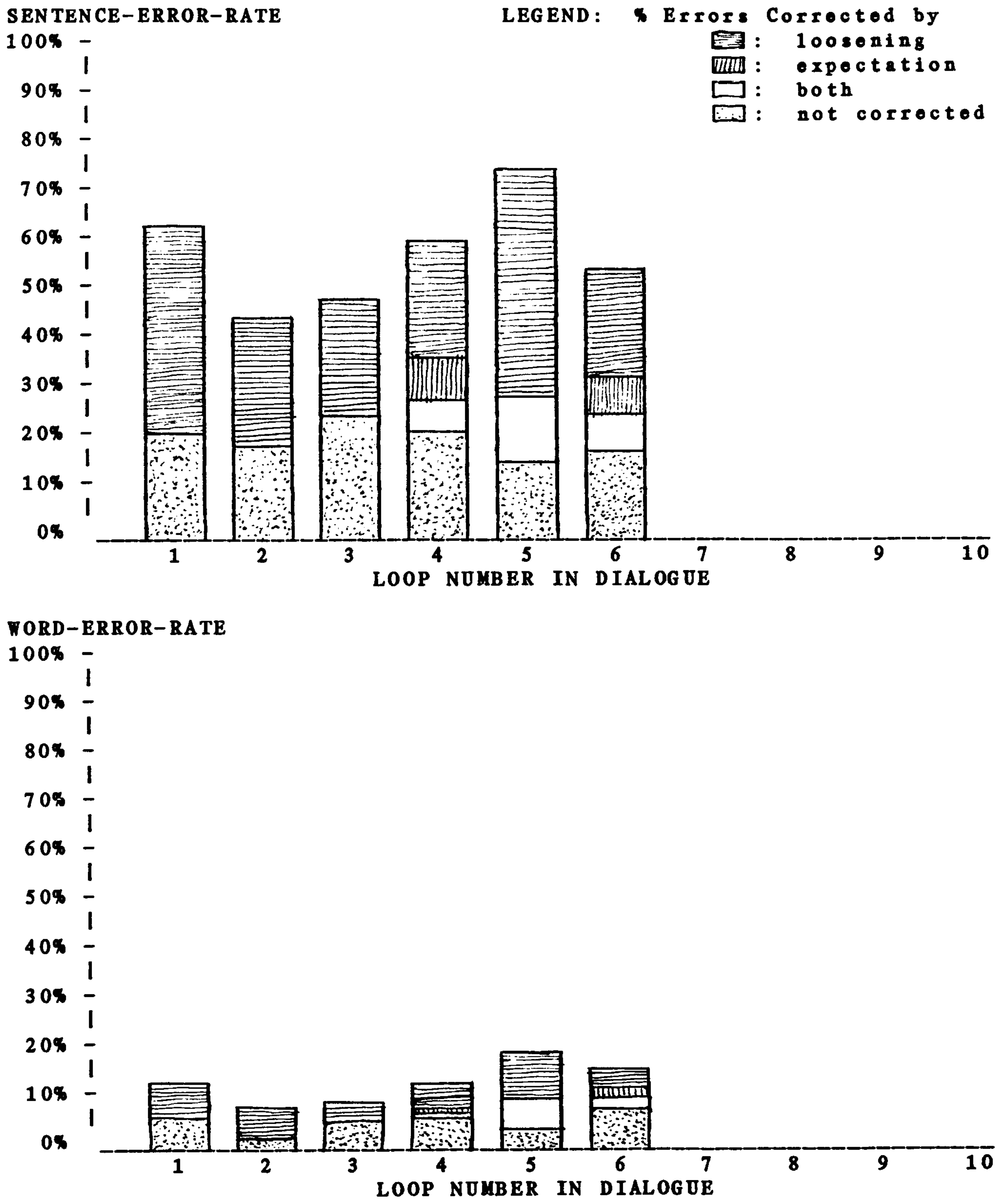

Figure 14. Error and correction rates for Test IV.

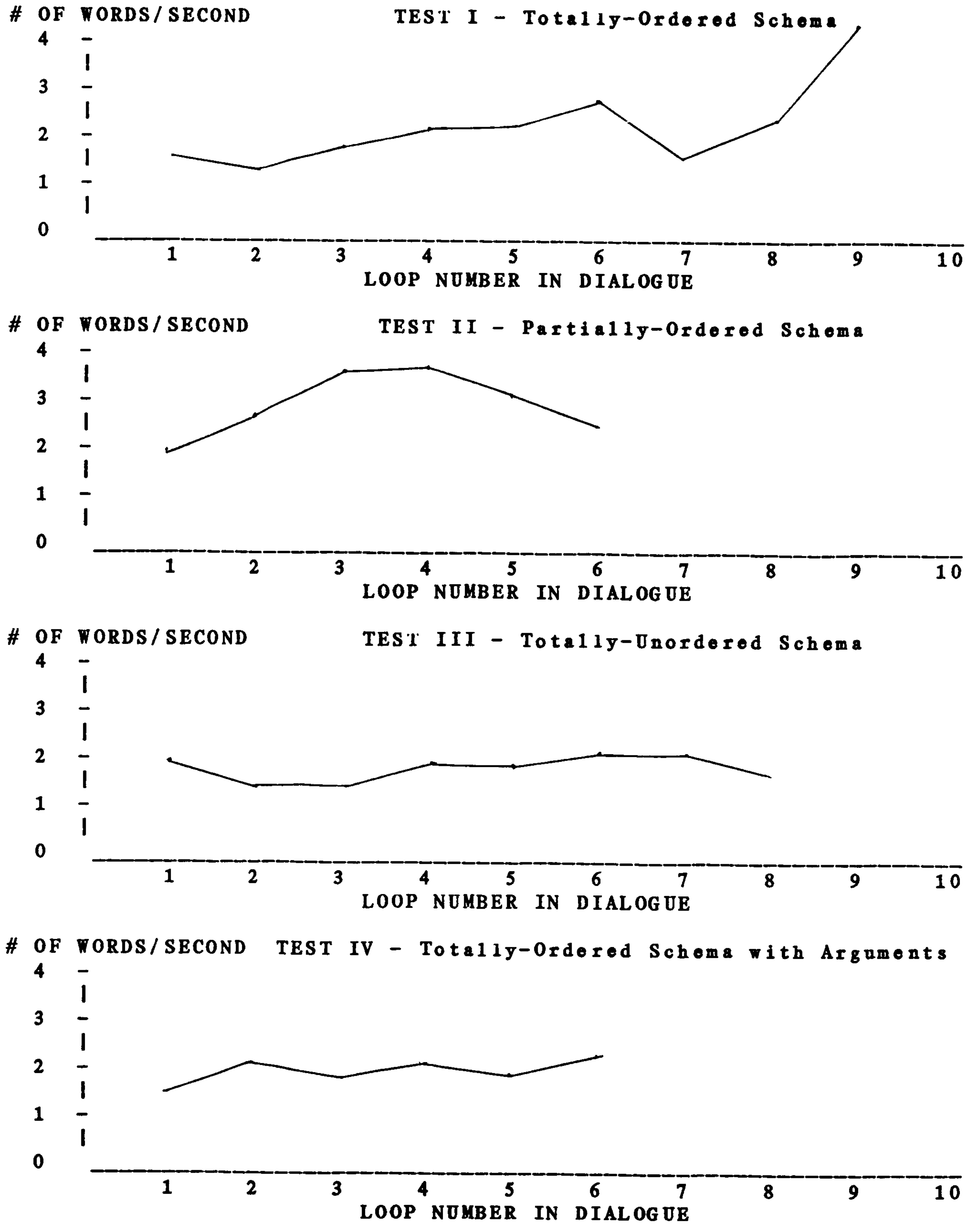

Figure 15. Speech rate in the four dialogue schema tests.

Test III demonstrates the error correction capabilities of the system when expectation only knows that one of a group of sentences will be said next. It produces a totally-unordered dialogue schema. The results of the systems error correction capabilities in such a situation appear in Figure 13.

Test IV uses a totally-ordered dialogue schema, but with a variation from test I. Each sentence sooner or later contains an argument so that the system does not know everything about the sentence that will be said next. The data given in Figure 14 shows the error correction rates for this dialogue. It clearly shows how error correction failures increase until after the third loop when argument creation begins so that the system no longer error corrects incorrectly.

Figure 15 shows the graphs of the average speech rate of the speakers for each of the four tests. Like the other eight graphs, these graphs reflect the loop structure of the dialogues. As can be seen, the speakers tended to increase their speech rate as they talked to the system. This behavior was hoped for because as the speech rate increased, so did the error rate of the speech recognizer, thus placing more of a burden on the error correcting abilities of the expectation system. Note that, in all eight graphs in Figures 11 through 14, the word and sentence error rates from the voice recognizer generally increased with the progress through the dialogue. This is due to the increased rate of speech. However, the actual failure rate of VNLCE did not increase by the same amount. These extra errors were corrected by the expectation system.

Figure 16 gives a summary of the average error and correction rates for each test and over all.

7 Related Literature

A number of speech understanding systems have been developed during the past fifteen years (Barnett et al. 1980, Dixon and Martin 1979, Erman et al. 1980, Haton and Pierrel 1976, Lea 1980, Lowerre and Reddy 1980, Medress 1980, Reddy 1976, Walker 1978, and Wolf and Woods 1980). Most of these efforts concentrated on the interaction between low level information sources from a speech recognizer and a natural language processor to discover the meaning of an input sentence. While some of these systems did exhibit expectation capabilities at the sentence level, none acquired dialogues of the kind described here for the sake of dialogue level expectation and error correction. A detailed description of the kinds of expectation mechanisms appearing in these systems appears in Fink (1983).

The problem of handling ill-formed input has been studied by Carbonell and Hayes (1983), Granger (1983), Jensen et al. (1983), Kwasny and Sondheimer (1981), Riesbeck and Schank (1976), Thompson (1980), Weischedel and Black (1980), and Weischedel and Sondheimer (1983). A wide variety of techniques have been developed for addressing problems at the word, phrase, sentence, and in some cases, dialogue level. However, these methodologies have not used historical information at the dialogue level as described here. In most cases, the goal of these systems is to characterize the ill-formed input into classes of errors and to correct on that basis. The work described here makes no attempt to classify the errors, but treats them as random events that occur at any point in a sentence. Thus, an error in this work has no pattern but occurs probabilistically. A verb is just as likely to be mis-recognized or not recognized as is a noun, adjective, determiner, etc.

	Test I	Test II	Test III	Test IV	Over-all
word-error-rate	18.78	11.75	11.25	12.17	13.49
corrected word-error-rate	.59	1.50	1.50	4.17	1.94
sentence-error-rate	61.22	40.83	52.25	56.33	52.66
corrected sentence-error-rate	3.22	6.00	5.38	16.00	7.65
average speaking rate	2.27	2.95	1.85	1.97	2.26

Figure 16. Average word and sentence error rate in percent, average speaking rate in words-spoken-per-minute.

The acquisition of dialogue as implemented in VNLCE is reminiscent of the program synthesis methodology developed by Biermann and Krishnaswamy (1976) where program flowcharts were constructed from traces of their behaviors. However, the "flowcharts" in the current project are probabilistic in nature and the problems associated with matching incoming sentences to existing nodes has not been previously addressed. Another dialogue acquisition system has been developed by Ho (1984). However, that system has different goals: to enable the user to consciously design a dialogue to embody a particular human-machine interaction. The acquisition system described here is aimed at dealing with ill-formed input and is completely automatic and invisible to the user. It self activates to bias recognition toward historically observed patterns but is not otherwise observable.

The VNLCE processor may be considered to be a learning system of the tradition described, for example, in Michalski et al. (1984). The current system learns finite state flowcharts whereas typical learning systems usually acquire coefficient values as in Minsky and Papert (1969), assertional statements as in Michalski (1980), or semantic nets as in Winston (1975). That is, the current system learns procedures rather than data structures. There is some literature on procedure acquisition such as the LISP synthesis work described in Biermann et al. (1984) and the PROLOG synthesis method of Shapiro (1982). However, the latter methodologies have not been applied to dialogue acquisition.

8 Conclusions and Areas for Future Research

We have shown that the ability to use expectation in the form of knowledge about the dialogue being spoken, as with humans, is a tremendous aid to speech recognition by computer. Since expectation, in this research, has been based on repetition of patterns, the expectation system's ability to correct varies, of course, with the repetitiveness of the dialogue itself. We have attempted, in sections 5 and 6, to justify this decision by demonstrating how the expectation system can acquire common programming constructs such as loops and arguments. It is our belief that repetitious patterns occur in everyday life, and that the expectation system is capable of dealing with such patterns, resulting in a generalized situation similar to a Schankian script. Finally, we have tested the expectation system's correction power in some representative situations, as discussed in section 6. It has been demonstrated that the expectation system has the capabilities of reducing a large sentence error rate to nearly zero in many situations. At the word level, error rates to the expectation system climbed as high as 47% in certain user dialogues when the user was speaking fast. At the same time, the error rate leaving the expectation system remained fairly low at between zero and fifteen percent. On the average, the system was able to lower a sentence error rate of 53% to 8%, and a word error rate of 13.5% to 2%. The use of expectation, along with an ability to ignore or add words to the input stream of the parser, is all that is needed to achieve this error correction rate on randomly erroneous input.

The parser design, with the five choices at each word slot, has the potential to run into problems with the exponential growth of the search and to result in unacceptably long parse times. However, when the rating scheme is used intelligently, it not only aids in finding the best parse of a word sequence, but it also helps to lower the search time necessary by pruning unreasonable search choices. The average parse time for a sentence, from the tests discussed above, was 5.1 seconds while the average total processing time for a sentence was 10.5 seconds. This was on a highly loaded PDP 11/70 under the UNIX[1] operating system. In the event that a particular word sequence leads the parser down a garden path, a time-out facility has been implemented that causes the parser to fail after one minute of real-time. However, out of a total of 629 sentences spoken in the above four tests, this feature was needed only 19 times.

The research reported on here was divided into two parts, the theory and the implementation. Most of the theory developed was implemented in the VNLCE system. This theory has been aimed at error correction of random errors using expectation based on historical information. However, there are many possible extensions that could be examined in the future and added to the implementation if the investigation indicates that it would create a yet more usable system. These include the following:

- use of low level knowledge from the speech recognition phase,
- use of high level knowledge about the domain in particular and the dialogue task in general,
- a "continue" facility and an "auto-loop" facility as described by Biermann and Krishnaswamy (1976),
- a "conditioning" facility as described by Fink et al. (1985),
- implementation of new types of paraphrasing,
- checking a larger environment in the expectation acquisition algorithm when deciding if an incoming sentence is the same or similar to one already seen, and
- examining inter-speaker dialogue patterns.

All but two of these areas for expansion are aimed at moving the expectation system from one that finds patterns in a user's dialogues and acquires historical knowledge about them to one that can acquire true procedures. The first two areas for expansion have nothing to do with creating a true procedure acquisition module but would be highly desirable from the point of view of the speech recognition application. Features three and four would simply make the system easier to use and would require little theoretical investigation. The final three would require research efforts.

In conclusion, we have designed a system that is capable of correcting ill-formed input and implemented the design in the area of speech recognition. The system performs error-correction through a mechanism also used by humans in the same situation, that of expectation. We have shown that the expectation algorithm is general enough to handle almost any dialogue structure. It is possible to predict approximately what kind of error correction to expect from the system based on the dialogue structure and the word error rate. We have also shown that the theory on which the implemented expectation system is based is capable of acquiring and generalizing real-world, script-like situations. This research can serve as a starting point for further research into the field of computer expectation, procedure acquisition, and learning.

References

Ballard, B. 1979 Semantic Processing for a Natural Language Programming System. Ph.D. Dissertation Report CS-1979-8, Duke University, Durham, North Carolina.

Barnett, J.; Berstein, M.; Gillman, R.; and Kameny, I. 1980 The SDC Speech Understanding System. In Lea 1980: 272-293.

Biermann, A. and Ballard, B. 1980 Toward Natural Language Computation. *AJCL* 6(2): 71-86.

Biermann, A.; Guiho, G.; and Kodratoff, Y., Eds. 1984 *Automatic Program Construction Techniques.* Macmillan Publishing Co., New York, New York.

Biermann, A. and Krishnaswamy, R. 1976 Construction of Programs from Example Computations. *IEEE Transactions on Software Engineering* SE-2(3): 141-153.

Biermann, A.; Rodman, R.; Rubin, D.; and Heidlage, J. 1985. Natural Language with Discrete Speech as a Mode for Human-to-Machine Communication. *Comm. of ACM* 28(6).

Carbonell, J. and Hayes, P. 1983 Recovery Strategies for Parsing Extragrammatical Language. *AJCL* 9(3-4): 123-146.

Dixon, N. and Martin, T., Eds. 1979 *Automatic Speech and Speaker Recognition.* IEEE Press, New York, New York.

Erman, L.; Hayes-Roth, F; Lesser, V.; and Reddy, D. 1980 The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty. *Computing Surveys,* 12(2).

Fink, P. 1983 The Acquisition and Use of Dialogue Expectation in Speech Recognition, Dissertation, Department of Computer Science, Duke University.

Fink, P.; Sigmon, A.; and Biermann, A. 1985 Computer Control Via Limited Natural Language. *IEEE Trans SMC* SMC-14(1): 54-68.

Granger, R. 1983 The NOMAD System: Expectation-Based Detection and Correction of Errors during Understanding of Syntactically Ill-Formed Text. *AJCL* 9(3-4): 188-196.

Haton, J. and Pierrel, J. 1976 Organization and Operation of a Connected Speech Understanding System at Lexical, Syntactic and Semantic Levels. 1976 IEEE International Conference on Acoustics, Speech and Signal Processing, Philadelphia, Pennsylvania: 430-433.

Ho, T.-P. 1984 The Dialogue Designing Dialogue System, Dissertation, Computer Science Department, California Institute of Technology.

Jensen, K.; Heidorn, G.; Miller, L.; and Ravin, Y. 1983 Parse Fitting and Prose Fixing: Getting a Hold on Ill-Formedness. *AJCL* 9(3-4): 147-160.

Kwasny, S. and Sondheimer, N. 1981 Relaxation Techniques for Parsing Grammatically Ill-Formed Input in Natural Language Understanding Systems. *AJCL* 7(2): 99-108.

Lea, W., Ed. 1980 *Trends in Speech Recognition.* Prentice-Hall, New Jersey.

Lowerre, B. and Reddy, R. 1980 The Harpy Speech Understanding System. In Lea 1980: 340-360.

Medress, M. 1980 The Sperry Univac System for Continuous Speech Recognition. In Lea 1980.

Michalski, R. 1980 Pattern Recognition as Rule-Guided Inductive Inference. *IEEE Trans. Pattern Analysis and Machine Intelligence.*

Michalski, R.; Carbonell, J.; and Mitchell, T. 1984 *Machine Learning.* Springer Verlag, New York.

Minsky, M. and Papert, S. 1969 *Perceptrons.* MIT Press, Cambridge, Massachusetts.

Reddy, D. 1976 Speech Recognition by Machine: A Review. *Proceedings of the IEEE* 64(4): 501-531.

Riesbeck, C. and Schank, R. 1976 Comprehension by Computer: Expectation-Based Analysis of Sentences in Context. Tech. Rep. 78, Computer Science Department, Yale University, New Haven, Connecticut.

Schank, R. and Abelson, R. 1977 *Scripts, Plans, Goals, and Understanding.* Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Shapiro, E. 1982 *Algorithmic Program Debugging.* MIT Press, Cambridge, Massachusetts.

Thompson, B. 1980 Linguistic Analysis of Natural Language Communication with Computers. Proceedings of the Eighth International Conference on Computational Linguistics, Tokyo, Japan: 190-201.

Walker, D., Ed. 1978 *Understanding Spoken Language.* Elsevier North-Holland, New York, New York.

Weischedel, R. and Black, J. 1980 Responding Intelligently to Unparsable Inputs. *AJCL* 6(2): 97-109.

Weischedel, R. and Sondheimer, N. 1983 Meta-Rules as a Basis for Processing Ill-Formed Input. *AJCL* 9(3-4): 161-177.

Winston, P. 1975 Learning Structural Descriptions from Examples. In Winston, P., Ed., *Psychology of Computer Vision.* McGraw-Hill, New York, New York.

Wolf, J. and Woods, W. 1980 The HWIM Speech Understanding System. In Lea 1980: 316-339.

Woods, W. 1970 Transition Network Grammars for Natural Language Analysis. *Comm. of the ACM* 13(10): 591-606.

Note

1. UNIX is a trademark of AT&T Bell Laboratories.