Integrating a Connectionist Parser and a Graph Matching Algorithm for Handling Disfluencies in Spoken Language Processing

By

Venkatesh Manian

A Thesis
submitted to the Faculty of Graduate studies
in partial fulfillment of the requirements
for the degree
of
Master of Science

Department of Computer Science
University of Manitoba
Winnipeg, Manitoba

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION PAGE

Integrating a Connectionist Parser and a Graph Matching Algorithm for

Handling Disfluencies in Spoken Language Processing

BY

Venkatesh Manian

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University

of Manitoba in partial fulfillment of the requirements of the degree

of

MASTER OF SCIENCE

Abstract

There are two major components in applications that have natural language interface: speech recognition and spoken language processing. A speech recognizer converts the input sound into written text (which represents spoken language) and the spoken language processing part performs the syntactic and semantic analysis of the input text. Spoken language is characterized by disfluencies such as "ums", "uhs", pauses, correction, repetition, and word fragments. Natural language parsers, which provide syntactic interpretation for a sentence, may fail if these spoken language phenomena are present. Hence, a robust parsing approach is required to handle such disfluencies.

In the first part of this thesis, I enhanced the Hybrid Connectionist Parser (HCP) to process interim syntactic derivations that occur during parsing. Temporary bindings with tags were used to look at all possible derivations. Finally, the derivation, which represents the complete structure for the input sentence was selected.

In phase two, the Phrase Structure Matching Algorithm (PSMA) was developed based on the graph matching algorithm [25] to remove disfluencies occurring in spoken language. The enhanced version of the HCP was used for robust parsing of spoken language. Initially, partially parsed structures are generated by the parser. The parsed structures are then used by the Phrase Structure Matching Algorithm (PSMA) to detect and remove the disfluent structure. A complete sentence structure for the input utterance was derived after the disfluent structure was detected.

Finally, the spoken language processing system was tested using the HCRC Map Task Corpus. The Post-processing and the Incremental Processing Methods were the two

different experiments performed to process the disfluent parts of the utterance. In the post-processing method, disfluent parts are corrected after parsing the utterance. The incremental processing method processes the disfluent part as soon as an error is detected and enough information is gathered to correct the error. 42% of the disfluent utterances were detected and corrected using the post-processing method. Conversely, the incremental method corrected disfluent parts in 60% of the utterances. Therefore, the post-processing method is more suitable for utterances which have one repair instance, whereas the incremental method is better suited to handle many repair instances within an utterance.

Acknowledgements

I take immense pleasure in acknowledging all people who have supported me in different ways. I want to start by thanking my supervisor Dr. Christel Kemke for her motivation and support throughout the research.

I would like to thank my committee member Dr. Ruppa k. Thulasiraman and Dr. Kevin Russell for their valuable suggestions and comments on this thesis.

Special thanks to my friends of Winnipeg for their encouragement and support. I would like to thank my parents, brother and sister for their encouragement and support. I dedicate this to my parents.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1 Introduction

Natural language is used to interact with the user in a real time speech processing system such as the ATIS (Air Travel Information Services), the CU Communicator system [34] and TRAINS [22]. These real time systems are designed to understand spontaneous speech to interact with the user. The spoken language output from a speech recognition system is characterized by disfluencies. Disfluencies occur when a speaker restates or modifies the sentence they started to say. False starts, filled pauses, repairs, repetitions, mid-sentence corrections and word fragments are classified as disfluencies. The processing of spoken language with disfluencies starts with recognizing the disfluent part of the sentence and detecting the user's intended utterance[1]. In this thesis, the Hybrid Connectionist Parser (HCP) is used to generate partially parsed structures for the input sentence. These structures are then given as input to the Phrase Structure Matching Algorithm (PSMA) to detect the disfluent structure.

A natural language parser will generate the syntactic structure for a sentence based on given grammar. A spoken language utterance is characterized by the presence of disfluencies. Therefore, the parser cannot generate a syntactic structure for a sentence which is disfluent. This thesis will demonstrate that the HCP and the PSMA can be integrated to handle disfluencies. The PSMA, which is based on the existing Graph Matching Algorithm, was altered to deal with different possible situations that occur during parsing. The PSMA was tested using the HCRC Map Task Corpus, which is a collection of spoken language transcripts.

[1] In this thesis, utterance always denotes a spoken language sentence.

In a spoken language utterance, the speaker restates or modifies what s/he started to say. Figure 1-1 shows an utterance taken from the Air Travel Information System (ATIS) corpus.

Show flights **from Boston on** uh **from Denver on** Monday
Reparandum IP IM Alteration

Figure 1-1: Basic structure of a spoken language utterance [39].

The utterance in Figure 1-1 incorporates a correction made by the speaker. The speaker enquires about flights from Boston but reframes his query and then enquires about flights from Denver. The *reparandum* shown in Figure 1-1 represents the part of the phrase that has to be replaced or corrected. The *interruption point* is the point at which the user detects the need to repeat or correct what they have uttered. The *interregnum* (IM) represents editing terms which follow the interruption point. The *alteration* represents the phrase or word that is supposed to replace the *reparandum*. Due to the presence of such corrections in an utterance, a breakage in the phrase structure occurs. In Figure 1-1, there is an incomplete preposition phrase generated by the fifth word "on" followed by a complete preposition phrase "from Denver". The repairs present in an utterance are classified depending on the way in which the user modifies his utterance.

1.1 Classification of Speech Repairs

Depending on the arrangement of the reparandum and its alteration in an utterance, speech repairs have been classified into three types: fresh starts, modification repairs, and abridged repairs.

Fresh Starts [17]**:** An utterance falls in this category if the user restates his previously said utterance. Figure 1-2 shows that the utterance "I need to send" is replaced by the following utterance "how many boxcars can one engine take".

I need to send let's see how many boxcars can one engine take
Reparandum IP IM Alteration

Figure 1-2: Fresh starts from [17].

Modification Repairs [17]**:** The speaker does not restate what they have uttered but replaces or repeats a word or a phrase. A characteristic of this speech repair is that there is a strong word correspondence between the reparandum and the alteration.

You can carry them both on tow both on the same engine.
Reparandum IP Alteration

Figure 1-3: Modification repair from [17].

Abridged Repairs [17]**:** The speaker stops or pauses while uttering a phrase and then continues what they started to say. Figure 1-4 shows an example of an abridged repair.

We need to um manage to get the bananas
IP IM

Figure 1-4: Abridged repair from [17].

1.2 Research overview

In this thesis, a syntactic method is used to deal with disfluent structures. The spoken language input is first given to a syntactic parser. The Hybrid Connectionist Parser (HCP) [25] parses the input text. I enhanced the existing Hybrid Connectionist Parsing method to deal with ambiguous syntactic structures. On parsing a spoken language utterance, the HCP generates partially parsed structures due to the presence of

disfluencies. In Example 1-1, partially parsed structures generated for the utterance "I I am at the walled city" are shown.

Example 1-1

1. `S [NP[PRP [I]] VP[]] and`
2. `S [NP[PRP [I]] VP[BEM[am] PP[IN[at] NP[DT[the] NOM[walled city]]]]`

On parsing the utterance "I I am at the walled city", a partial parse tree for the first word "I" is generated. The partial parse tree generated for the first word "I" is shown as the first structure in Example 1-1. The second structure in Example 1-1 shows a complete sentence structure formed for the remaining parts of the utterance, "I am at the walled city". In the second part of the thesis, the PSMA was developed to process partially parsed structures.

The PSMA derives a proper syntactic structure for the input text using partial parse trees. Two different methods have been followed to process partially parsed structures. The first method is referred to as the Post-Processing Method. In this method, the PSMA is applied after deriving partially parsed structures for the input utterance. Moreover, the final completed phrase is taken as the corrected part of the utterance. In Example 1-1, the sentence structure formed for the words "I am at the walled city" is taken as a corrected part of the utterance. The PSMA then searches for a similar sentence structure. The structure formed for the first word is also an incomplete sentence structure. Therefore, the PSMA detects the sentence structure formed for the first word "I" as the reparandum.

The second method is referred to as the Incremental Processing Method. In the incremental processing method, the PSMA is triggered as soon as a disfluency is

detected. In Example 1-1, the sentence structure formed for the first word "I" expects a verb phrase (VP) to follow. Due to the presence of a repetition of the word "I", the parser generates a noun phrase structure. The parser detects the presence of a disfluency and the PSMA is triggered. The PSMA then checks for a similar noun phrase structure. Due to the presence of a similar noun phrase formed for the first word "I", an overlapping of the noun phrase takes place. Now, the parser takes control and continues to parse the remaining words of the utterance. Figure 1-5 shows the necessary steps used in the incremental processing method. The incomplete sentence structure created based on the first word "I" is shown as the reparandum. The structure represented as the reparandum expects a verb phrase as the next incoming phrase. Since the parser generates a noun phrase based on the second input word, the PSMA is triggered. The PSMA detects the presence of similar structures. Finally, overlapping of the noun phrase structures takes place and the parser continues to generate structures for the next word.

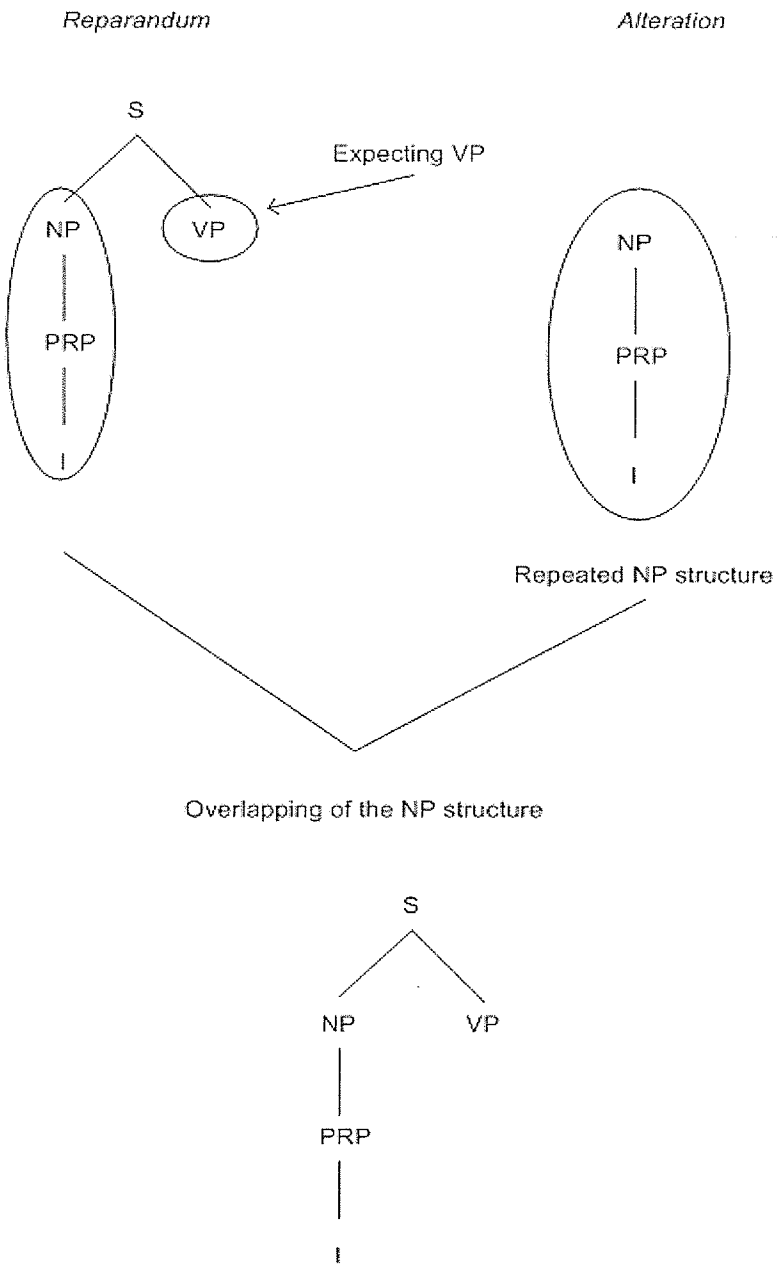


Figure 1-5: Incremental processing method

1.3 Contributions of the Thesis

The major contributions of this thesis are as follows:

- Incremental parsers are used to generate syntactic structures for sentences which are grammatical. In this thesis, the Hybrid Connectionist Parser (HCP) was used to process disfluencies of spoken language. Detecting the interruption point in speech is difficult because of the presence of interim derivation. In this thesis, we have derived a method to detect the interruption in an incremental fashion.
- The HCP method was enhanced to deal with interim derivations and different possible structural interpretations that occur during incremental parsing. The presence of disfluent parts in a spoken language utterance causes the HCP to generate broken phrase structures which gives enough information to detect and correct repairs by the PSMA.
- Unlike other methods that focus on different allowable patterns of repair to process disfluencies in speech, the PSMA looks at the similarity in the phrase structures between the reparandum and its alterations. Hence, a more accurate method for speech repair detection and correction was developed based on the structural similarity between the reparandum and its alteration.
- The PSMA searches for the structure that is modified or replaced by the alteration; or with which the current fully activated structure can be merged. The search for the previous matching structure helps to skip or remove interim disfluent structure such as filled pauses and editing terms and to deal with all types of the repairs described in Section 1.1.

1.4 Organization of the Thesis

This thesis is organized as follows: Chapter 2 provides an introduction to natural language processing. The Hybrid Connectionist Parsing (HCP) method and the use of temporary binding to deal with interim derivation are explained in Chapter 3. Different works on spoken language are explained in Chapter 4. The PSMA and its development process are addressed in Chapter 5. Experimental results are summarized in Chapter 6. Finally, conclusions and future work of this thesis are provided in Chapter 7.

Chapter 2 Natural Language Processing

In this chapter, the syntactic way to process natural language based on context free grammars is explained (i.e. context free grammars are used to generate syntactic structures). Natural language processing involves computation of both written and spoken language [23]. The knowledge sources used for natural language computation are syntax, semantics and morphology.

2.1 Context Free Grammars and Parsing

Context free grammars contain a set of rules used to describe a syntactic structure of sentences of a language [23]. The syntactic structure represents the parse tree generated for a sentence. Figure 2-1 shows the parse tree generated for the sentence "Book that flight". Context free grammar rules generated for the sentence, "Book that flight", is given in Example 2-1.

Example 2-1

1. S → VP
2. VP → V NP
3. NP → Art N
4. V → book | Book
5. Art → that
6. N → flight

Context free grammar rules contain non-terminal and terminal symbols, productions and the sentence symbol (S) [6]. Non-terminal symbols (i.e. 'S', 'V', 'VP', 'NP', 'Art' and 'N') in the preceding example represent grammatical categories and terminal symbols (i.e. 'Book', 'that' and 'flight') represent words of a sentence. Productions

represent rules, in which a non-terminal symbol (left-hand side of the rules) replaces grammatical categories or words on the right-hand side of the rule. Terminal symbols described are called lexical items. The rules that assign syntactic categories to these lexical items are called lexical rules, and the left hand side of the lexical rules represents part of speech tags (V, Art and N in the above example represent part of speech tags) or syntactic categories for the words.

Parsing is described as the process of deriving the syntactic structure of a sentence based on a given grammar. The syntactic structure represents the parse tree generated for a sentence. As the number of rules used by the parser increases, there will be more than one possible parse tree for a sentence. Syntactic disambiguation is a process of choosing the most probable parse for a sentence.

The essential constraint of the parse tree is that it has to cover all words of a sentence and the root node has to be a S. Parsing the sentence, "Book that flight", based on the above rules, will result in a parse tree as shown in Figure 2-1. The parse tree covers the entire sentence and the root node is represented by S. The parse tree shows that the sentence in the example is formed by a Verb Phrase (VP); the VP is formed by a Verb (V) and a Noun Phrase (NP). The NP is formed by an Article (Art) and a Noun (N). Finally, the speech tags for the words "Book", "that" and "flight" are V, Art and N respectively. Top-down and bottom-up parsing are the two different parsing strategies with the above constraints and it is explained in Section 2.1.1 and 2.1.2 respectively.

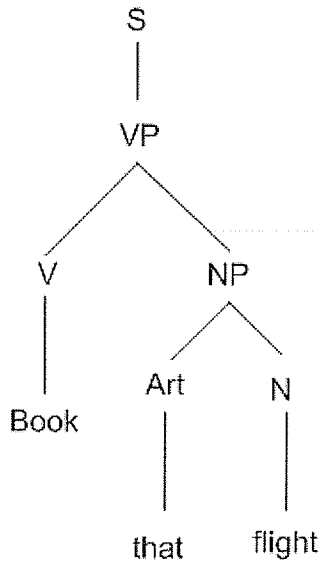


Figure 2-1: An example of a parse tree [23].

2.1.1 Top-down Parsing Method

In a top-down parsing method [23], the parser generates the parse tree from S. The rules given in Example 2-2 are used to generate a structure for the sentence, "Book that flight".

Example 2-2

1. S → VP
2. S → NP VP
3. S → Aux NP VP
4. VP → V NP
5. NP → Det N
6. NP → PropN
7. V → book | Book
8. Art → that
9. N → flight

Figure 2-2 represents the way in which a top-down parser generates its parse tree. Syntactic structures are generated by the parser starting from S.

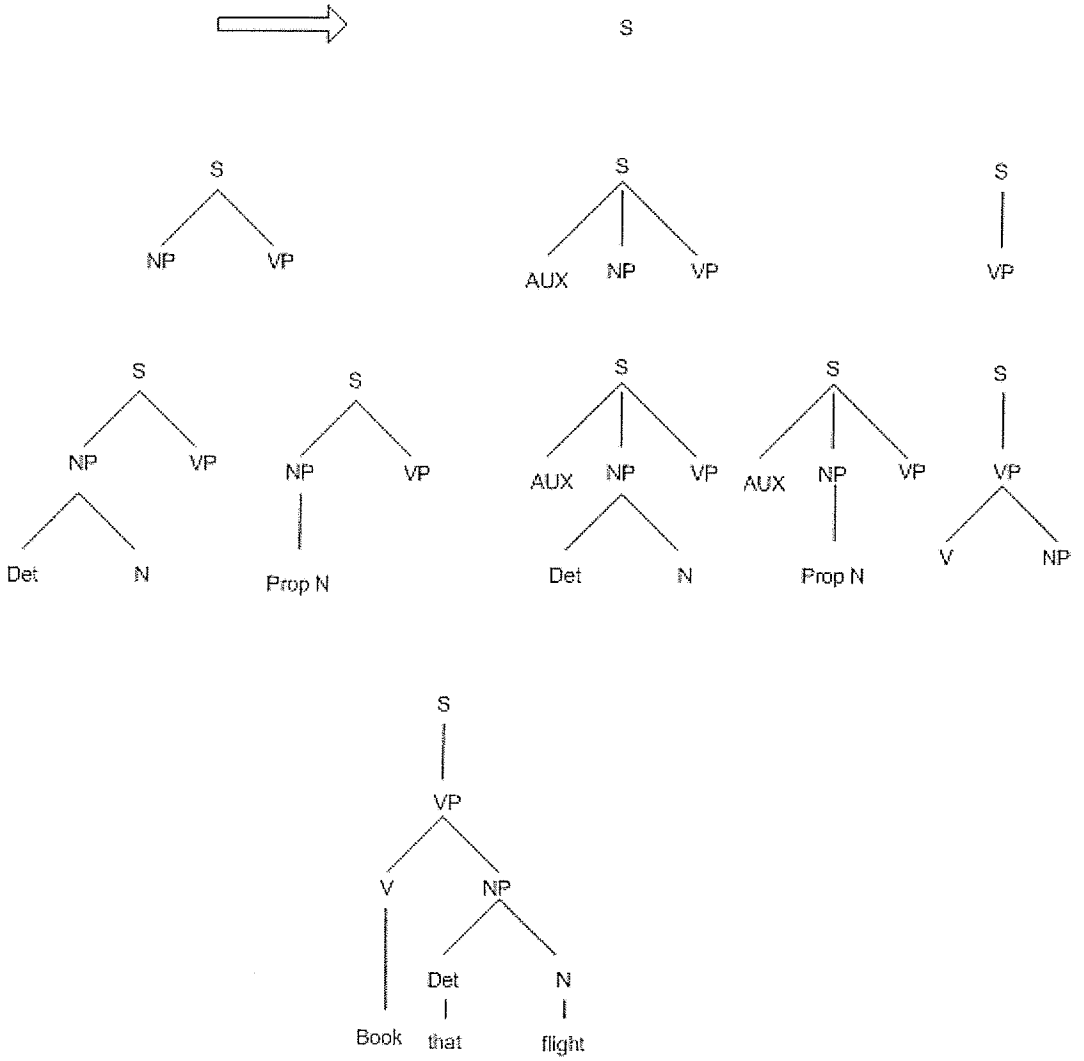


Figure 2-2: Different stages in a top-down parsing method [23].

In the second stage, parse trees are generated from rules with S (i.e `S` → `NP VP`, `S` → `Aux NP VP` and `S` → `VP`) as its left-hand side category. Thus, there are three structures generated from three rules. In the third stage, each of these structures is further expanded.

In each stage the leftmost-category (at the leaf level[2]) of a structure, which is not a terminal symbol, is expanded. The process of expanding the left-most leaf node continues in each of the structure until the leaf node of a structure contains only word categories. If the symbols match with the words of the input sentence, then the corresponding tree represents its parse tree. In the first four parse trees of the fourth stage, the categories of the left-most leaf node do not represent the category of the word "Book". If the input sentence is "Book the flight", then the fifth parse tree in the third stage will represent its parse tree. Because, the leaf nodes of the structure represent V and NP, where V represents the category for the word "Book" and NP represents the phrase "the book".

2.1.2 Bottom-up Parsing Method

In a bottom-up parsing method [23], the parse tree is generated by applying the rules starting from words of a sentence. The list of rules used in the bottom-up parsing method is given in Example 2-3.

Example 2-3

1. S ← VP
2. S ← NP VP
3. S ← Aux NP VP
4. VP ← V NP
5. NP ← Det N
6. NP ← PropN
7. V ← book | Book
8. Art ← that

[2] Leaf node represents the nodes in the lowest level of the tree. The root node represents the top-most node of the tree (sentence symbol "S")

9. N ← flight

The lexical rules (i.e. Rules: 7, 8 and 9) assign part of speech tags for words in a sentence. The lexical rules assign syntactic categories to these lexical items (i.e. words of the sentence) and the left hand side of the lexical rules represents part of speech tags (POS) for these words. The partial parse tree is then built based on the tags. Figure 2-3 represents different steps followed by a bottom-up parser. In Figure 2-3, the parser generates syntactic structure starting from the sentence "Book that flight".

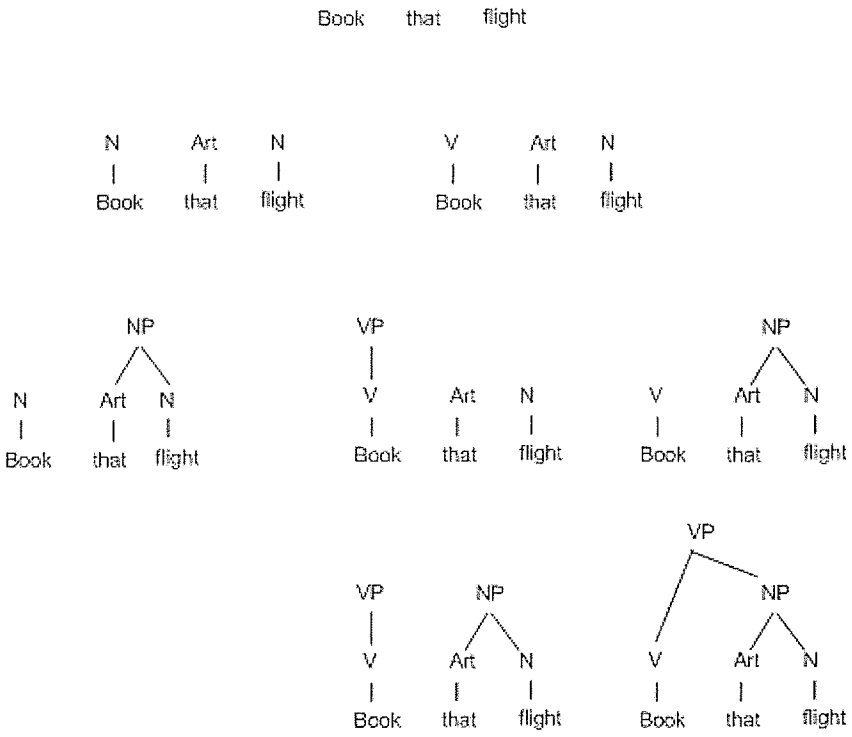


Figure 2-3: Different stages in a bottom-up parsing method [23].

In the second stage in Figure 2-3, partial parse trees represent the words and its tags. Due to the ambiguous nature of the word "Book", it has two different tags. Therefore, two separate partial parse trees are generated. In the next stage, different possible ways to generate the tree based on the application of rules are showed. In the fourth stage, the

structure formed by taking "Book" as N is dropped as the network cannot be further generated. In the fourth stage, there is a partial parse tree with VP followed by NP (first partial parse tree) and a partial parse tree formed by VP as its root node (second partial parse tree). Since, there is no rule with "VP NP" on the right-hand; the first structure on the fourth ply can be dropped. The final parse tree based on the rule S ← VP is shown in Figure 2-4.

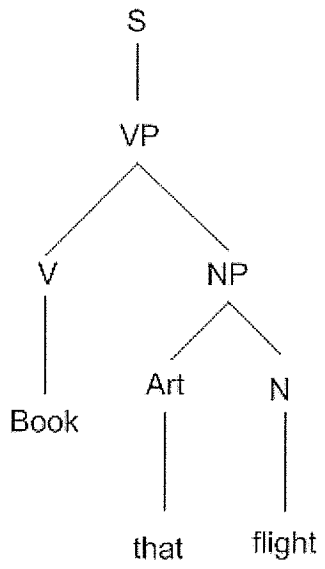


Figure 2-4: Parse tree for "Book that flight"

Chapter 3 Hybrid Connectionist Parsing (HCP)

In this chapter I will explain the Hybrid Connectionist Parsing (HCP) algorithm and the enhancement of the HCP using temporary bindings. The HCP [25] is an incremental parser, which uses the concept of traditional parsing algorithm and neural networks. The parsing algorithms described in Section 2.1.1 and 2.1.2 assumes that all the words of a sentence are present before the parser begins its process. Incremental parsing generates syntactic structures as each word is recognized. Such systems are very useful with respect to spoken language processing as the input words can be processed as soon as they are recognized.

3.1 Representation of a Mini-Network

A grammar rule can be represented in the form of a neural network which is called a mini-network [25]. The mini-network shown in Figure 3-1 is a two-layer neural network with a root node representing the left-hand side of the rule and the leaf nodes representing the categories on the right-hand side of the rule ($A \rightarrow B_1 \ldots B_n$). In Figure 3-1, leaf nodes represent lowest level nodes with no branches (i.e. $B_1 \ldots B_n$). The root node "A" of a network represents the topmost node in the network.

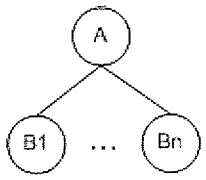


Figure 3-1: An example of a mini-network based on the rule $A \rightarrow B_1 \ldots B_n$ [25].

Each node is associated with an activation value. The connection weight between a leaf node and the root node is calculated using the formula 1/n, where n is the number of leaf nodes in the network. The activation value of a node takes a value between 0 and 1. The range between 0 and 1 determines the state of the node. A node which represents a syntactic category is fully activated if its activation value is 1.

3.2 Parsing of Natural Language Text

Parsing is described as the process of deriving the syntactic structure of a sentence based on a given grammar. Incremental parsing of a natural language text takes place as each word is recognized. As each word is recognized, the part of speech category to which this word belongs is determined and a mini-network is created. The mini-network created, is based on the rule *word* → *category* (i.e. `the` → `Art`). Transfer of activation from the fully activated node in the lower layer (i.e. the node containing the word) to all the nodes with which it is connected in the higher level takes place. Activation transfer in the network causes the root node containing the category to become fully activated. Rules with the category of the fully activated root node as the first category on the right-hand side of the rule are determined. Then, a new mini-network is created based on the selected rules, followed by merging the fully-activated root node with the leftmost leaf node of the new network. Activation transfer takes place in the new network. The process of generating and merging takes place successively in this fashion throughout the parsing process. Different stages of the HCP are shown in Figure 3-2.

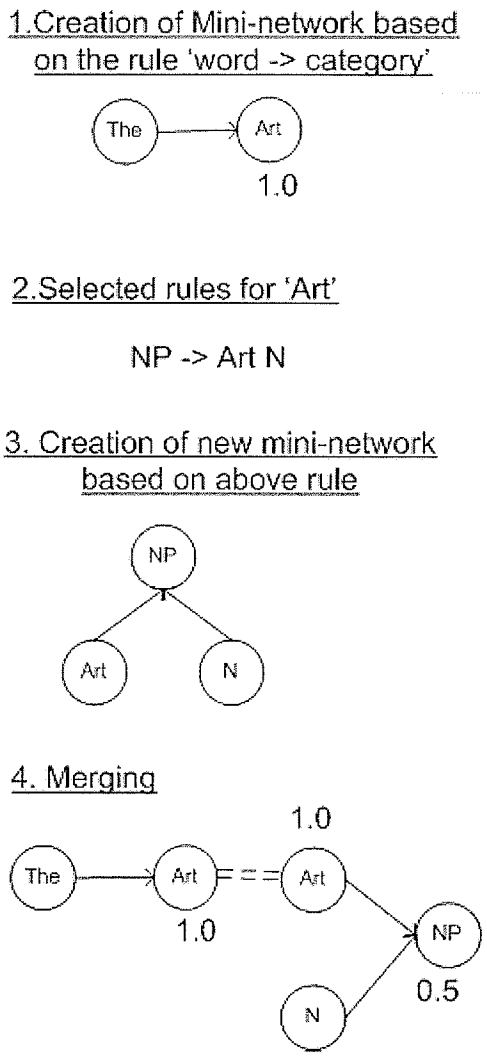


Figure 3-2 : Parsing process [25].

3.3 Representation of a Complex Mini-Network

A complex mini-network is used to replace the position of mini-network when more than one rule is selected based on the category of the fully activated node. There are cases, where more than one rule is selected for a given category. In Example 3-1, category "V" is used as the selection criterion.

Example 3-1

VP ← V

VP ← V NP

VP ← V PP

VP ← V NP PP

In this example, a complex mini-network is used to determine the correct rule from the competing rules. An example of a complex mini-network is represented in Figure 3-3. A complex mini-network is a three-layer network, with the leaf nodes representing all categories on the right-hand side of the competing rules, the middle layer represents hypothesis nodes and the root node represents the common left-hand side of the rule (all rules have the same category on the left-hand side). A hypothesis node is defined for each rule in the competing derivations. Derivation is a term used to denote the rules derived based on a selection criterion (i.e. "V" in the above example). In this example, the hypothesis node "H1" represents the rule "VP ← V", "H2" represents the rule "VP ← V NP", "H3" represents the rule "VP ← V PP" and "H4" represents the rule "VP ← V NP PP[3]".

[3] PP stands for prepositional phrase

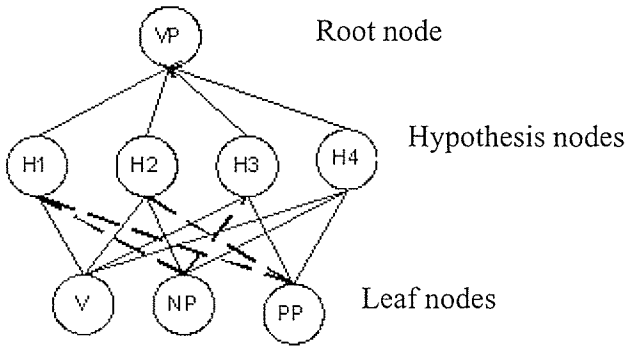


Figure 3-3: Complex mini-network [25].

There are two types of connections between the leaf nodes and the hypothesis nodes. The dotted connections represent inhibiting connections. These connections carry a negative connection weight (-0.1). Thus, if a leaf node associated with an inhibiting connection is activated, a negative weight is transferred to the hypothesis node associated with the connection and deactivates the particular hypothesis node. The connection weights are calculated using the formula 1/n, where 'n' represents the number of categories on the right-hand side of the rule represented by the hypothesis node.

The use of inhibiting connections and hypothesis nodes helps to determine the correct rule based on the leaf nodes that are activated. If the leaf nodes V and NP are activated, the hypothesis node "H2" is activated. The second "VP" rule will be used.

In Section 3.1, 3.2 and 3.3, the way in which a grammar rule is represented, the incremental parsing algorithm and finally the use of complex mini-network to deal with interim ambiguous situation are shown. As the number of rules used by the parser increases, the parser has to deal with more ambiguous situations. The HCP is enhanced by using *temporary bindings* and *tags* to handle competing interim derivations.

3.4 Enhanced Version of the HCP

Enhanced version of the HCP deals with ambiguous situations that occur during parsing. The Hybrid Connectionist Parser uses a complex mini-network to represent the rules selected based on the category of the fully activated node. A complex mini-network can represent rules having the same left-hand side category. Example 3-2 represents rules having the same left-hand side category (i.e. VP). The rules in Example 3-2 are selected based on the category verb i.e. "V"

Example 3-2

VP ← V

VP ← V NP

VP ← V PP

VP ← V NP PP

The HCP is enhanced to deal with rules with different left-hand side categories. As a partial parse tree is generated for an input word, there can be more than one way in which the incremental parser can make its next move. The temporary bindings between networks are used to represent different ways in which a partial parse tree can be generated. The bindings are also numbered and these numbers are called *tags*. Tags represent identifier for a path. After parsing the input sentence, different paths are traversed to derive the sentence structure, which covers all words of the input sentence.

3.4.1 Representation of Temporary Bindings and Tags

The temporary bindings are used to represent different interim derivations in parallel. Therefore, each path represents the way in which a parse tree can be generated.

Scenario 1 and Scenario 2 mentioned in this section, describe the use of temporary bindings to deal with different ambiguous situations.

Scenario 1:

In this scenario, temporary bindings are used to deal with rules having different left-hand side categories. Example 3-3 shows an example for rules with different left-hand side categories (i.e. NP and S). The rules in Example 3-3 are selected based on the category noun phrase (NP).

Example 3-3

NP ← NP PP

S ← NP VP

If the rules have different left-hand side categories, the rules with the same left-hand side category will be grouped together. A mini-network will be created for each of these groups and the fully activated root node will be connected to each of these groups by means of a temporary binding. Binding of the fully activated root node to the leftmost leaf nodes of the newly generated mini-networks is depicted in Figure 3-4.

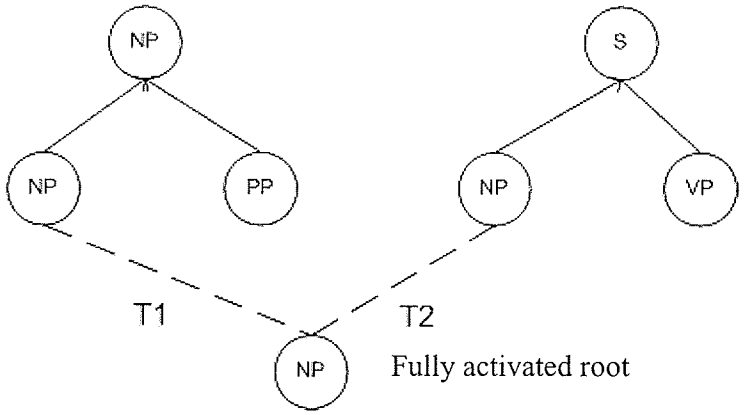


Figure 3-4: Binding of the fully activated root node to the leftmost leaf nodes of the newly generated mini-networks.

Scenario 2:

In this scenario, temporary bindings are used to deal with interim derivations. When a root node is fully activated, the parser can continue processing either by binding to the previous expecting node or by generating a new network. Temporary bindings will be used to represent different ways in which a parse tree can be extended. Figure 3-5 shows a situation where a fully activated noun phrase (NP) is bound to the previous expected node and the new mini-networks[4]. The previous expecting node in Figure 3-5 will represent the NP node present in the VP structure.

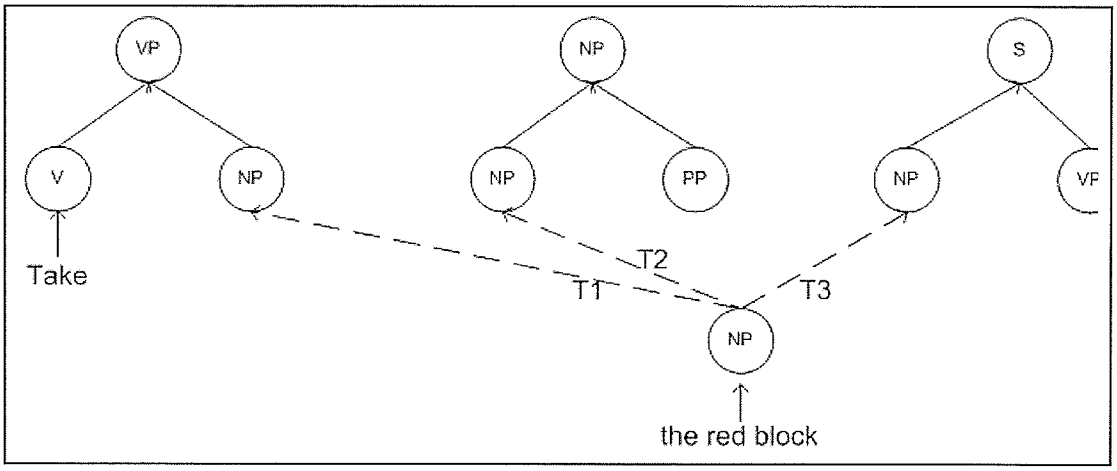


Figure 3-5: Binding of the fully activated root node to the previous expecting node and to the newly generated mini-networks.

[4] The mini-networks represent the two networks formed based on the following rules

NP ← NP PP

S← NP VP

A tag is assigned to each temporary binding. Tags describe the nodes that are bonded to the fully activated node and the branch it uses. When there is a binding from a fully activated root node then the node is assigned with a unique number. The numbers are assigned in an ascending order from 1 to n, so that the fully activated nodes have a unique value. Each of the branches from the fully activated root node will be assigned a unique value from 1 to n, representing the branch number from the node (i.e. first branch, second branch and so on). In Figure 3-5, the NP node that is bound to the previous expected node and the new mini-network will have a unique value of '1'. 'T1' represents '1, 1' that is the first branch from the NP node with the value of '1'.

3.4.2 Parsing Using the Enhanced Version of the HCP

The hypothesis of an incremental parser is that the parse tree has to be built as each word is read. This section describes the parsing algorithm for the sentence "I took the book" using the enhanced version of the HCP. The HCP generate syntactic structures as each word is recognized. The partially parsed structure generated after parsing the word "I" is shown in Figure 3-6. In this partially parsed structure, temporary bindings are used to connect mini-networks, followed by the assignment of tags. The fully activated pronoun (PN) node (syntactic category of the word "I") is connected to the mini-network created using the rule "NP ← PN". Activation transfer takes place in the network. The NP is fully activated and mini-networks are generated based on the rules selected i.e. with NP as the first category on the right hand side of the rule. The rules selected with noun phrase as the first category on the right hand side of the rule are shown in Example 3-4.

Example 3-4

NP ← NP PP

S ← NP VP

The selected rules in Example 3-4 have different left hand side categories. A mini-network is created for each of the selected rule and a temporary binding is used to connect the fully activated root node to each of the generated network. The next stage after binding the generated network to the root node is activation transfer.

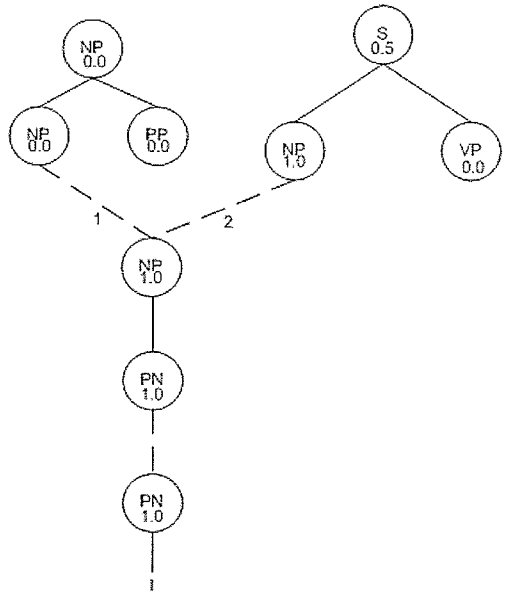


Figure 3-6: Partial parse tree generated for "I"

Figure 3-7 depicts different process performed by the activation transfer function. The network on the left hand side of Figure 3-7 gives the activation value of different nodes in the network after transferring activation through the first mini-network i.e. the first branch is selected for activation transfer. The leftmost noun phrase node in the first network has an activation value of 1 and the network is partially activated. If the network associated with the first branch is activated, the leftmost node associated with the second

branch is deactivated. Similarly, the activation transfer takes place in the network on the right-hand side of Figure 3-7.

Activation transfer takes place in the first branch

Activation transfer take place in the second branch.

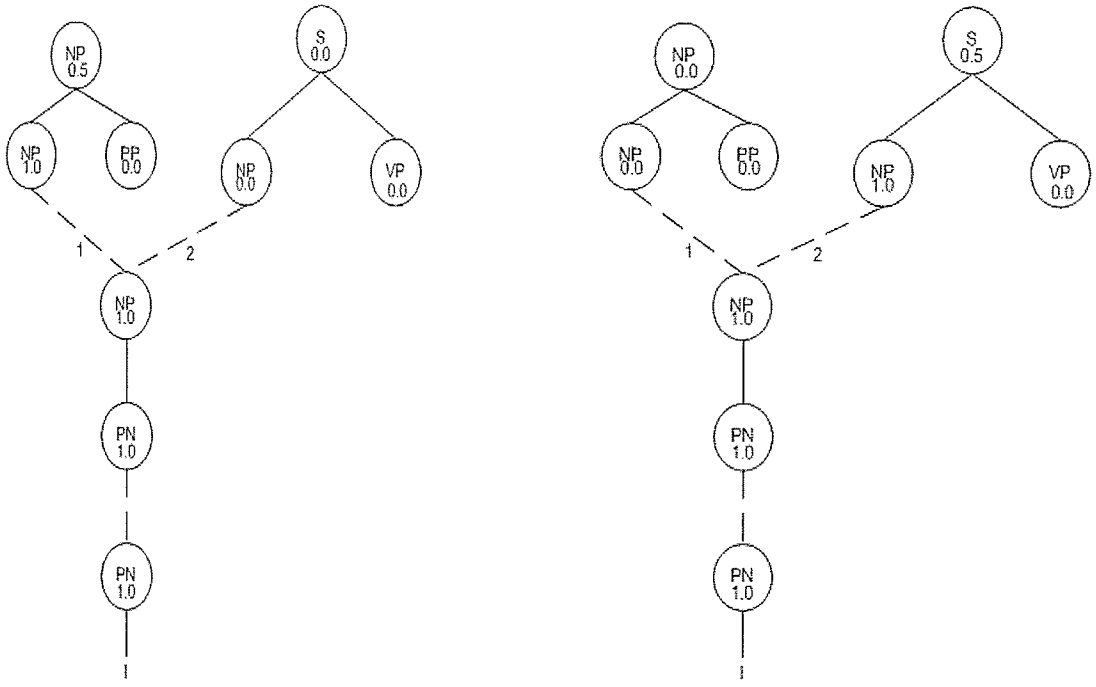


Figure 3-7 : Activation transfer in a partial parse tree.

In the left-hand side or in the right-hand side of Figure 3-7, if a mini-network associated with a branch is activated then other mini-networks associated with the remaining branches from NP will be deactivated. Further structures cannot be extended from the network[5] and the parser read the next word "took". The partial parse tree for "I took" is shown in Figure 3-8. The network shown in Figure 3-8 consist of a fully activated NP (i.e. NP activated based on the rule "NP ← PN") and VP (i.e. "VP"

[5] The network can be extended only if the root node is fully activated.

activated based on the rule "VP ← V"). On further reading input words, the parser will generate a partially parsed structure for the noun phrase "the book".

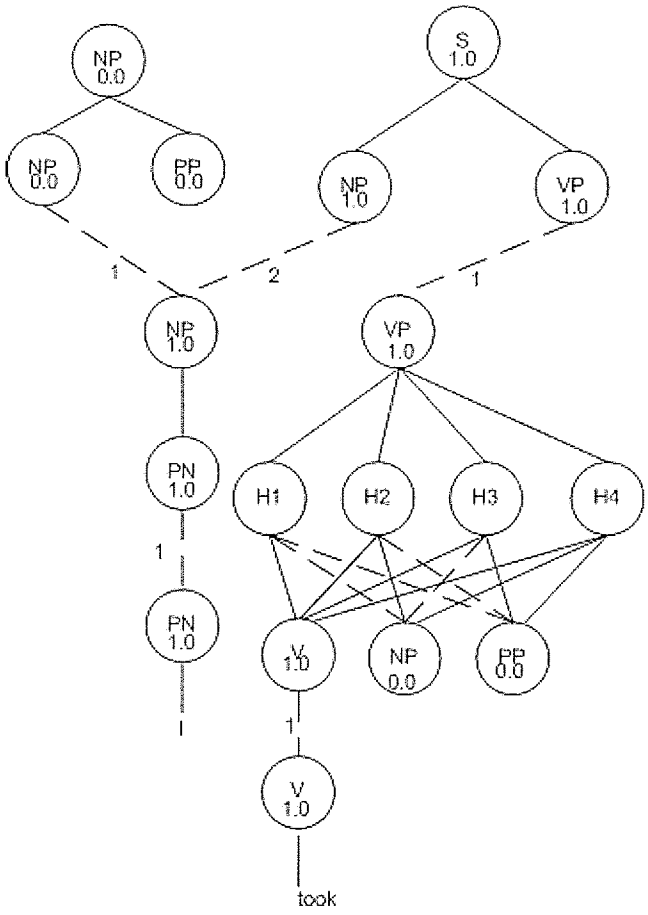


Figure 3-8: Partial parse tree for "I took".

The fully activated noun phrase can be bound to previous expecting noun phrase node or can be extended. Figure 3-9 shows the structure, where NP can be bonded to the previous expecting node.

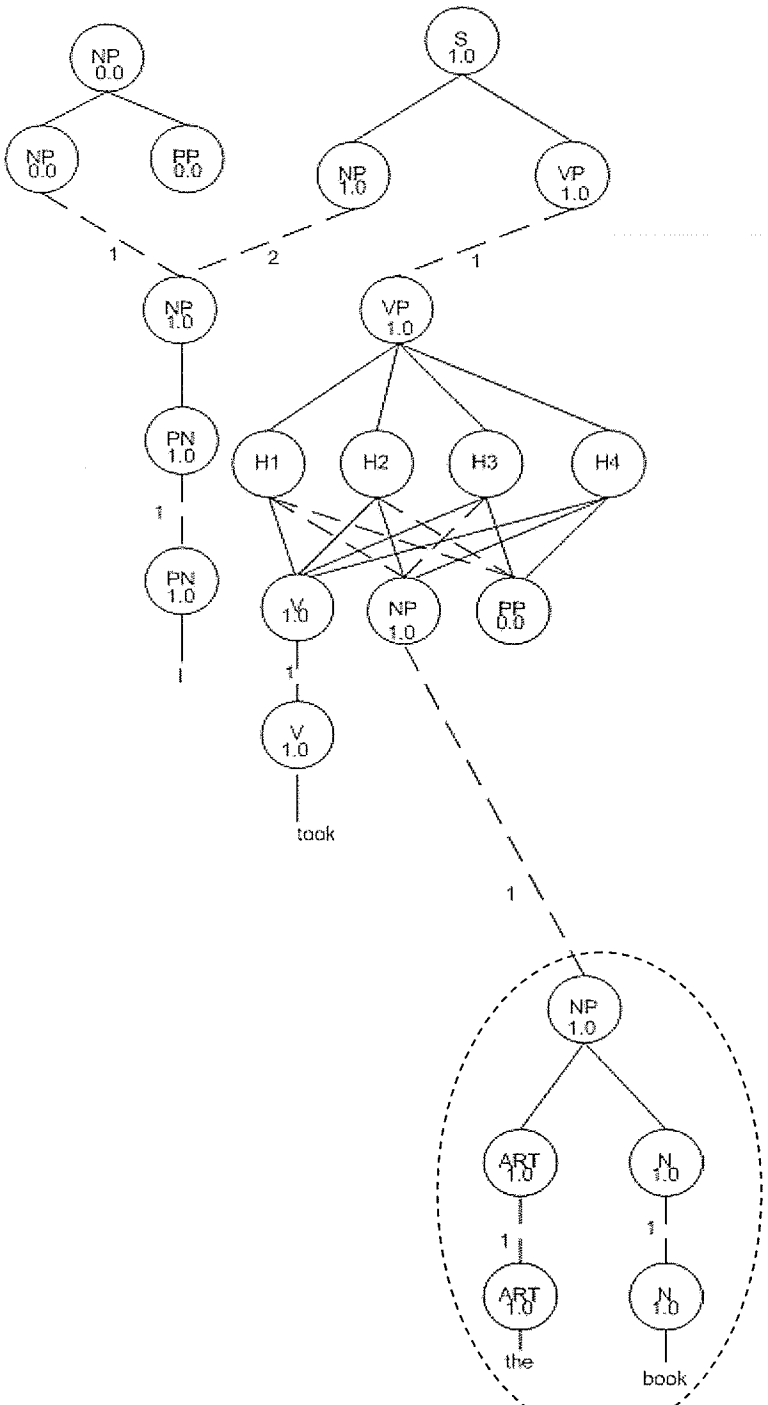


Figure 3-9: A fully activated root node is bound to the previous expected node.

After binding, transfer of activation takes place in the network. VP remains in its activated state based on the rule "VP ← V NP". The parse tree for the sentence "I took the book" will be derived as,

S[NP[PN[I]] VP[V[took] NP[ART[the] N[book]]]]

The fully activated noun phrase structure (i.e. NP[the book]) can be extended by generating new mini-networks. All possible rules with NP as the first category on the right-hand side of the rule will be selected,

NP ← NP PP and

S ← NP VP

Mini-network will be created for each rule and it will be bound using a temporary binding. Figure 3-10 shows two mini-networks and it is bound to the fully activated noun phrase node. In Figure 3-10, the two mini-networks are bound using the branch 2 and 3 to the fully activated root. After binding, activation transfer takes place in the network. Activating the network that is selected and deactivating the remaining networks will be an important part of this algorithm. Therefore, the leaf node that is connected using branch 1 from noun phrase i.e. NP [the book] is deactivated. The node connected using temporary binding 2 from the NP node generated for the input text "*the book*" will be activated and the networks associated with the remaining bindings are deactivated. The activation value for the entire parse tree will be recalculated. The activation state of the network is shown in Figure 3-10. In Figure 3-10, the noun phrase structure generated for the input words "the book" is extended based on the rule "NP ← NP PP" (second branch). The root node of the network connected using branch 2 is expecting a preposition phrase (PP) to become fully activated.

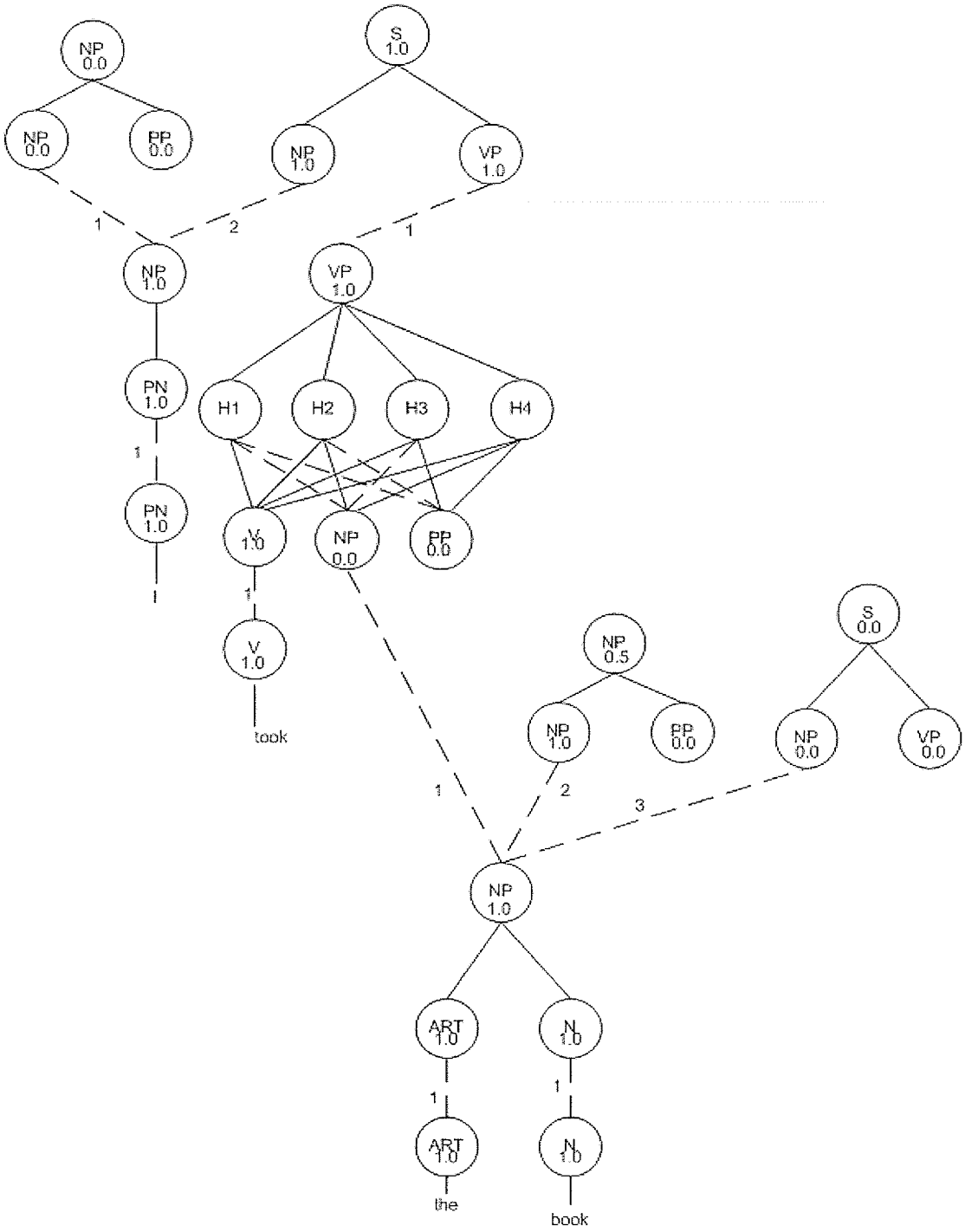


Figure 3-10: The state of the network after selecting temporary binding 2.

The network connected using the second branch will not be useful as there are no input words. The state of the network after selecting the network connected using the temporary binding 3 is shown in Figure 3-11.

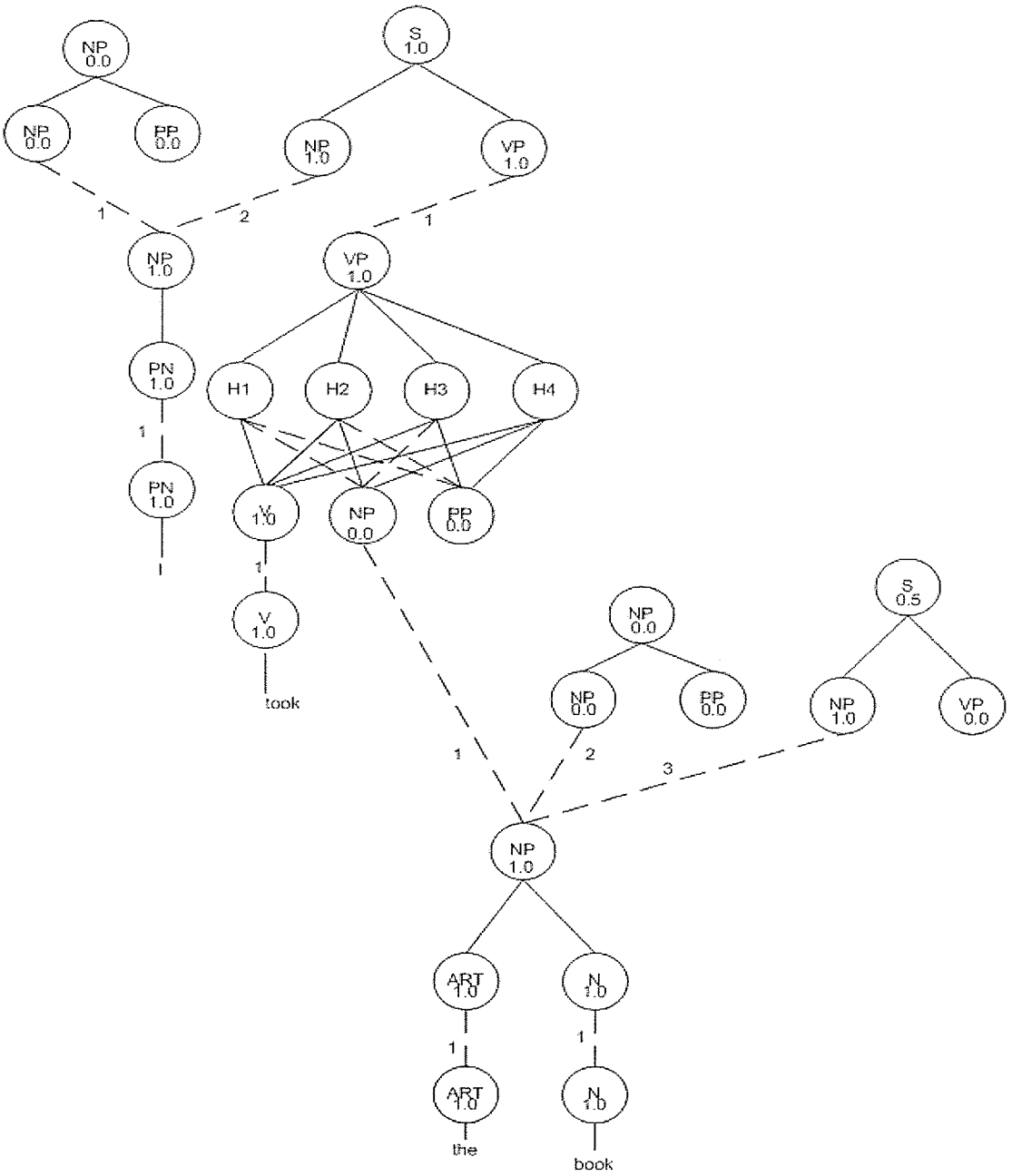


Figure 3-11: The state of the network after selecting temporary binding 3.

Figure 3-10 and Figure 3-11 show that the networks connected using branch 2 and branch 3 from the fully activated noun phrase node (i.e. NP[the book]) are not useful and these derivations are called interim derivations. After parsing the sentence, the interim derivations that are not useful will be neglected and the network connected using binding 1 will be considered. The final parse tree will be

S[NP[PN[I]] VP[V[took] NP[ART[the] N[book]]]]

3.5 Summary

In this chapter, the Hybrid Connectionist Parsing Algorithm is used to derive syntactic structures for sentences. This chapter started with the description of a mini-network, which is used to represent a grammar rule. A complex mini-network is used to process different competing rules with the same left-hand side categories.

The enhanced version of the HCP uses temporary bindings and tags to deal with ambiguous situations that occur during parsing.

Chapter 4 Related Work on Spoken Language Processing

In this thesis, I work on disfluencies in spoken language. Previous works on spoken language have used syntactic information [10], [20], [29] and [12] prosodic signals [17], [28], [33] and [41] and other processing methods like pattern matching to detect and correct repairs. In this chapter, different knowledge sources and methods used to detect and correct repairs are explained.

4.1 In-Parser Speech Repair Identifier

In the case of an in-parser speech repair identifier, a parser is used to assign syntactic structures to a sentence based on grammar rules. Unlike written text, a parser will not be able to derive a sentence structure for a spoken language sentence which covers all the words in the sentence (due to disfluencies). This section describes the different methods that use parser to process disfluencies in speech.

4.1.1 The work of Core

Core [10], in his work on spoken dialog, has designed a parser to handle speech repairs based on metarules. These metarules specify different patterns of speech repairs that occur in spoken language. Hence, these rules help to form a complete phrase structure by skipping the reparandum or other editing terms. In this system, Non-interference metarule, Editing term metarule and Repair metarule are used to handle different situations. In the case of repair metarule, the possible reparandum start and end points are calculated to form a phrase structure around the reparandum. Based on the results of in-parser speech repair identifier, the system is able to detect 237 out of 495

speech repairs using the word and the part of speech tag information. The Correct reparandum start is detected for 187 out of 237 using only word matching and 183 out of 237 using word and part of speech similarities.

4.1.2 The work of Hindle

Hindle [20] uses a set of rules to handle the repairs in speech. His work is based on a hypothetical edit signal, assumed to be a kind of phonetic signal, which indicates the presence of a repair. The edit signal marks the end of the reparandum or the interruption point. Hindle formulated four rules to detect and correct repairs in speech:

1. Surface Copy Editor
2. Category Copy Editor
3. Stack Copy Editor
4. Handling Fresh Starts or Restarts

The surface copy editor rule is used to remove the repetition of words, where the first word is followed by an edit signal. The category copy editor rule and the stack copy editor rule works on the syntactic categories to remove repeated and incomplete structures. The last rule handles fresh starts by looking for phrases such as *you know, like I said* etc. after an edit signal which is followed by the speaker's correction. Experimental results show that there is only a 3% rate of failure when tested on a corpus of transcribed speech. The disadvantage of this work is that it assumes an edit signal to detect the presence of a repair.

4.1.3 The work of Mckelvie

Mckelvie [29] uses a syntactic method to deal with disfluent speech. The grammar rules he uses are initially formulated for fluent speech and later modified to include disfluent patterns present in spoken language. He proves that syntactic rules can be developed for spoken language. However, Mckelvie's work does not explicitly show the number of repairs that are detected by the parser and the number of false alarms detected.

4.1.4 The work of Dowding et al.

Dowding et al., [12] developed a natural language system, GEMINI. This system is used to parse a spoken language utterance using a bottom-up parser and an utterance level parser. Both syntactic and semantic information are used to process the utterance. The bottom-up parser uses the syntactic and semantic rules to generate structures which encompass syntactic and semantic information. Then an utterance level parser is used to derive a semantically correct structure. If the utterance level parser fails to derive an acceptable structure for the utterance, then a correction phase is used to detect and correct the repairs in speech. Their training set contained 178 repairs and the system is able to detect 89 repairs, and 81 out of 89 were corrected. Their testing utterance contained 26 repairs and the system is able to detect 11 repairs, and corrected 8 out of 11 repairs.

4.1.5 The work of Wermter and Weber

Wermter and Weber [43] describe a flat screening method to handle spoken language utterance. The system (SCREEN) uses a flat representation for syntactic and semantic analysis and artificial neural networks to deal with ambiguous structures and

repairs in the utterance. The correction part of the system is responsible for speech repair detection and correction. The system deals with pauses, interjections, repetitions of a word and phrase. Word corrections are also dealt, where two different words have same syntactic category.

4.1.6 The work of Lavie

Lavie [26] has developed a new parsing algorithm (GLR*) to deal with spoken language. The parser is designed to handle repairs and problems that occur due to grammar coverage. To deal with the above situations, the parser skips words and finds the maximum set of words which form a complete structure. This system does not perform any speech repair detection and correction.

4.2 Other Methods to Process Speech Repairs

4.2.1 The work of Bear et al.

Bear et al. [3] worked on different sources of information to detect and correct repairs in speech. In this work, a simple pattern matching technique, the syntactic and semantic information of a parser, and acoustical information are used. They conclude that any single source of information is not sufficient to deal with repairs. They say that the above sources of information (i.e. pattern matching, syntactic and semantic information, and acoustical information) can be integrated to deal with repairs in speech.

4.2.2 The work of Heeman et al.

Heeman and Allen [15] have focused on the speech recognition problem. They say that the problem of recognizing part of speech (POS) tags, discourse markers,

detecting disfluent parts in speech and intonational phrases are intertwined. Moreover, the speech recognition problem is also connected to the task of recognizing POS tags. Hence, they solve the above problem of detecting and correcting repairs, finding utterance ends and discourse makers at the speech recognition phase. With this system, Heeman and Allen solved 66% of speech repairs and they had a precision of 74%.

Heeman et al. [18] combines the detection and correction algorithm to confirm whether a repair has actually occurred. Speech repairs are detected using transition probabilities of word categories. After repairs are detected, the correction algorithm uses word match information to detect the span of the reparandum. Finally, they use the correction process on the alternative structures generated by the repair detection algorithm to confirm whether the speech repair has occurred. By combining the detection and correction algorithm, the value of recall (in the repair detection) increased from 74.9% to 80.8% for modification repair. In the case of fresh starts, there isn't significant increase in the value of recall. The value increased from 57.0% to 58.5%.

4.2.3 The work of Nakatani and Hirschberg

Nakatani and Hirschberg [33] determine the use of acoustic and prosodic cues to detect and correct repairs in speech. The system is tested with 202 utterances containing 223 repairs. They consider every point between words (w_i, w_j) as an interruption point and various features are examined to determine the possibility of a repair. Some of the features include pause duration, words fragments, filled pause, pattern matching between w_i and w_j. 186 repairs are detected with a recall of 83.4%. 177 of the repairs are detected with the help of word fragments and pause duration. In their next experiment, the word

fragments are not considered. In this test, 174 repairs are detected correctly with 78.1% recall.

4.2.4 The work of Shriberg et al.

Shriberg et al. [40] show that, prosodic information can provide better results to detect repairs in speech than other methods that use lexical information. Sheiberg et al. use features like the duration of pause, the duration of vowels, the fundamental frequency, and the signal to noise ratio to detect repairs. These measurements are present in the speech data that are used for training. Finally, a decision tree is used to find the point (interruption point) that follows a filled pause, repetitions, repairs and false starts from other points. Shriberg et al's., work gave a better result to detect filled pause than the SRI's recognizer.

Shriberg [39] discusses the presence of regular patterns in speech. Despite the traditional view that difluencies are irregular events, the aim of this work is to show that these utterances tend to have a regular structure in different dimensions.

Hindle [20] assumed an edit signal to detect the presence of a repair. These works [10], [12], [18] and [38] on spoken language processing do not address the phrase structure parallelism (similarity in the phrase structure between the reparandum and the repair) to detect and correct repairs. Core [10] identified phrase structure parallelism as an area which has the potential to process repairs.

Some systems [15], [17], [28], [40] and [41] attempt to identify repairs during the speech recognition phase (pre-parser speech repair identification) or use prosodic signals to detect and correct repairs in speech. In this thesis, I will work at the syntactic level to

detect and correct repairs. A syntactic method to process disfluencies will be advantageous because the use of syntactic information of the language and the structure of the speech repair are the same for all corpora of spoken language.

4.3 Summary

In this chapter, different works on spoken language processing are explained. We classified the works based on the methods which use a parser to deal with repairs in speech. The remaining works which uses statistical method, prosodic cues, and pattern matching techniques are described in Section 4.2.

Chapter 5 The Phrase Structure Matching Algorithm

The Phrase Structure Matching Algorithm (PSMA) is based on the concept of Graph Matching Algorithm [25]. The Graph Matching Algorithm is a system that is used to correct the disfluent structure present in an utterance by detecting structural similarity between the reparandum and its alteration. An incomplete and a complete structure are taken by the Graph Matching Algorithm and are used to detect the intended sentence structure. For example; "Peter takes the red, the blue block" contains a disfluent structure that the Graph Matching Algorithm detects and corrects. The Graph Matching Algorithm infers that the "the blue block" is the corrected part of the utterance. Example 5-1 demonstrates the bracketed notation of this utterance and how it is being understood by the Graph Matching Algorithm. In Example 5-1, the complete noun phrase (NP) structure after the comma represents the alteration. The Graph Matching Algorithm searched the utterance for a previously incomplete noun phrase structure before the comma, and then the complete NP overlapped the incomplete NP. The previous incomplete NP structure is underlined in Example 5-1.

Example 5-1

S[NP (PR-peter) VP (V-takes) NP (DET-the ADJ-red NIL)] , [NP (DET-the ADJ-blue N-block)]

The PSMA which is based on the existing Graph Matching Algorithm was altered to deal with different possible situations that occur during parsing. The PSMA was tested using HCRC Map Task Corpus, a collection of spoken language transcripts. By integrating the PSMA with a parser the Graph Matching Algorithm was then able to detect and remove disfluent parts of a more complicated utterance. The Graph Matching

Algorithm which had not been tested on a corpus, benefits from the PSMA's testing, by creating a more finely tuned Graph Matching Algorithm.

Unlike a written text, a parser will not be able to derive a sentence structure for a spoken language sentence. The presence of disfluency in spoken language is the reason for phrase structure breakage. The PSMA processes (not "deals with") partial parse trees generated by the parser to detect and remove the disfluent structure present in the utterance.

This thesis uses two experiments to demonstrate the efficiency of the PSMA to detect and remove disfluency in speech. Section 5.1 describes the first experiment, in which the PSMA is triggered after deriving a partially parsed structure for the entire utterance. Section 5.2 describes the second experiment, in which the PSMA is triggered whenever disfluency is detected and enough structural (syntactic) information has been gathered to derive a proper structure.

5.1 Post-Processing of Partially Parsed Structures of a Sentence

In the post-processing method, the PSMA is triggered after parsing the entire utterance. Even though, the presence of a disfluency can be detected at the point where there is a structural breakage, the reparandum onset can be detected in a number of ways.

1. Based on the structural similarity between the alteration and its reparandum.
2. Based on the connection (with respect to syntactic structure) between the corrected part of the utterance and the initial words of a disfluent sentence.

The PSMA proceeds in two stages,

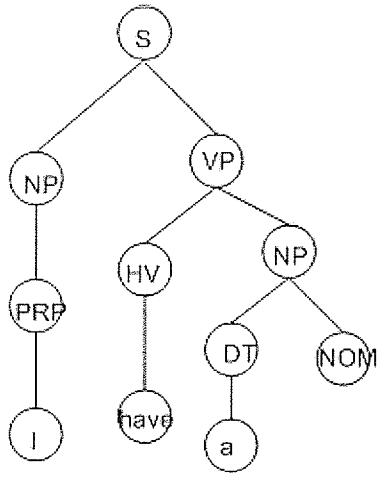
1. Based on finding the similarity between the reparandum and its alteration.
2. If there is no similarity, then the connection between the alteration and the initial parts of the utterance are used to determine a complete sentence structure.

A spoken language utterance is processed in two stages. In the first stage, partially parsed structures are generated by the HCP. Then the partially parsed structures are processed by the PSMA to form a complete sentence structure.

5.1.1 Derivation of Partial Parse Trees by the HCP

The HCP is an incremental parser, which generate a syntactic structure as each word is recognized. Due to the presence of disfluencies in speech, a phrase structure breakage occurs and the parser continues to parse the remaining parts of the utterance. The partial parse tree generated on parsing the initial part "I have a" of the utterance "I have a ... I have got a picket fence" is shown in Figure 5-1.

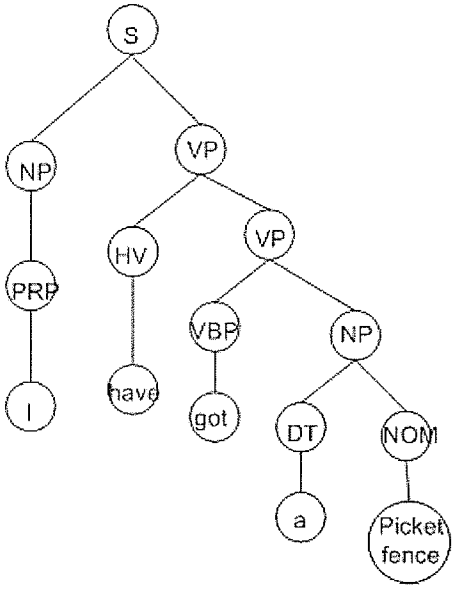
In Figure 5-1, there is an incomplete noun phrase (NP) formed by the word "a" and the parser will expect a nominal (NOM) for the NP to become complete. When the parser receives "I" as the next word, the next incoming structure is a pronoun (PRP). Since there is no previous expecting PRP node, the parser continues to parse the remaining utterance "I have got a picket fence". The complete sentence structure generated for "I have got a picket fence" is shown in Figure 5-2.



Bracketed Notation

S[NP[PRP[I]] VP[HV[have] NP[DT[a] NOM[]]]]

Figure 5-1: Partial parse tree for the initial part (reparandum) "I have a" of the utterance "I have a I have got a picket fence"



Bracketed Notation

S[NP[PRP[I]] VP[HV[have] VP[VBP[got] NP[DT[a]

NOM[picket fence]]]]]

Figure 5-2: Partial parse tree for *"I have got a picket fence".*

After parsing the entire utterance, an incomplete sentence structure is formed by the part of the utterance "I have a", followed by a complete sentence structure formed using the remaining parts of the utterance "I have got a picket fence". The PSMA is triggered after parsing the utterance "I have a … I have got a picket fence". The following information is derived from the structure of the utterance generated by the parser:

1. an incomplete sentence structure,
2. a complete sentence structure, and
3. the interruption point, where a breakage in the phrase structure takes place.

The PSMA searches for the matching structure with the same label as that of the root node of the complete sentence structure. When the root node label of the complete and the incomplete structure match, then the similarity is derived. Finally, the structure represented by the incomplete sentence structure is replaced by the complete sentence structure. A description of the search process used to detect the span of the reparandum (i.e. previous matching structure) is given in the next section.

5.1.2 Detection of the Span of the Reparandum

Detection of the span of the reparandum forms the major part of the PSMA. The portion of the utterance, which the user repeats or corrects, will be detected at this stage.

The basic function of the matching process is to take a fully formed complete structure, after the *interruption point* and check for a matching phrase. A search is made in the structure before the interruption point for a similar root category.

The user need not repeat the phrase they have uttered. The search for a previous matching structure might fail, because, the structure of the corrected utterance will be different. The next phase will be to search for the leftmost open node with the same category as the fully activated phrase (see Section 5.1.2.2). In this way, the corrected part of the utterance can complete the initial part of the sentence. Levet [27] says that, the corrected part of the utterance has a structural relation with the initial part of the utterance. According to Levelt [27], there are three stages in making a self-correction. In the first stage, the speaker monitors his own speech and interrupts it

when he detects an error. In the second stage, editing terms like "uh" and "um" are uttered by the speaker. In the third stage, the user corrects himself and produces a repair. Levelt says that the correction part of the utterance is structurally related to the initial part of the utterance

5.1.2.1 Searching for Previous Matching Structure

In this section, the PSMA detects the span of the reparandum by looking at the structural similarity between the corrected part and the initial part of the utterance. A search is made in each structure preceding the fully activated phrase for a matching root node category. When a match occurs, then the structural similarity between the detected structure (i.e. reparandum) and the fully activated phrase representing the corrected part of the utterance helps to detect the correct matching structure.

Figure 5-1 represents the reparandum "I have a". Figure 5-2 shows the structure generated on parsing the remaining part for the utterance "I have got a picket fence". In the post-processing method, a complete structure which covers the maximum number of words from the end of the sentence is taken as the corrected part of the utterance. A search is made in the initial part of the utterance to detect the span of the reparandum. Four different networks that are searched by the PSMA are shown in Figure 5-3. The search starts with the structure formed by the word "a" and its category "DT. The root label (i.e. DT) does not match with the root node of the sentence structure (S). The next network created based on the category of the word "a" is chosen. The second network in Figure 5-3 has NP as its root node label, which does not match with the sentence symbol i.e. "S". The PSMA then searches for the next higher category based on NP. Since the noun phrase "NP [DT[a] NOM [ ]]" is part of the verb phrase (VP), the next root node

that will be searched will be the VP structure. The third network selected is shown in Figure 5-3. Similarly, the fourth network in Figure 5-3 shows that the next root node that will be searched will be a sentence structure. The root node label of the selected structure (i.e. fourth network in Figure 5-3) matches with the root node of the complete sentence structure after the interruption point. The PSMA will then determine the similarity between the sentence structure for the initial part of the utterance "I have a" and final complete sentence structure for utterance "I have got a picket fence". Figure 5-3 demonstrates the four different networks searched by the PSMA to detect the span of the reparandum. The fourth diagram in Figure 5-3 illustrates the correct structure as chosen by the PSMA.

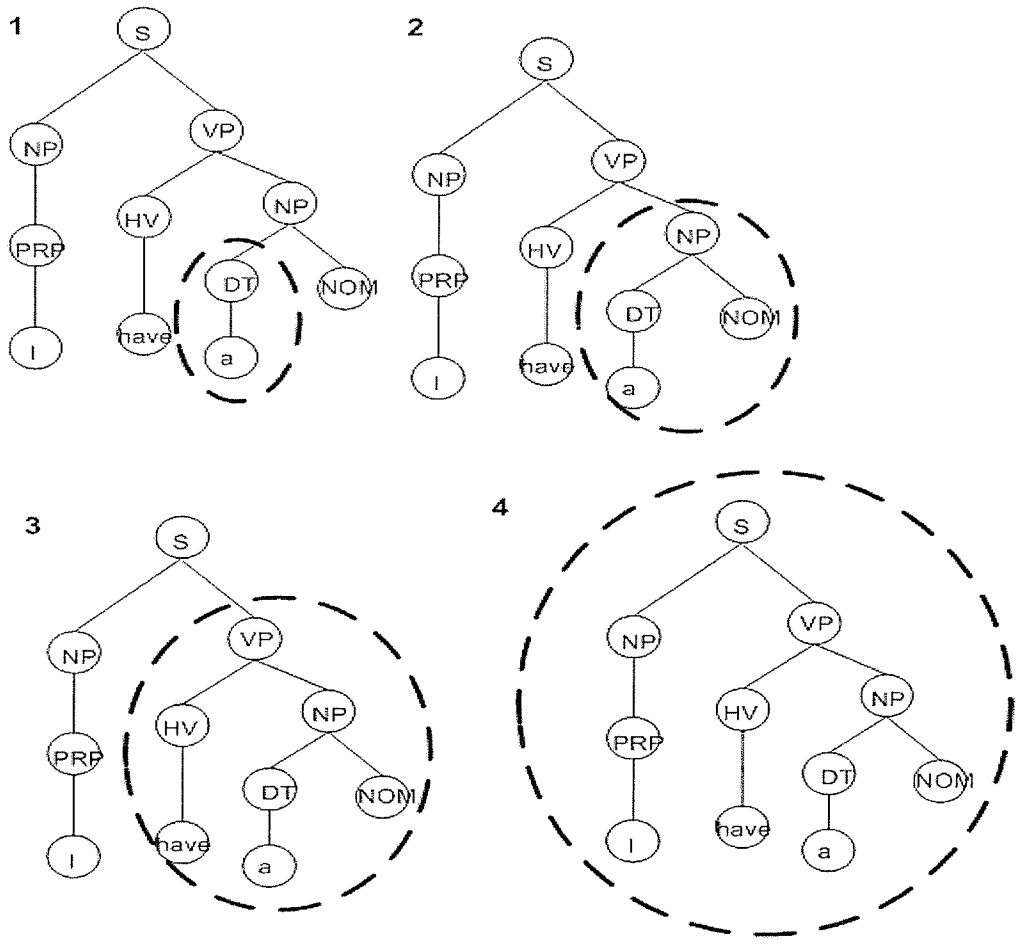


Figure 5-3: Four different networks searched by the PSMA.

Figure 5-3 shows four different networks that are checked to detect the span of the reparandum. The fourth network is selected as the root node of the network has a matching sentence symbol (S). After selecting the network with the same root node symbol as that of the correction, the networks representing the correction and the reparandum are traversed to determine the similarities. To check for similarity, a top-down traversal of the structures shown in Figure 5-4 is then done. As the network is traversed, the similarity between the nodes at each stage is determined. Two structures representing the correction and the reparandum are then checked for similarities. In the utterance "I have a … I have got a picket fence", there is a similar root node representing the S.

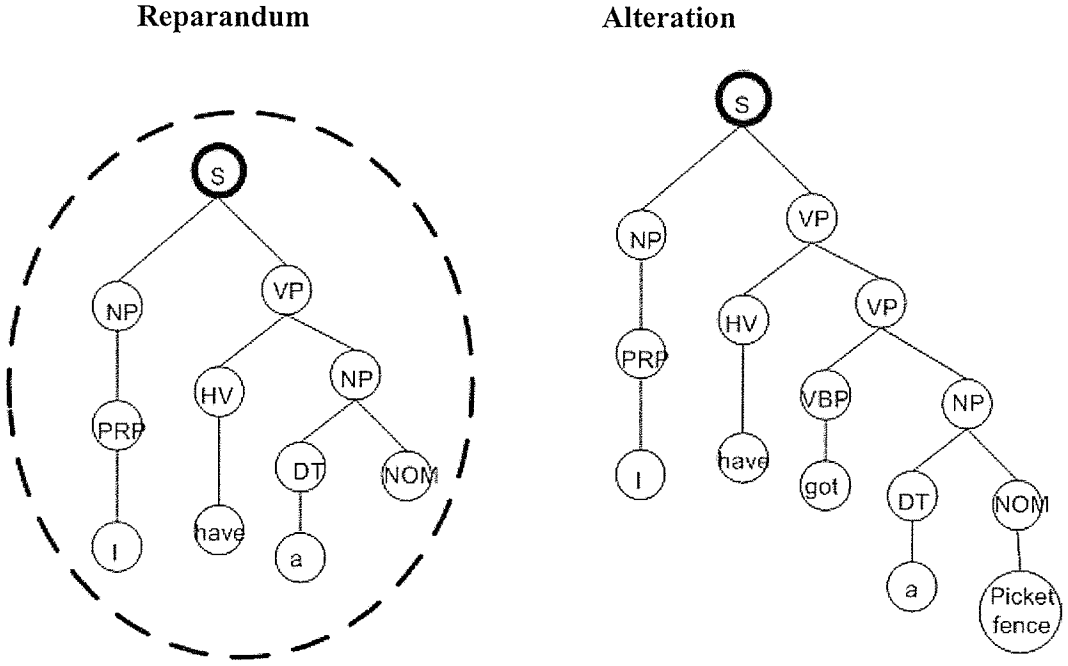


Figure 5-4: Structure representing the reparandum and its alteration

The graphs representing the reparandum and its alteration are then traversed and the following similarities are obtained;

Matching	---------	S
Matching	---------	NP
Matching	---------	PRP
Matching	---------	I
Matching	---------	VP
Matching	---------	HV
Matching	---------	have
No Match	---------	NP

There is a structural similarity between the corrected part and the initial part of the utterance, therefore the final completed structure will replace the initial part of the utterance.

5.1.2.2 Finding structural connection between the reparandum and the initial part of the utterance

In a spoken language utterance, the user starts a sentence, then repeats or modifies what he started to say. Therefore, Based on Levelt's third claim on self-repair production [27], the corrected part of the utterance (i.e. alteration) will be a continuation of the initial part of the utterance. In the utterance "I am still at the parallel with the base", the complete adjective phrase (ADJP) formed by the words "parallel with the base" can complete the incomplete verb phrase structure formed by the words "am still". The structural connectivity between the alteration and the initial part of the utterance helps to determine the complete sentence structure. Example 5-2 shows an utterance in which the structural connectivity between the alteration and the initial part of the utterance is useful to detect the span of the reparandum.

Example 5-2

I am still **<u>at the</u>** Reparandum IP **<u>parallel with the base</u>.** Alteration

The partial parse tree generated on parsing the entire sentence is shown in Figure 5-5. In Figure 5-5, the adjective phrase (ADJP) formed by "parallel with the base" has no structural similarity with its reparandum "at the" which is a preposition phrase (PP). The final corrected part of the utterance i.e. "parallel with the base" is a continuation of the initial part of the utterance i.e. "I am still". The structural connection between the alteration and the initial part of the utterance is useful to derive a complete sentence structure.

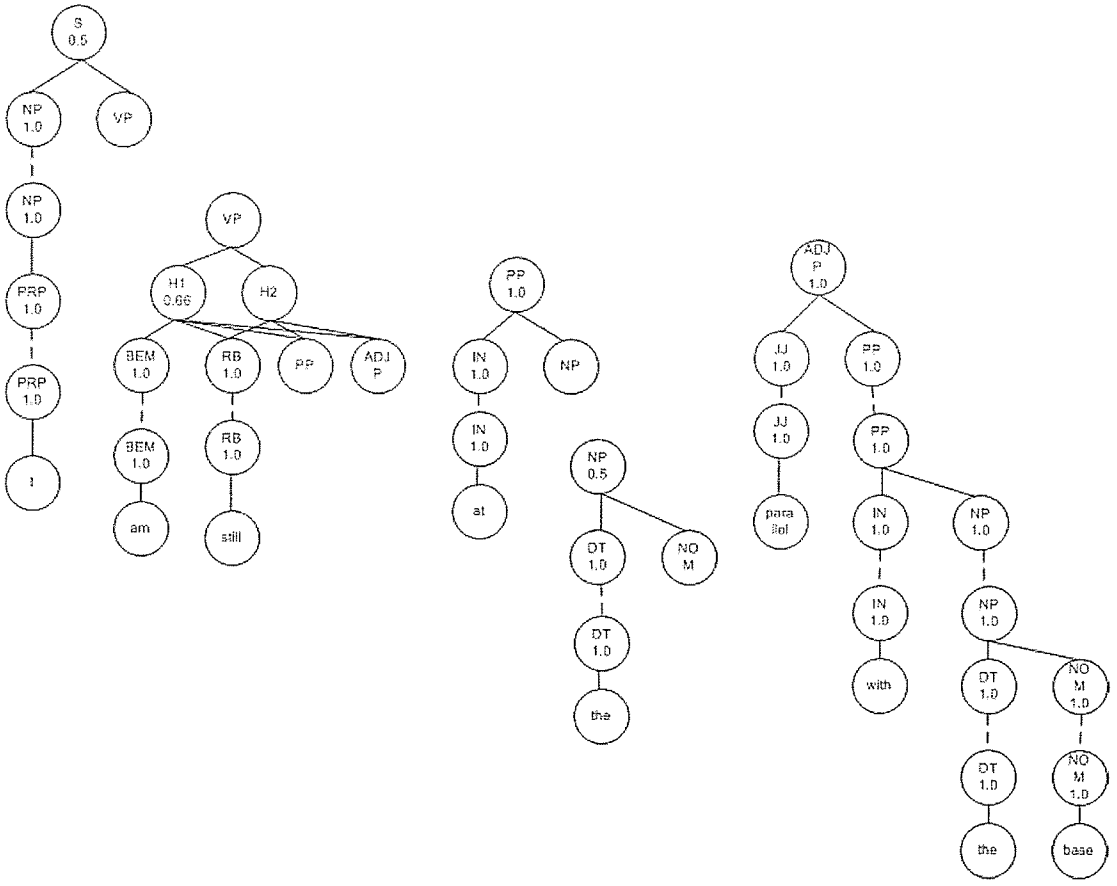


Figure 5-5: Partial parse tree for the utterance *"I am still at the parallel with the base".*

In Figure 5-5, the fully activated ADJP has to be bonded to an open node in the previous structure. Search is made in the structure formed by the category of the word "the" and word "at". Since, there is no matching open node, search continues in the previous structure formed by the category of the word "still". The word "still" is an adverb (RB) and it is bonded to open node of the VP structure. The VP structure, which is partially activated, requires a fully activated PP or an ADJP to become fully activated. Since, the user replaces his words "at the" which is a preposition phrase i.e. "PP" by an adjective phrase i.e. "ADJP [parallel with the base]", the final fully activated phrase will complete the structure formed by the initial part of the utterance "I am still".

5.2 Incremental Processing of Spoken Language

The incremental processing method performs correction of disfluent part as soon as an error[6] is detected and enough structural (syntactic) information has been gathered to derive a proper structure. The detection of interruption point in an utterance forms a major phase in a spoken language processing system. Even though, a repair is detected at an early stage by the parser, the post-processing method triggers the PSMA only after parsing the entire utterance. The presence of a repair also hinders the parsing process and processing of other repairs within the same utterance. The incremental processing of spoken language helps to deal with such situations by correcting the repair as soon as they are detected. Section 5.2.1 shows how the PSMA is triggered in an incremental fashion.

[6] The error denotes speech repairs.

5.2.1 Detection of the Interruption point and Triggering of the PSMA

Interruption point is the point, at which the user detects the need to repeat or correct what they have uttered. The interruption point is detected whenever there is a violation in the parsing process.

Whenever the parser reads a word it generates a syntactic structure based on the category of the word. After generating a mini-neural network, transfer of activation takes place within that network. When the root node of the generated network is fully activated the parser will bind its fully activated root node to a previous expecting node.

Another option is to generate further structure and later bind it to previous expecting node. When there is no binding to the previous expecting node at the word category level and at the phrase level (i.e. fully activated phrase) the decision about the presence of phrase structure breakage is made and the PSMA is triggered.

The HCP looks at different possible options while generating a parsed structure for the input sentence. For a spoken language, disfluents parts can be detected whenever there is a violation in the parsing process. Therefore the incremental processing method detects and removes the reparandum as soon as there is a violation in the parsing process.

The incremental processing of the utterance "I I am at the walled city" starts with generating syntactic structure for the first word "I". The structure of the first word "I" is represented in Figure 5-6.

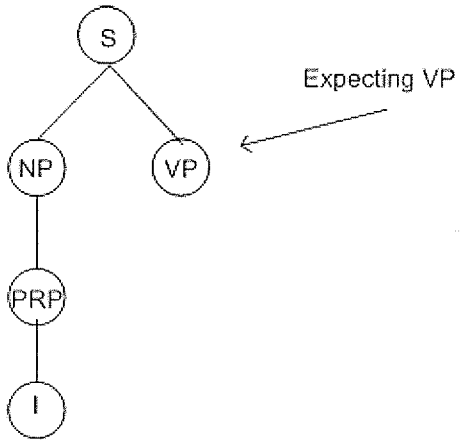


Figure 5-6: Partially parsed structure based on the first word "I".

Since, further structure cannot be generated based on the incomplete sentence structure (Refer Figure 5-6), the parser reads the next word "I" as input. The syntactic category based on the category of the word "I" i.e. pronoun (PRP) is selected and the network is created. The parser then checks for previous expecting node. In Figure 5-7, the fully activated PRP checks for previous expecting node.

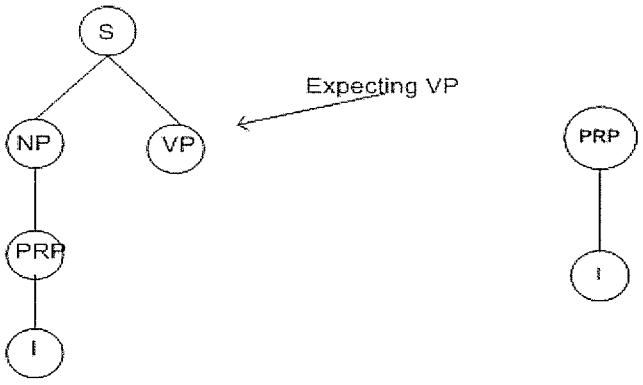


Figure 5-7: Partially parsed structure after parsing the utterance "I I".

The fully activated PRP node does not have any previous expecting node. Example 5-3, demonstrates the PRP as the first category on the right hand side of the grammar rule selected by the parser.

Example 5-3

1. NP ← PRP

The network is generated based on the grammar rule shown in Example 5-3. The fully activated NP in Figure 5-8 checks for the previous expecting node. Since, there is no expecting node the parser assumes the presence of interruption point and triggers the PSMA.

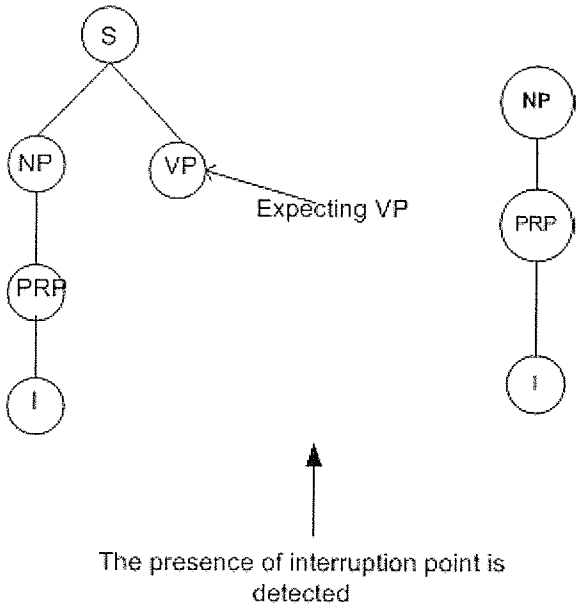


Figure 5-8: Detection of interruption point and the PSMA is triggered.

Figure 5-8 shows that the fully activated noun phrase has no previous expecting node and the PSMA is triggered. The PSMA then searches for the suitable matching structure. The search starts with the network formed by the first word before the interruption point. The first network that is searched will be the one formed based on the category of the first word. In Figure 5-9, the network within the dotted curve is the first network searched by the PSMA.

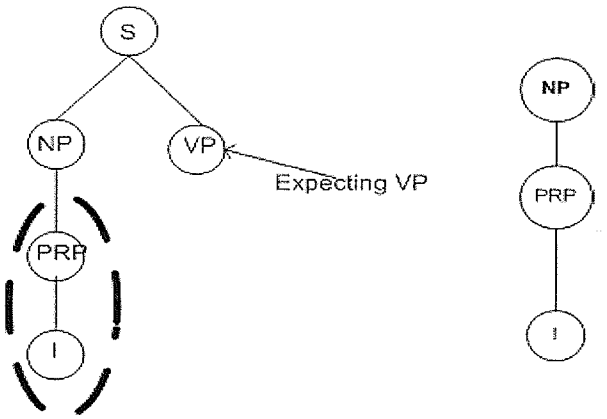


Figure 5-9: The first network that is searched by the PSMA.

The root node label (PRP) does not match with the noun phrase structure. The PSMA will then search in the next network generated based on the category of the word (Refer Figure 5-10).

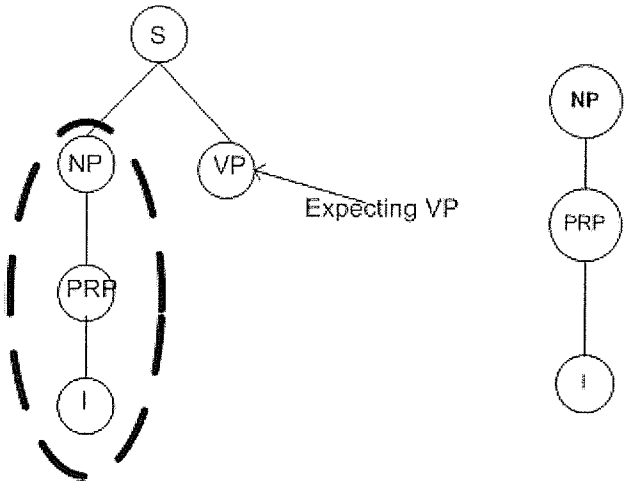


Figure 5-10: The noun phrase structure that is searched by the PSMA.

The root node symbol of the reparandum and its correction match and the PSMA then checks for similarity of the structure. The graphs are traversed top-down and similarity of the nodes at each level is determined.

Matching – NP

Matching – PRP

Matching – I

Since, there are similarities at the word category level i.e. PRP and at the word level i.e. "I", the structure formed by the corrected part of the utterance replaces the noun phrase structure detected by the PSMA.

When the PSMA does not detect the suitable matching structure, the parser continues its operation by generating further network. The parser builds a bigger structure by accepting further input words and later triggering the PSMA. In the utterance "I am I am going to hit the roundrocks", the interruption point will be detected by the parser after generating the NP for the third word "I". Since, the *reparadum* is an incomplete sentence structure formed by words "I" and "am" the PSMA will detect the matching structure i.e. reparandum only after parsing "I am going". The incomplete sentence structure created on parsing the initial parts of the sentence "I am" is shown in Figure 5-11.

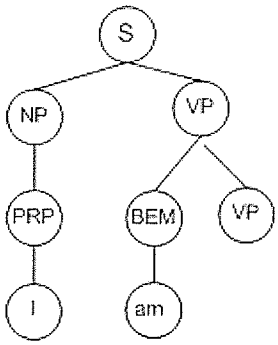


Figure 5-11: Incomplete sentence structure created on parsing "I am".

In Figure 5-11, the incomplete sentence structure requires a VP as the next input phrase. The parser reads the next word and generates syntactic structure. The NP formed by the next word (i.e. "I") cannot be bound to the previous expecting node. The presence of interruption point is detected as there is no binding to the previous expecting node at the

word category level (PRP) and at the phrase level (NP). After detecting the interruption point, the PSMA is triggered. The PSMA then searches for the suitable matching structure. A generation of syntactic structure based on the third word "I" and detection of the interruption point by the parser is shown in Figure 5-12.

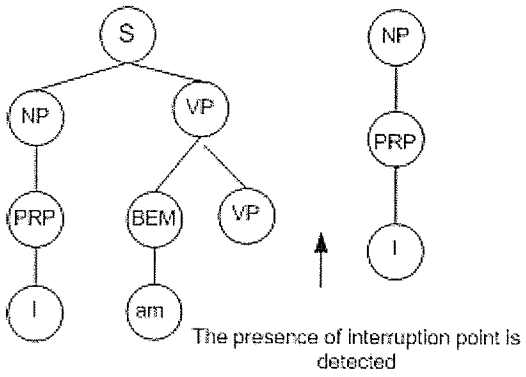


Figure 5-12: Detection of interruption point and triggering of the PSMA.

Figure 5-12 demonstrates that the NP structure formed for the word "I" cannot be bound to the bound to the previous expecting node. The interruption point detected is also shown in Figure 5-12. The PSMA triggered then searches for a suitable matching structure.

Figure 5-13 shows the three different networks, which represents three different steps followed by the PSMA to detect the suitable matching structure. The PSMA first checks the network created based on the word "am" and its category (i.e. BEM). In the second network, the PSMA checks the network based on the word category. The PSMA then selects the next higher sentence structure which includes the VP because it is a part of that structure. In the third step, the incomplete sentence structure is selected. Since there is no matching noun phrase "NP", the parser continues to parse the input words and later tries to find the suitable matching structure.

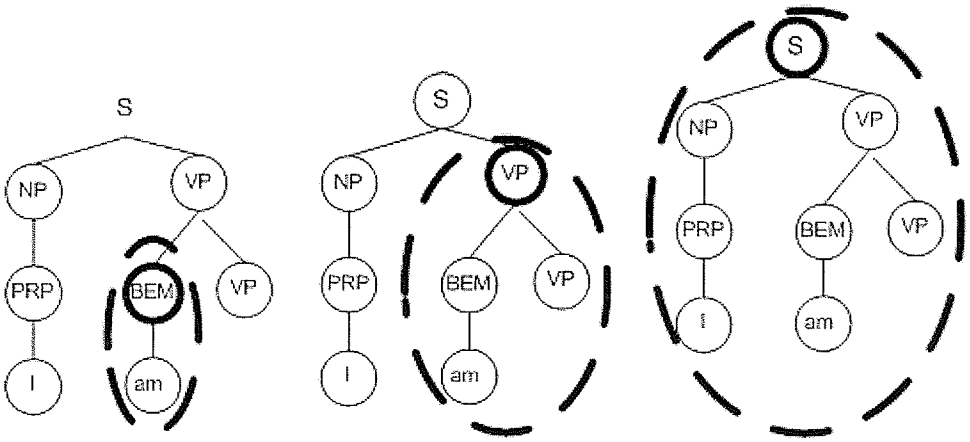


Figure 5-13: The PSMA checks the verb phrase structure for similarity.

A complete sentence structure is created by generating syntactic structure for the words "I am going". The root node then checks for the previous expecting node with symbol "S".

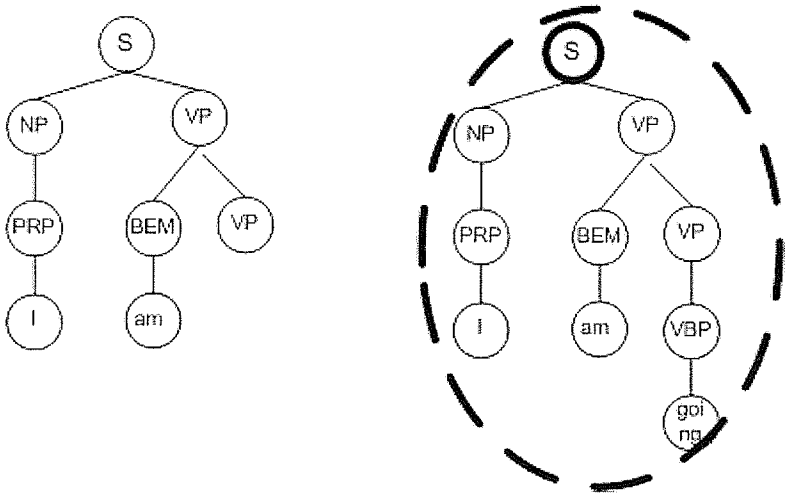


Figure 5-14: Triggering of the PSMA after parsing "I am going"

Since, there is no previous expecting node with symbol "S", the PSMA looks for previous matching sentence "S" structure. The steps followed by the PSMA are shown Figure 5-13. Finally, the PSMA detects the incomplete sentence structure formed by the

words "I" and "am" and the incomplete sentence structure is replaced by the complete sentence "S" structure.

5.3 Summary

In this chapter, the post-processing and the incremental processing method are used to detect and process disfluencies. In the post-processing method, the PSMA is triggered after parsing the utterance. Since, the presence of repairs hinders with the parsing process and processing of other repairs, the post-processing method cannot be used in utterance which has many repairs. In the incremental processing method, the PSMA is triggered as soon as a repair is detected. The incremental process handles utterances with many repair instances.

Chapter 6 Experimental Studies

Experimental studies for this thesis started with the selection of a suitable corpus. Section 6.1 gives a background on different corpuses available in spoken language followed by a selection of a suitable corpus for testing. The results obtained using the post-processing and the incremental processing methods are analyzed in Section 6.2.

6.1 Spoken Language Corpus

A Spoken language corpus contains transcripts of speech collected for research purposes in the field of speech and natural language processing. There are number of spoken language corpuses available. Some of the available corpuses are:

- The TRAINS Corpus
- The HCRC Map Task Corpus
- The Air Travel Information Services (ATIS) Corpus
- The Switchboard Telephone Speech Corpus
- TIMIT Corpus

Apart from the TIMIT Corpus [42], all other corpuses contain transcripts of natural speech, and the data was collected in a simulated environment. The simulated environment is based on the domain or the topic in which two speakers talk, either over a telephone or using a microphone. The data collected in a simulated environment can be used to solve problems with respect to the particular domain and with spoken language in general. In the next paragraph, I will give a brief overview of these corpuses.

The TRAINS project [22] was based on developing a planning assistant which uses a natural language interface. The system was designed to aid the user to construct and

monitor plans with respect to the assignment and the shipment of goods. TRAINS Dialog Corpus is a collection of task oriented dialogs between two speakers with one playing the role as a system. The corpus is collected to give a clear picture of the various phenomena that occur in such dialogs.

The HCRC Map Task Corpus [2] consists of transcripts of conversation between two participants (i.e. the instruction giver and the instruction follower). The instruction giver has a map with the route marked on it and the follower's map has no route marked on it (contains only places). The task of the instruction giver is to explain the route, so that the follower can regenerate the route in his map.

The ATIS spoken language system pilot corpus [19], consists of transcripts of speech from the ATIS (Air Travel Information System) domain. The corpus was collected in an environment, in which the user initially introduced the system. Then the user is given a travel planning scenario. The user queries the system based on the topic given. The user is then given 5 minutes to plan the details of the scenario and pen and paper to take notes. A Push to talk mechanism is used in the system and the speech data was collected only after the marked mouse button was depressed.

The SWITCHBOARD Corpus [14] contains transcripts of 2500 conversations. 500 speakers participated in the collection. The number of speakers involved in this experiment and the amount of data collected was large enough, which makes this corpus suitable for speaker identification and speech recognition experiments.

Finally, the TIMIT corpus [42] is a collection of read speech which is used for the development of automatic speech recognition system. The speakers who took part in the

creation of this corpus will read 10 phonetically rich sentences which are recorded and transcribed.

TRAINS, ATIS and the HCRC Map Task Corpuses can be categorized as task oriented corpuses. Even though the dialogs are restricted to the possible conversation that takes place in that domain, other features such as discourse and disfluencies are large enough to study the characteristic of features in speech and natural language research. The SWITCHBOARD is a large corpus which has transcripts of conversations on different topics and is used for different applications such as speaker identification, topic detection, speech recognition and acoustic phonetic studies.

There are other spoken language corpuses such as Santa Barbara Corpus Spoken American English [37], CALLHOME [5] and Monroe Corpus [21]. The Monroe corpus describes itself as a "complex and large corpus which gives the speakers scope for demonstrating a variety ways of managing initiative, participating in conversational games" [21]. Each dialogue of this corpus has duration of 20 min compared to 4.5min for TRAINS and 7 min for Maptask corpus.

In this thesis, I will use a task oriented corpus. The selected corpus will deal with all disfluent phenomena (speech repairs in specific) that occur in spontaneous speech. The selection is therefore limited to TRAINS, Maptask and ATIS corpuses. The TRAINS and Maptask corpuses are based on natural conversation between humans compared to ATIS which was developed in a more controlled environment. In this thesis, I will use the Maptask corpus to work on speech repairs.

6.2 Results and Analysis

In this section, results obtained using the post-processing methods and the incremental processing methods are analyzed.

6.2.1 Post-Processing of Partially Parsed Structure for a Sentence

In the Post-Processing method, the processing of disfluent utterance starts only after parsing the entire utterance. Three different experiments are conducted using the post-processing method. In the first method, 55 sentences are tested to see how well the system works. In the second experiment, 70 test sentences are used and only 28 sentences are corrected properly. As the number of rules used by the system increases, the parser accepts disfluent parts of the utterance as a correct phrase. Other problems faced in this experiment are also discussed. In Experiment 3, the fine tuning of grammar rules and the modification of the matching algorithm are done to deal with the problems of Experiment 2. Out of the 98 test sentences only 42 are corrected properly in Experiment 3.

6.2.1.1 Experiment 1

In experiment 1, sentences which have one repair instance are tested. This experiment is aimed towards testing the basic functionality of the parser. The number of rules used by the system is limited to cover only those required to generate structures for 55 test utterances. The matching algorithm is able to detect and correct repairs for 38 utterances. The remaining 17 sentences are not corrected properly due to the following problems

1. Use of recursive rules like

   a. VP ← V VP

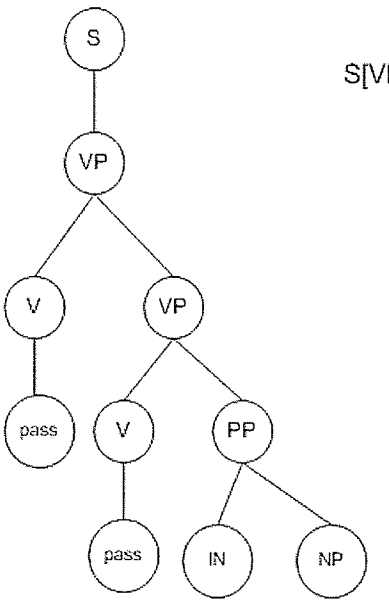
   b. ADJP ← JJ ADJP

The use of recursive rules helps to derive a structure for a sentence with disfluencies. Example 6-1 shows an utterance, which is recognized as a proper sentence due to the use of recursive rules.

Example 6-1

```
pass pass below the ghost town
```

On parsing the utterance in Example 6-1, a complete sentence structure is generated. Figure 6-1 shows the parse tree generated for the utterance "pass pass below the ghost town". In Figure 6-1, the use of recursive rule (i.e. `VP ← V VP`) facilitates the generation of a syntactic structure for the words "pass pass". The first word "pass" is detected as a verb (V) and the remaining parts of the utterence are detected as a verb phrase (VP). Bracketed notation, shown in Figure 6-1, is used to represent the parse tree generated.



Bracketed Notation

S[VP[V[Pass] VP[V[Pass] PP[IN[below] NP[the ghost town]]]]]

Figure 6-1: Parse tree for the utterance "pass pass below the ghost town."

2. There might not be enough similarities between the structures representing the reparandum and its alteration. The matching algorithm checks for similarities between the phrase structures. In Figure 6-2, the right hand side structure represents the corrected part of the utterance **"do a c so that it is like a backward c"** and the left-hand side structure represents the reparandum part of the utterance **"do the"**. The structure on the left-hand side and on the right-hand side of Figure 6-2 has the same root node "S". The categories responsible for the creation of the sentence structure (S) in the left and in the right structure of Figure 6-2 are different. Similarities between the structures are derived by traversing the structure top down. The following similarity information is derived.

   Matching – S (Root node)

   Matching detail between the nodes after traversing the edge marked "1".

   No Match – S (Leftmost category on the second level of the second structure)

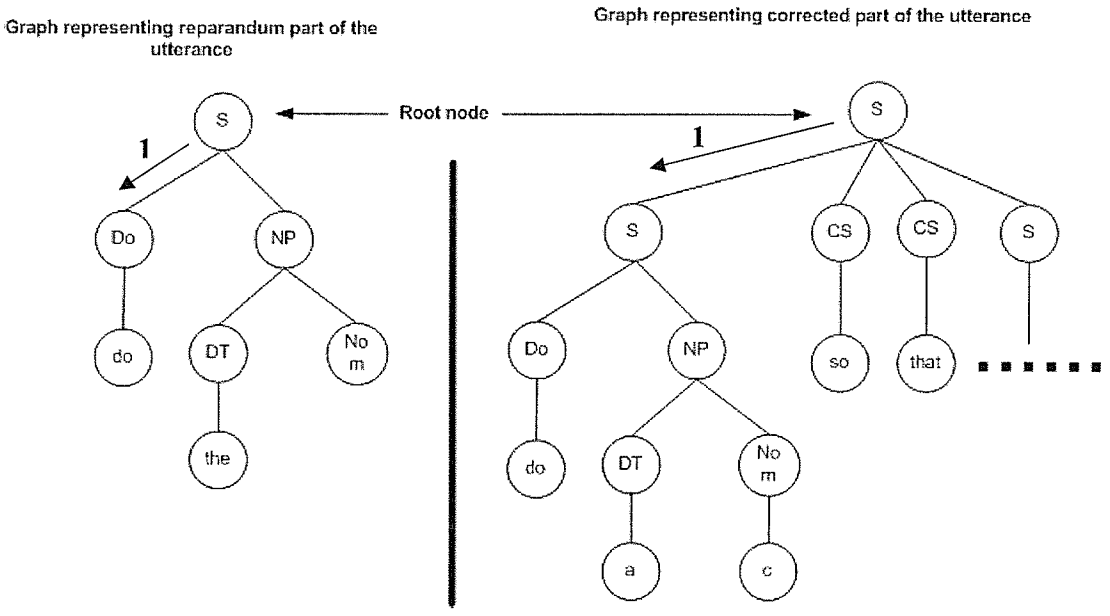


Figure 6-2: Graphs representing the reparandum and the corrected part of the utterance.

There are some similarities between the structures apart from the root node. The structure representing the reparandum and the corrected part of the utterance are traversed top-down. The left-most category in the second level of the corrected part of the utterance is a sentence node (S). In the reparandum, the left-hand side category is a do verb category i.e. "Do". Therefore, there is no match with respect to symbols in the nodes. Even though, the left-hand and the right-hand side structure represents the reparandum and the corrected part of the utterance respectively, the reparandum is not detected due to the lack of similarities between their structures.

6.2.1.2 Experiment 2

In this experiment, 71 sentences are tested and more grammar rules are added to accommodate more input structures. In this experiment, the following results are obtained.

The number of corrected sentence structures: 28

The number of incorrect structures: 43

The number of correct sentence structures derived in this experiment denotes the sentence in which the repairs are detected and corrected properly. Out of 43 incorrect structures, the system is able to detect the point were the interruption has occurred for 12 utterances. The presence of recursive rules is also responsible for the incorrect interpretation. In this experiment, 9 utterances have incorrect derivations due to the recursive rules. Remaining, 22 utterances have incorrect structures due to the following problems

- The detection of incorrect interruption point
- The presence of rules which can accept incorrect structures

In this thesis, the HCP is used to determine all possible derivations for a sentence. The detection of the interruption point after parsing an utterance is a complicated task. In an incremental parser, ambiguous situations have to be dealt. Interim derivation can be understood as a partially parsed structure followed by an interruption point. In this section (i.e. The Post-Processing of Partially Parsed Structure) the final completed phrase is treated as the corrected phrase or the alteration. The major problem at this stage is the coverage of words from the end of an utterance to form a phrase. There are cases, in which different coverage can give different interpretations. Example 6-2 shows how an incorrect interpretation can give the wrong results with respect to repair detection and correction. The utterance in Example 6-2 belongs to the repair category "fresh starts". In Example 6-2, the user starts with "I will have to" and then repeats the words "I will" and finally says the sentence "I will have to kindof change it slightly because I have taken it on the wrong side of the cactus thing". In Example 6-2, "Repair" represents the corrected part of the utterance or the alteration. The words "kind" and "of" are represented as a single word, so that the parser recognizes "kind of" as an adverb (RB). In Example 6-2, the structure created for the words "will I have to kindof change it slightly because I have taken it on the wrong side of the cactus thing" as a complete sentence structure (i.e. alteration). The PSMA then detects a sentence structure formed by "will have to I" as the reparandum. Since the structures don't match, the PSMA will not be able to detect the suitable matching structure.

Example 6-2

```
Right, I will have to I will I will have to kindof change it slightly
because I have taken it on the wrong side of the cactus thing

Reparandum = S[MOD[will]VP[HV[have]TO[to]NP[PRP[I]]]]
```

```
            Will have to I
Repair
=S[S[MOD[will]NP[PRP[I]]VP[VP[MOD[will]VP[HV[have]TO[to]VP[ADVP[RB[kind
of]]VBP[change]NP[PRP[it]]]]]ADVP[RB[slightly]]]]CS[because]S[NP[PRP[I]
]VP[VP[VP[HV[have]VP[VBP[taken]NP[PRP[it]]]]PP[IN[on]NP[DT[the]ADJP[JJ[
wrong]]NOM[NN[side]]]]]PP[IN[of]NP[DT[the]ADJP[JJ[cactus]]NOM[NN[thing]
]]]]]]
will I have to kindof change it slightly because I have taken it on the
wrong side of the cactus thing
Matching =S - root node
No Matching =S - leftmost node after traversing the repair top-down
```

The part of the utterance derived as the alteration (i.e. given as "Repair" in Example 6-2) is not a correct. The structure derived as the alteration is one possible interpretation. Choosing the correct interpretation will solve the problem of detecting the interruption point in the speech. After detecting the final completed phrase, the search for a matching structure is processed to detect the start of the reparandum. Due to the wrong interpretation of the alteration (represented as "repair" in Example 6-2); there will not be a correct interpretation of the reparandum.

Example 6-3 shows an utterance, in which a repair at the beginning of the utterance hinders the parsing process and the interpretation of the corrected part of the utterance.

Example 6-3

it and it crosses over the the mountain stream and then goes directly south underneath the dead tree

The utterance in Example 6-3 has two repair instances. The first repair instance belongs to the category of *fresh starts*. The first repair instance is in **Figure 6-3**. The first word "It" is followed by an interruption point. The second word "and" is used as an editing term and then the user restates the utterance. Due to the presence of the rule "NP ← NP CC NP" the disfluent part of the utterance "it and it" is considered as a proper structure.

it and it crosses over the the mountain stream and then goes directly south underneath the dead tree

Figure 6-3: The first repair instance.

The second repair instance belongs to the category of modification repair. Second repair instance is shown in Figure 6-4.

it and it crosses over **the the mountain stream** and then goes directly south underneath the dead tree

Figure 6-4: The second repair instance.

Under modification repair, the speaker does not restate what he started to say but replaces or repeats a word or phrase. In the second repair instance the word "the" is repeated. Since, the noun phrase rule "NP ← DT NOM" can take only one determiner "the" into its structure, there is a breakage in the phrase structure. There will be two structures generated, they are

1. An incomplete noun phrase based on the word "the"
2. A complete noun phrase based on the words "the mountain stream".

Figure 6-5, shows the incomplete and the complete noun phrase formed due to the word "the" and "the mountain stream" respectively. Since, the fully activated noun phrase cannot be bound to a previous expecting node, a breakage in phrase structure occurs.

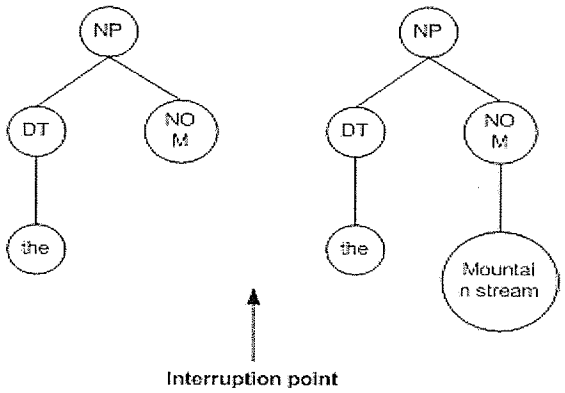


Figure 6-5: Partially parsed structure for "the the mountain stream".

On parsing the remaining parts of the utterance in Example 6-3 (i.e. "and then goes south directly under the dead tree"), a complete noun phrase will not be generated. Instead, there will be a structure formed due to the word "and" and a verb phrase formed due to the words "then goes south directly under the dead tree". Figure 6-6 shows the partial parse tree for "the the mountain stream and then goes directly south the dead tree". Thus, the final completed verb phrase cannot be taken as the corrected part or the alteration of the utterance.

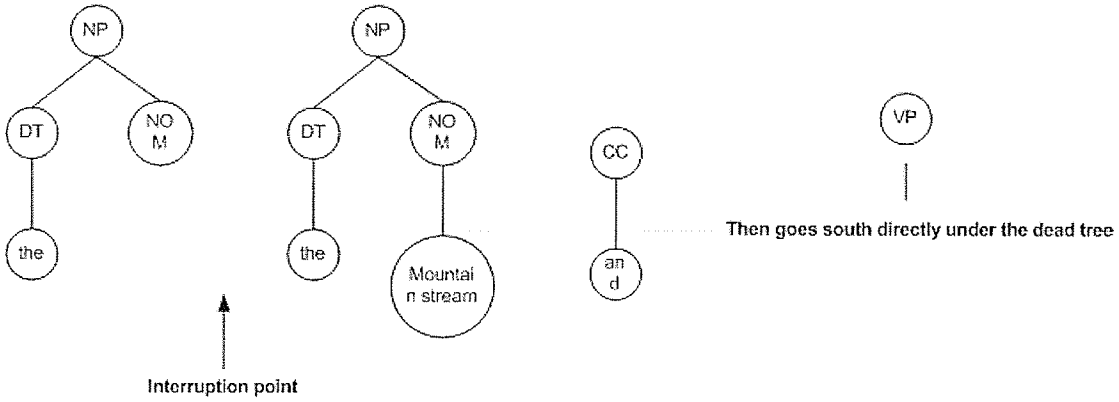


Figure 6-6: Partial parse tree for "the the mountain stream and then goes directly south the dead tree"

Solving repairs at an early stage will resolve problems of incorrect interpretation of the alteration.

6.2.1.3 Experiment 3

98 test sentences are used in this experiment. In Experiment 2 and Experiment 3, the graphs are traversed top-down and their similarity helps to derive the structure representing the reparandum. The post-processing method takes the phrase which covers maximum number of words from the end of the sentence as the corrected part of the utterance. Therefore, in the utterance shown in Example 6-4, "do a c so that it is like a backward c" is the corrected part of the utterance.

Example 6-4

Do the Do a c so that it is like a backward c

The partial parse tree derived for the utterance "do the do a c so that it is like a backward c" is shown in Figure 6-7. The sentence structure "S" derived for "do a c so that it is like a backward c" represents the corrected part of the utterance. The incomplete sentence structure (S) formed for the words "do the" has to be the reparandum, but, the

PSMA rejects it as their structure does not have any similarities. The graphs in Figure 6-7 are traversed top-down. The dotted lines in the graph represent the traversing path and the number represents the sequence in which the edges are traversed. The root nodes of both structures are similar. The matching root node is

Matching - S

Then, the first path in both graphs is traversed. In the corrected part of the utterance there is a node with the symbol "S". In the reparandum, there is a node with the symbol "DO". Therefore, we have

No Matching – S

The node with symbol "S", after traversing the first path, does not have a suitable matching node. Therefore, the detected structure on the left-hand side of Figure 6-7 cannot be taken as the reparandum.

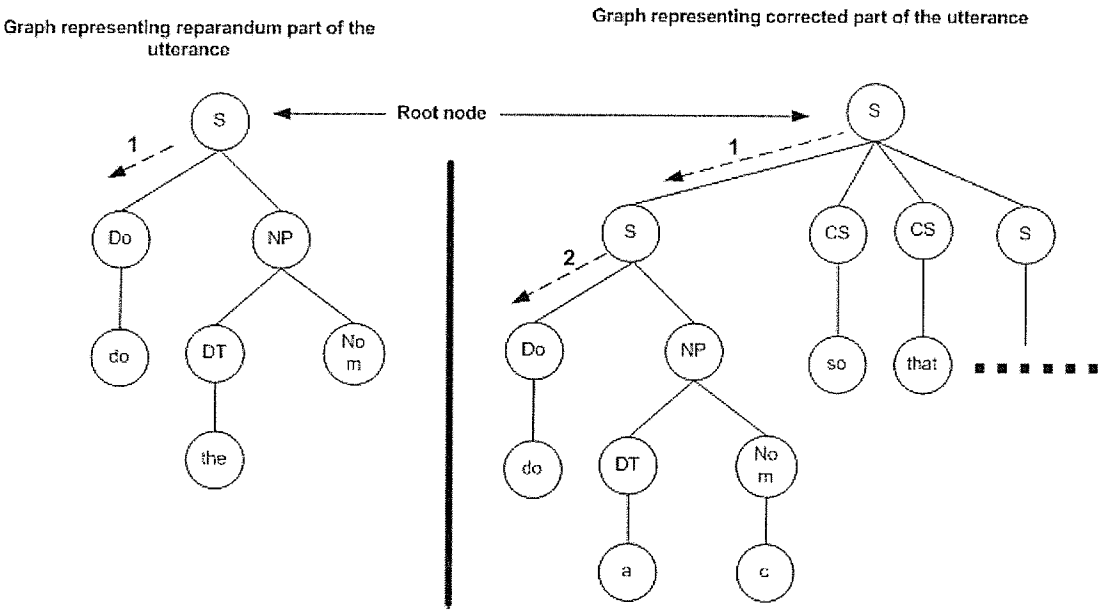


Figure 6-7: Partial parse tree for "do the do a c so that it is like a backward c"

In this experiment, the decision about the similarity between the reparandum and its correction is made by checking their root node symbol alone. In Figure 6-7, the left-hand side structure is taken as the reparandum, because they have the same root node label i.e. "S".

The input sentences used in this experiment were taken from transcripts of a dialogue corpus. The sentences that are collected were characterized by the presence of extra materials like affirmatives, interjections, and filled pauses at the beginning of the sentence. These extra materials occur as the speaker tries to respond to the other speaker's statement. Therefore, the extra materials are removed before giving these sentences to the system. After giving all the input sentences to the system, the following statistics are received.

No of correct sentences: 42

No of incorrect sentence structure: 56

6.2.2 Incremental Processing of Spoken Language

Experiment 1 show positive results with respect to the detection and the correction of repairs. Later experiments (i.e. Experiment 2 and Experiment 3) provided only 40% results. The post-processing method can provide a positive result if the utterances are small and have only one repair instance in it. Moreover, it is difficult to detect the exact structure representing the repair and its correction for a large utterance. In this section, an incremental way is followed to process repairs. Once, a repair is detected, the processing of speech repairs takes place immediately after sufficient information about its correction is gathered. Experiment 4 and Experiment 5 show the results obtained using the incremental processing method.

6.2.2.1 Experiment 4

Incremental processing of speech repairs is tested with spoken language sentences. In this experiment, 95 test sentences are used and the following statistics are derived.

No of correct output structures: 52

No of incorrect structure: 43

Incorrect structures are derived due to the following reasons,

1. As the number of rules used by the parser increases, the parser will have the capacity to generate a structure for disfluent parts in the sentence. 11 incorrect structures are derived due to the use of rules which can help the parser to accept disfluent structures. In Example 6-5, bracketed notation of the structure generated for the utterance "that is that is correct, and go below the diamond mine" is shown. In the utterance shown in Example 6-5, the user starts with "that is" and then repeats "that is". Even though, there is repetition, the rule "VP ← VBP S" can accept this disfluent structure. The output structure shown in Example 6-5 is one possible interpretation of the utterance (and not the correct one). It is tricky to get the correct interpretation using only syntactic information.

Example 6-5

```
thatPd is  ... thatPd is correct , and ... go below the diamond mine
OUTPUT:::
S[NP[PD[thatPd]]VP[VP[VBP[is]S[NP[PD[thatPd]]VP[VBP[is]ADJP[JJ[corre
ct]]]]]CC[and]VP[VBP[go]PP[IN[below]NP[DT[the]ADJP[JJ[diamond]]NOM[N
N[mine]]]]]]]
thatPd is thatPd is correct and go below the diamond mine
```

2. The use of recursive rules also has the ability to accept incorrect phrases. The number of incorrect structures derived due to the use of recursive rules is 2. In Example 6-6, the output sentence is the same as the input sentence. The use of the recursive rule "VP ← DO VP" causes the incorrect parts of the utterance "dont dont" to be accepted as the correct structure.

Example 6-6

```
dont ... dont cross the river at thatDt point
OUTPUT :::
VP[DO[dont]VP[DO[dont]VP[VBP[cross]NP[DT[the]NOM[NN[river]]]PP[IN
[at]NP[DT[thatDt]NOM[NN[point]]]]]]]
dont dont cross the river at thatDt point
```

Figure 6-8 shows the structure generated for the utterance "don't don't cross the river at that point" In Figure 6-8, the structure generated for the corrected part of the utterance is show within a dashed circle.

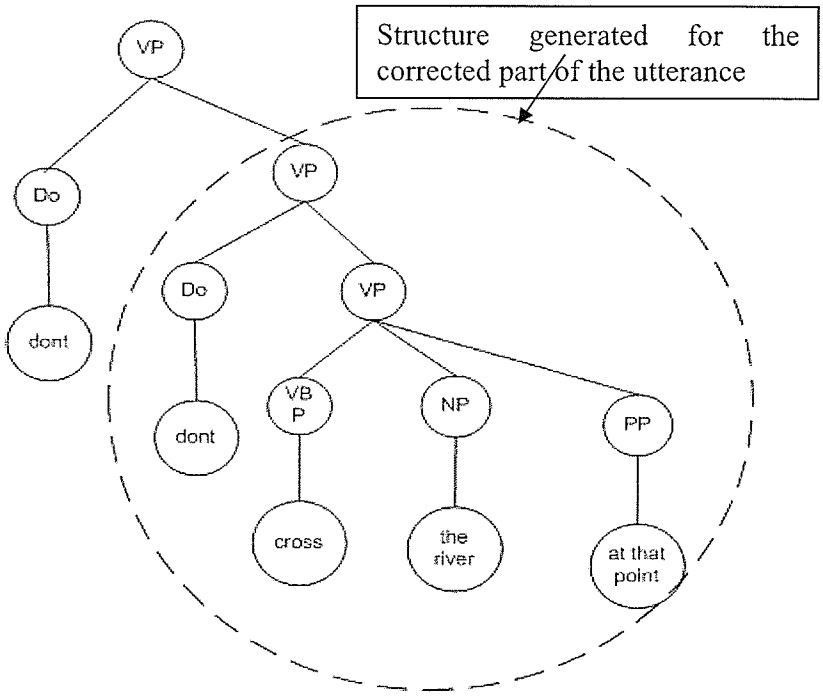


Figure 6-8: Partial parse tree for the utterance "don't don't cross the river at that point"

3. Due to the different possible interpretations of a phrase, choosing a possible syntactic structure and detection of the interruption point in speech becomes a tricky task. In Experiment 4, 14 utterances have wrong interpretations for the corrected part of the utterance (alteration). Example 6-7 gives an example for the incorrect interpretation of the corrected part of the utterance. In Example 6-7, "got I have got a great viewpoint above the tree just below the eastlake" as the corrected part of the utterance instead of "I have a great viewpoint above the tree just below the eastlake". After detecting the partially parsed structure for the corrected part of the utterance, the PSMA searches for a suitable matching structure. Since there is no matching structure, the second part of the output represents partially parsed structure for the word "have" and word "I".

Example 6-7

```
I have got , oh , I have got a great viewpoint ... eh ... ... above
the tree ... eh justQl below the eastlake
OUTPUT :::
VP[VBP[got]S[NP[PRP[I]]VP[HV[have]VBP[got]NP[DT[a]ADJP[JJ[great]]NOM
[NN[viewpoint]]]PP[IN[above]NP[DT[the]NOM[NN[tree]]]]PP[QL[justQl]IN
[below]NP[DT[the]NOM[NN[eastlake]]]]]]]
got I have got a great viewpoint above the tree justQl below the
eastlake
OUTPUT :::
HV[have]
have
OUTPUT :::
PRP[I]
I
```

As described in Example 6-5 and Example 6-6, it is quite tricky to detect the exact interruption point in the utterance. In this thesis, the corrected part is derived by taking the phrase, which covers the maximum number of words from the end of the utterance. Moreover, I haven't looked at different possible interpretations and choose the best one; which can give a better result with respect to choosing the correct interruption point. Even though the correct interruption point is detected for an utterance, there are cases, in which an incorrect matching structure (i.e. reparandum) is detected. There are 11 utterances which fall under this category. In Example 6-8, the PSMA is triggered after parsing the initial part of the utterance, "don't go don't go". The verb phrase formed for the third and the fourth word (i.e. "don't go") searches for a matching verb phrase. The verb phrase which is formed for the third and the fourth word is represented in Example 6-8 as follows.

```
Repair =VP[DO[dont]VP[VBP[go]]]
```

The search for the previous matching structure or the reparandum is performed by looking for a matching verb phrase structure. The verb phrase which is formed for the second word i.e. "go" is selected as the matching phrase. The verb phrase which is formed based on the word "go" is represented in Example 6-8 as follows.

```
Reparandum = VP[VBP[go]]
```

Since, the root node of the structures is the same i.e. "VP", the overlapping of parse trees takes place. The output of the integrated system is represented in Example 6-8 as,

```
OUTPUT :::
VP[DO[dont]VP[DO[dont]VP[VBP[go]ADJP[JJ[any]JJR[further]CS[than]
NP[DT[the]NOM[NN[desert]]]]]]]
dont dont go any further than the desert
```

Example 6-8:

```
dont go dont go any further than the desert

Reparandum = VP[VBP[go]]
Matching Root = VP
Repair =VP[DO[dont]VP[VBP[go]]]
Matching =VP
No Matching =DO
OUTPUT ::: Overlapping Parse Trees :::VP[DO[dont]VP[VBP[go]]]
dont go
OUTPUT :::
VP[DO[dont]VP[DO[dont]VP[VBP[go]ADJP[JJ[any]JJR[further]CS[than]NP[D
T[the]NOM[NN[desert]]]]]]]
dont dont go any further than the desert
```

5. When the corrected part of the user's utterance does not have the same structure as the reparandum, the PSMA will not be able to detect the reparandum. In Figure 6-9, the incomplete preposition phrase which is formed by the words "at the" is followed by an adjective phrase (ADJP) which is formed for "parallel with the base". The PP is followed by a complete ADJP. Since, there is no match with respect to the root node category; the PSMA fails to find a suitable match. The output of the PSMA is given in Example 6-9.

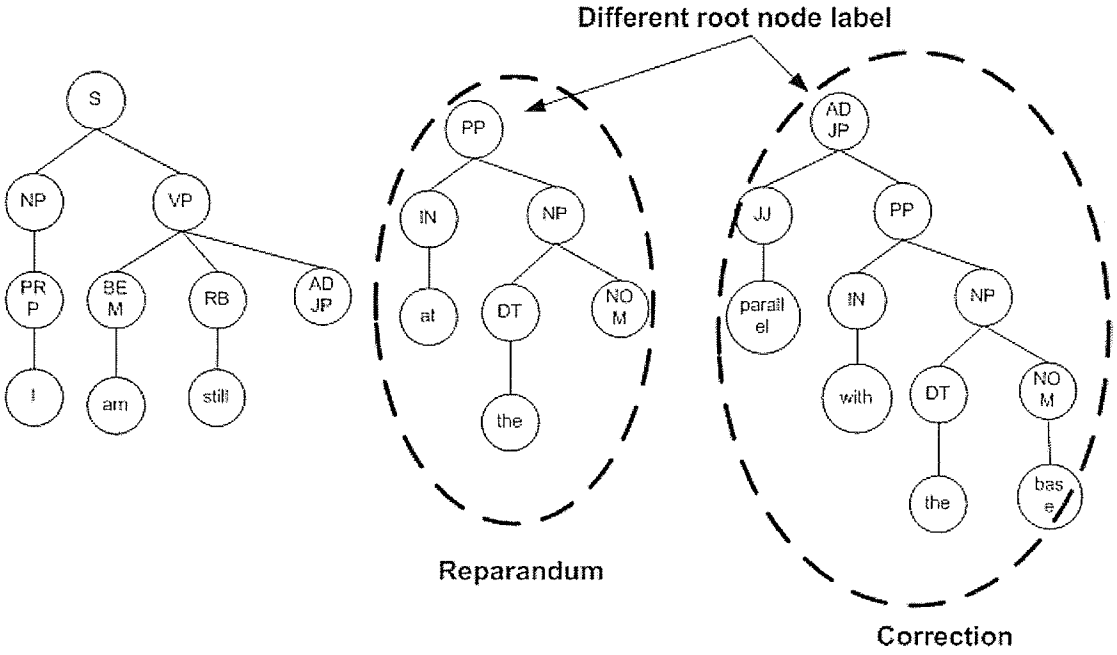


Figure 6-9: Partial parse tree for "I am still at the parallel with the base"

Example 6-9 gives the bracketed notation of the partial parse tree shown in Figure 6-9. In Example 6-9 only complete structures are represented as output. The complete adjective phrase (ADJP) which is formed by "parallel with the base" is first represented as follows.

```
OUTPUT :::
ADJP[JJ[parallel]PP[IN[with]NP[DT[the]NOM[NN[base]]]]]
parallel with the base
```

Since, all other phrases are incomplete, only complete structures formed by the words and their categories are printed.

Example 6-9

```
I am still ... at the ... ... parallel with the base
OUTPUT :::
ADJP[JJ[parallel]PP[IN[with]NP[DT[the]NOM[NN[base]]]]]
parallel with the base
OUTPUT :::
DT[the]
the
OUTPUT :::
IN[at]
at
OUTPUT :::
RB[still]
still
OUTPUT :::
BEM[am]
am
OUTPUT :::
PRP[I]
I
```

6.2.2.2 Experiment 5

In this Experiment, 5 different test conditions are hypothesized. At least one of the conditions has to be satisfied before detecting the structure representing the reparandum. The repetition of words is detected and removed as the words are read from left to right.

Condition 1

The repetitions of words are deleted before parsing the utterance. As the word is read by the parser, the parser checks whether the current word is a repetition of the previous word. If there is a repetition, then the current word is deleted.

Condition 2

Even though the interruption point is correctly detected by the system, the detection and the removal of the correct structure representing the reparandum is necessary to derive a sentence structure. In Experiment 4, the structure representing the reparandum is detected by taking the structure which has the same root node label as its correction. In the utterance "don't go don't go below the wooden pole", the verb phrase formed by the first two words represents the reparandum and the next two word "don't go" forms the alteration. In Experiment 4 the PSMA searches for the previous matching structure by checking only the root node label. The verb phrase formed for the second word "go" is selected as the reparandum. Figure 6-10, shows the partially parsed structure for "don't go don't go" and the verb phrase selected as the reparandum.

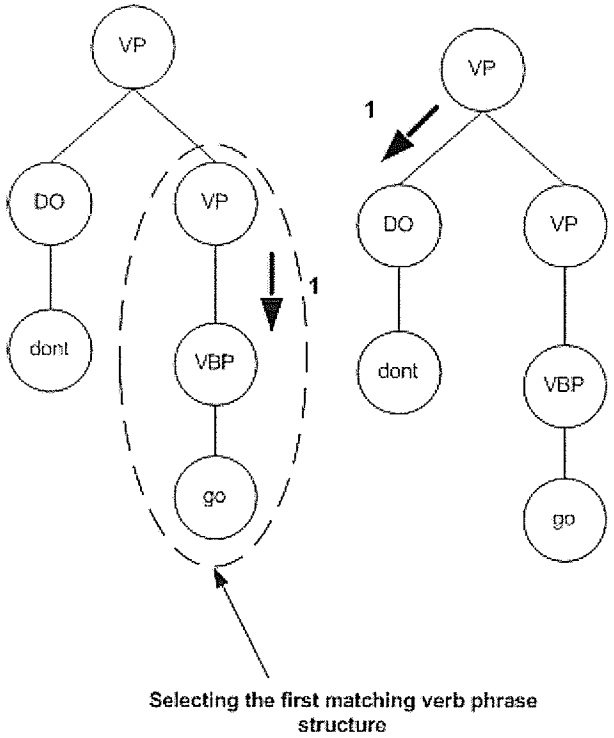


Figure 6-10: Partial parse tree for "don't go don't go"

Looking only at the root node label resulted in the detection of an incorrect structure. In this experiment, the detection of the reparandum is accomplished by looking at the structural similarity of the reparandum and its corrections. Traversing the reparandum and its correction shown in Figure 6-10 will give the following similarity details.

```
Matching Root = VP
No Matching = DO
```

Since, there is not any similarity the PSMA will look for next matching structure. The verb phrase formed for the first two words "don't go" is selected as the reparandum. The detected verb phrase structure formed for the first two words is shown in Figure 6-11. The sequence, in which the structures are traversed, to derive similarity information, is shown in Figure 6-11. The graphs are similar because they have,

Matching - VP

Matching - DO

Matching - dont

Matching - VP

Matching - VBP

Matching- go

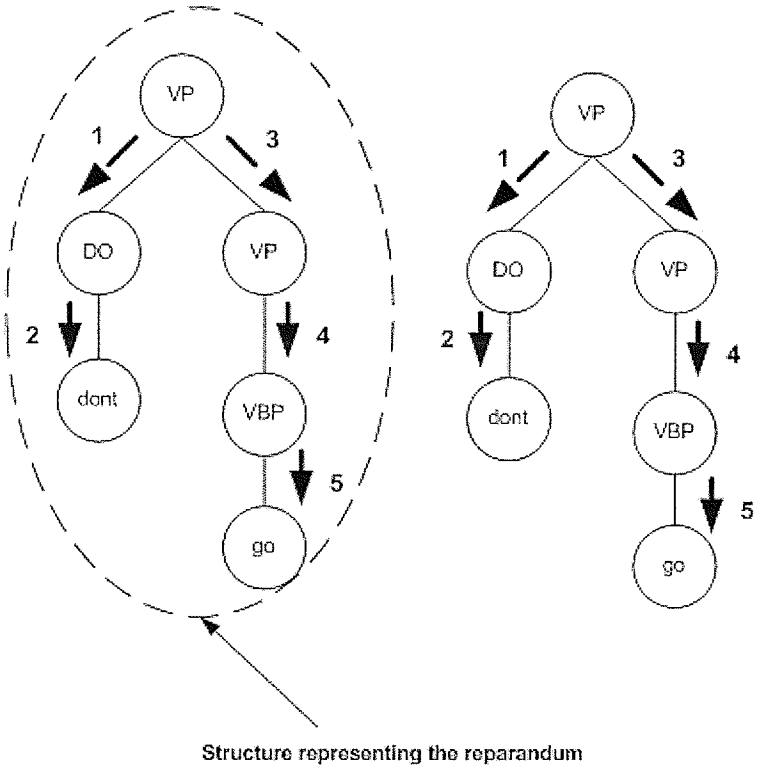


Figure 6-11: Graph traversal to represent similarity

Condition 3

When Condition 2 did not satisfy then this condition is checked. In condition 3, if the partially activated structure (representing the reprandum) is followed by a fully activated structure (representing its correction) and both structures have the same root node label, then they are represented as similar structures. When a partially activated structure is followed by a fully activated structure, the structure representing the reparandum is replaced by its correction. In Example 6-10, NP has no structural similarity but the partially activated NP is followed by a fully activated NP. Overlapping of the NP structure will then takes place.

Example 6-10

```
Reparandum = NP[PPG[your]]—partially activated NP
```

```
Correction =NP[DT[the]NOM[NN[left]]] - fully activated NP
```

Condition 4

Whenever a user reframes their utterance, they modify the utterance by either adding extra words to the initial parts of the utterance. The structure of the reparandum and its correction need not be the same. However there will be some similarities between the structure representing the reparandum and its correction. In Figure 6-12, the reparandum "below the" which is a preposition phrase is replaced by a verb phrase i.e. VP [go below the wooden pole]. The PSMA will fail to find the matching structure with the root node label VP.

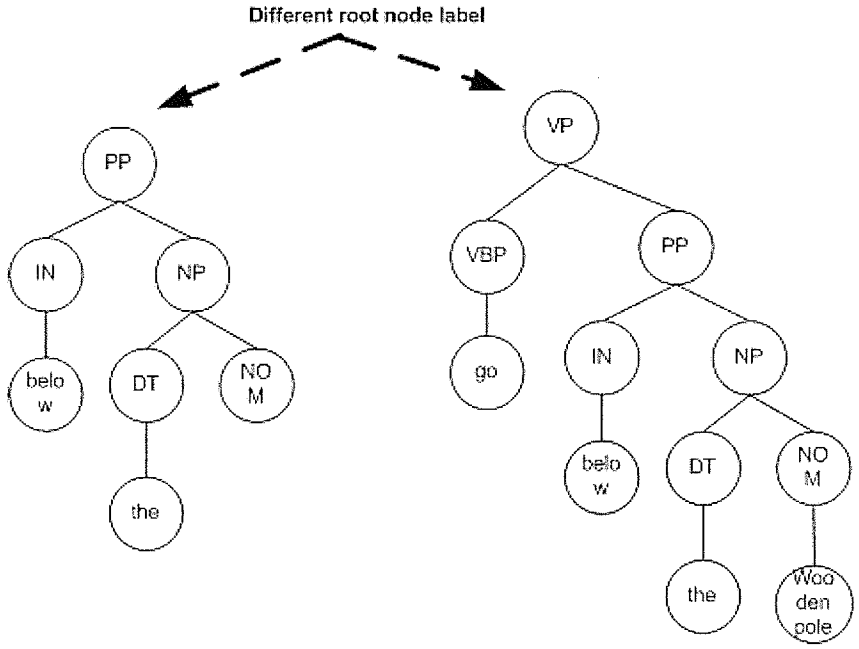


Figure 6-12: No matching verb phrase structure

Since, the reparandum is repeated by adding an extra word "go", it is difficult to detect the exact structure representing the reparandum. If the structure representing the reparandum is not detected then Condition 4 is checked, where the structure representing

the reparandum can be within the structure of its correction. In Figure 6-12, the correction is formed by a verb phrase structure. Since, there is no structural similarity between the reparandum and its correction, the next phase is to check whether the preposition phrase within the verb phrase (i.e. structure of the alteration) has similarities with the initial part of the part of the utterance. If there are similarities, then overlapping of partially parsed structures takes place. Figure 6-13 shows the similarity between the preposition phrases. The sequence, in which the network is traversed, is also in Figure 6-13.

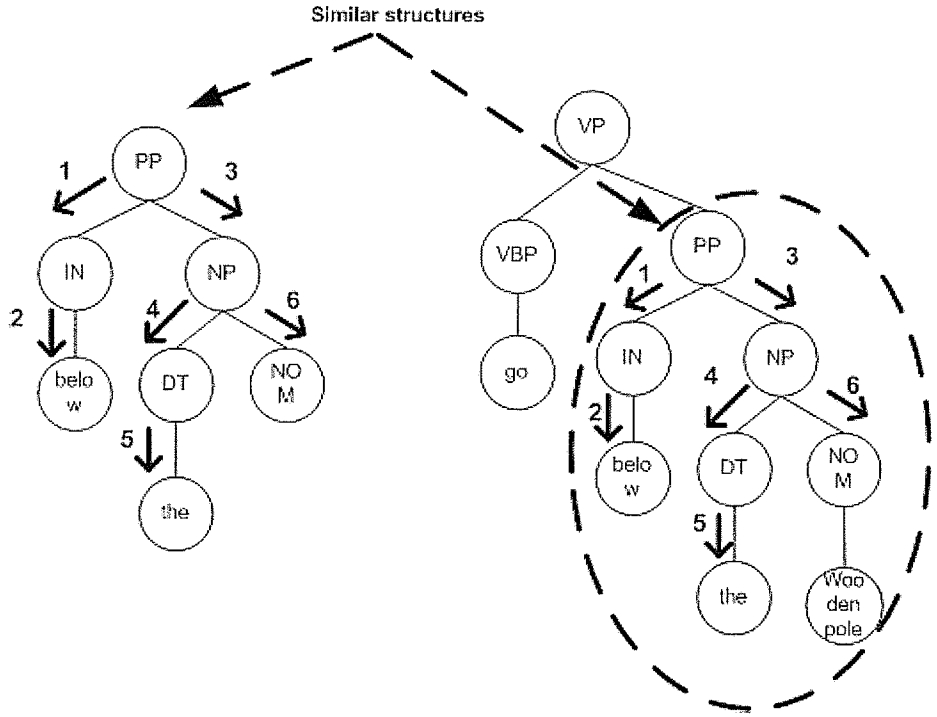


Figure 6-13: Similar preposition phrases

Condition 5

In the incremental method, the repairs are corrected whenever an error is detected. Sometimes it is difficult to detect the presence of a repair without processing the entire sentence. After parsing the sentence, the structure of the sentence is derived from the end of the sentence. When the structure derived does not cover the entire sentence, the PSMA is triggered at the point where the repair is detected. In Example

6-11, a few words like "far", "way", "just" and "up" have tags added to them. These words are categorized as ambiguous words that can have more than different syntactic categories depending on the context. In this sentence, the word "far" and "way" are used as qualifiers. Therefore, these words have "QL" tags added to the end of the word. In Example 6-11, the presence of a repair is not detected after parsing the word "wayQL". After parsing the entire utterance, the parser is able to derive a preposition phrase for the part of the utterance "wayQl over on the righthand side about justQl upRp from the middle of the map". Then the PSMA searches for the previous matching preposition phrase. An Incomplete preposition phrase formed by the third word "farQL" is taken as the reparandum. After overlapping the partially parsed structure, the complete sentence structure is formed for "it is wayQl over on the righthand side about justQl upRp from the middle of the map".

Example 6-11

```
it is farQl ... wayQl over on the righthand side about justQl upRp from
the middle of the map
Reparandum = PP[QL[farQl]]
Repair =3
Matching =PP
Matching =QL
No Matching =wayQl
OUTPUT ::: Overlapping Parse Trees
:::PP[QL[wayQl]RP[over]PP[IN[on]NP[NP[NP[DT[the]ADJP[JJ[righthand]]NOM[
NN[side]]]PP[RP[about]PP[QL[justQl]RP[upRp]PP[IN[from]NP[DT[the]NOM[NN[
middle]]]]]]]PP[IN[of]NP[DT[the]NOM[NN[map]]]]]]]
wayQl over on the righthand side about justQl upRp from the middle of
the map
```

```
OUTPUT :::
S[NP[PRP[it]]VP[VBP[is]PP[QL[wayQl]RP[over]PP[IN[on]NP[NP[NP[DT[the]ADJ
P[JJ[righthand]]NOM[NN[side]]]PP[RP[about]PP[QL[justQl]RP[upRp]PP[IN[fr
om]NP[DT[the]NOM[NN[middle]]]]]]]PP[IN[of]NP[DT[the]NOM[NN[map]]]]]]]]]
it is wayQl over on the righthand side about justQl upRp from the
middle of the map
```

After all the test conditions are integrated into the matching algorithm, the following statistics are obtained.

No of input sentence: 95

No of correct output structures: 60

No of incorrect structure: 35

In this experiment, the graph matching algorithm is fine-tuned to detect the corrected part of the utterance that have to be removed. Even though, there are 35 incorrect structures (i.e. input utterance for which the repairs are incorrectly detected), there are different categories of repairs under which these incorrect structures fall. Only 9 out of 35 incorrect structures are derived due to the detection of a wrong reparandum start. Only 9 incorrect structures means that the PSMA can detect the accurate structure that represents the reparandum start. 22 out of 35 incorrect structures are due to rules which can generate proper structure for the disfluent parts of an utterance. 4 incorrect structures are due to improper detection of the interruption point. The test data, the matching process to detect the reparandum, and the final structure derived are given in Appendix.

6.2.3 Discussion and Evaluation

Detection of the interruption point in speech and the correction of the speech repair forms a major phase in any spoken language processing system. In this thesis, the HCP is integrated with the PSMA to detect and correct repairs in speech. The PSMA detects the span of the reparandum and performs the repair correction. Two different processing methods have been implemented in this thesis. The post-processing method is found to be useful for utterances with one or two repair instances. The incremental processing method provides more promising results for utterances with many repair instances. The incremental processing method detects and corrects repairs in speech as soon as they are recognized. Precision and recall are the two different metrics to evaluate the speech repair detection and the correction algorithm [10], [15], [33].

The value of precision is calculated using the formula,

Precision is calculated using the formula = Tp /(Tp + Fp), Where

True positive (Tp) Total number of repairs that are detected as repairs.

False positive (Fp) Total number of false repairs that are wrongly detected as repairs.

Recall is calculated using the formula = Tp/(Tp + Fn), where

False Negative (Fn) Total number of repairs that are not detected as repairs.

The value of the precision and recall are calculated based on the mistakes made by the detection and the correction algorithm. Work of Bear et al., [3], Core [10], Heeman and Allen [15], Nakatani and Hirschberg [33] use speech transcripts which contains both correct and incorrect utterances. They measure the value of precision with respect to the false positives. In this thesis, all utterances selected from the corpus will have at least one disfluent part in it. Therefore, the value of precision is not calculated. Even though the

post-processing method is used to process repairs in speech, the method did not prove to be useful for utterances with many repair instances[7] in it. The results from the post-processing method are not compared with other works on speech repair detection and correction [10], [12], [15] and [33]. The incremental processing method is able to detect 89 out of 120 repairs with a recall rate of 73.5%. The PSMA is able to correct 80 out of the 89 repairs detected.

Core [10], use separate speech repair detection and correction algorithm. The parser performs speech repair detection after every word. Moreover, the detection algorithm also determines the type of the repair (*fresh starts or modification repair*). The reparandum is detected by performing a word matching process between a possible reparandum and its correction. The speech repair detection algorithm is able to detect 211 repairs out of 495 with a recall rate of 42.63%. Out of the 211 repairs, the correct reparandum start is detected for 165 utterances. The correction algorithm is modified to include POS[8] matching. The system is tested with a different set of the TRAINS dialog. The system is able to detect 237 out of the 521 repairs and correct 183 of the detected repairs[9]. Using only word match information, 187 repairs are corrected properly.

In the second part of Core's work, he proves that the parser can help to identify repairs in speech. In the second experiment, the output of Heeman and Allen's [16] pre-parser speech repair identifier[10] is given as input the parser. The output of the pre-parser

---

[7] Repair instance refer to the points where disfluency occur (interruption point in speech repair).

[8] POS represent part of speech tag or word categories.

[9] In this experiment, both word and pos similarity is used to detect the start of the reparandum.

[10] Pre-parser identifier is a classification given to the speech repair identification system, where the repairs are detected and corrected at the speech recognition phase.

identifier represents 100 possible hypotheses. The hypothesis is based on possible speech repairs, part of speech for the words. Heeman and Allen's system will select the best out of the possible 100 cases. Core's parser will process each of the hypotheses and select the best one based on the parser's coverage on the maximum number of words. Out of the three experiments performed based on finding the best hypothesis, I mention only the result of Experiment 3. Experiment 1 and Experiment 2 faced with a major problem of grammar coverage. In Experiment 3, a small amount of test data is used and with the goal to receive a 100% recall rate. Out 127 repairs, 90 repairs are correctly guessed in Heeman and Allen's method. Core's parser detects 98 repairs with a recall rate of 77.17%.

The second part of Core's [10] work showed that the parser can help in speech repair detection, which is used as a motivation in my current work on speech repair detection. Due to the presence of disfluency in speech, a breakage in the phrase structure[11] occurs. The breakage is used as the clue to trigger the PSMA. The PSMA checks similarity in phrase structure and detects the reparandum if there is a repair. Using this method, repairs are correctly detected in 60 out of 95 input utterances.

Heeman and Allen [15] have worked on the speech repair detection and the correction algorithm as part of the speech recognition system. The speech repair detection module is based on finding the probabilistic distribution between the words and their categories (POS tags).Since, the distribution for the words and its tags are different with

[11] Incremental parser generates syntactic structure as each word is recognized. From the second word to the last word in an utterance, the parser (HCP) generates structures and tries to bind to the previous expecting node. If there is no expecting node, then this situation is taken as the phrase structure breakage. This situation need not sufficiently depict the presence of repair. The PSMA will check for the repair and perform correction if an error has occurred.

respect to speech repairs, speech repairs are detected based on the difference in the probability distribution. Apart from detecting the interruption point of speech, the repair detection module also detects the type of the repair i.e. fresh starts, modification repair or abridged repair. The speech repair correction module works by finding the reparandum onset that best predicts the onset of its alteration. Other sources of information that help in repair detection are speech repair correction, intonational phrasing, and pauses. Using this model Heeman and Allen are able to detect and correct repairs with a recall rate of 65.85% and with an error rate of 56.88%.

Bear et al., [3] have analyzed the use of different knowledge sources such as the pattern matching of words, local and global syntax and semantics, acoustic cue to detect repairs in speech. Their pattern matching module looked at similar words separated by one or two intervening words as well as simple syntactic similarity. Out of 406 repairs, 309 are detected and 177 of the detected repairs are corrected properly. Apart from the pattern matching method, they show that the parser can detect the presence of a repair. A subset of 335 sentences with 179 containing repairs and 176 false positives are used as input to the parser. If a parser generates proper structure for a sentence, then the sentence is classified as false positive. When the parser fails to generate a proper structure then the pattern matcher is used to detect the exact repair. Using syntactic and semantic information, 85% of the repair and 80% of the false positive are detected.

Dowding et al., [12] in their work on spoken language use a parser to generate syntactic structure for the input. When the parser fails, then the correction algorithm based on pattern matching technique is used. Out of 26 repairs used as test data, the system is able to detect 11 repairs and correct 8 out of the 11 repairs.

The work of Core [10], Heeman and Allen [15], Bear et al., [3] and Dowding et al., [12] have described different knowledge sources used to detect and correct repairs. Comparing with these works on spoken language processing, the results of the incremental processing method are more promising. The following table summarizes the recall rates obtained using the work of Core [10], Heeman and Allen [15], Bear et al. [3] and Dowding et al. [12].

	Incremental Method (HCP+PSMA)	**Core's Work**	**Heeman and Allen**	**Bear et al.**	**Dowding et al.**
Repair Detection	73.5%	42.63%	76.7%	76%	42%
Repair Correction	66.66%	32.93%	65.85%	44%	31%

Table 6-1: Summarizing recall rates of different methods.

The incremental processing method detected 89 out of 121 repairs with a recall rate of 73.5%. The PSMA is able to correct 80 out of 89 repairs. In this work, only syntactic information is used to detect and correct repairs in speech. The phrase structure breakage is used as vital information to detect repairs in speech which is not used in other methods [10], [15], [3] and [12]. Moreover, due to the use of incremental parser, the phrase structure breakage is detected as words are parsed incrementally and the PSMA confirms the presence of a repair by looking at the similarity between the partially parsed structures. The PSMA looks at the similarity between the partially parsed structures. The phrase level and the word category level are taken into consideration to correct repairs. Phrase structure matching is an important knowledge source, which is not considered in other methods [10], [15], [3] and [12].

Chapter 7 Conclusion and Future Directions

In this thesis, a new syntactic way to process spoken language sentences with repairs is discussed. The enhancement of the existing HCP (Hybrid Connectionist Parser) forms a major contribution in this thesis. The incremental nature of the HCP helps in generating syntactic structures for words as soon as they are recognized. Temporary bindings are incorporated into the HCP to represent at all possible derivations in parallel. On parsing written text, all possible parse tree will be derived based on the given grammar rules.

Spoken language sentence are characterized by the presence of disfluencies such as "ums", "uhs", pauses, interjections, repetitions, and corrections. Explicit repairs such as "ums", "uhs", pauses and interjections are removed as the parser start to generate structures for words. If a word given to the parser is a pause or "um" or "uh" then the word is skipped. Spoken language processing forms the second major part of this thesis. To process sentences with repairs, the PSMA is developed. The PSMA deals with labeled graphs to detect the structure which has to be removed. The HCP is integrated with the PSMA to detect and correct repairs using only syntactic information of a sentence. All possible structure generated by the incremental parser (HCP) is used by the PSMA to find the suitable structure which is repeated or corrected. There are two stages in the spoken language processing module. In the first stage, the repair present in the sentence is detected. In the next stage, the PSMA uses this information to detect the structure which is repeated or modified.

To test the efficiency of the system, the post-processing and the incremental processing methods have been used. In the post-processing method, the PSMA is triggered only after the parser generates partially parsed structures for the words in the

sentence. 3 different experiments were conducted using the post-processing method. The first experiment is conducted using only selected sentences with only one or two repair instance. Repairs in speech are detected and corrected for 38 sentences from a total of 55 sentences. Even though this experiment showed a positive result with respect to repair detection and correction, Experiment 2 and Experiment 3 did not give high results when tested with wide range of input sentences. One of the major problems with this post-processing method is detection of interruption point. In the post-processing method, final fully activated structure is taken as the corrected part of the user's sentence. Based on the result of Experiment 1, smaller sentence with few repairs gave positive results. As the number of repair instances present in a sentence increases, the presence of a repair at the start of the utterance hinders the generation of parse trees for later part of the sentence.

Incremental processing of spoken language helps to detect and correct a repair as soon as the repair is recognized and enough information to correct the repair has been gathered. Using this method, 60 out of 95 sentences are corrected properly. 22 out of 35 incorrect output sentences are due to rules which can accept disfluent structures as proper structures.

7.1 Future Directions

The current work on spoken language processing provided more promising results with respect to speech repair detection and correction. The incremental processing method detected 89 out of 121 repairs with a recall rate of 73.5%.

Using the incremental processing method, the repairs are corrected as soon as they are detected. Due to different possible interpretation of the input structure, the incorrect interruption point is detected. Example 6-7 gives an example for the incorrect

interpretation of the corrected part of the utterance. In Example 6-7, "got I have got a great viewpoint above the tree just below the eastlake" is derived as the corrected part of the utterance instead of "I have a great viewpoint above the tree just below the eastlake". After detecting the partially parsed structure for the corrected part of the utterance, the PSMA searches for a suitable matching structure. Since there is no matching structure, the second part of the output represents partially parsed structure for the word "have" and word "I".

Example 7-1

```
I have got , oh , I have got a great viewpoint ... eh ... ... above
the tree ... eh justQl below the eastlake
OUTPUT :::
VP[VBP[got]S[NP[PRP[I]]VP[HV[have]VBP[got]NP[DT[a]ADJP[JJ[great]]NOM
[NN[viewpoint]]]PP[IN[above]NP[DT[the]NOM[NN[tree]]]]PP[QL[justQl]IN
[below]NP[DT[the]NOM[NN[eastlake]]]]]]]
got I have got a great viewpoint above the tree justQl below the
eastlake
OUTPUT :::
HV[have]
have
OUTPUT :::
PRP[I]
I
```

In Example 6-7, the presence of a repair is detected. Due to the detection of incorrect interruption point, the PSMA did not find a suitable matching structure. Looking at different possible interpretation of the alteration and ranking different interpretation can help detect the correct output. Another possible interpretation of the alteration in Example 6-7 is

Alteration:

```
S[NP[PRP[I]]VP[HV[have]VBP[got]NP[DT[a]ADJP[JJ[great]]NOM[NN[viewpoi
nt]]]PP[IN[above]NP[DT[the]NOM[NN[tree]]]]PP[QL[justQl]IN[below]NP[DT[t
he]NOM[NN[eastlake]]]]]]
```

I have got a great viewpoint above the tree justQl below the eastlake

The PSMA will then search for a suitable matching structure. The structure formed for the words "I got" will be detected as the reparandum. The structure of the reparandum is represented as

Reparandum:

```
S[NP[PRP[I]]VP[VBP[got]]
```

I got

The structure of the reparandum of its alteration will have the following similarities.

Matching – S –root node

Matching – NP

Matching – PRP – word category

Matching – I – first word

Matching – VP

No Match – VBP

As, there are similarities between the structure formed by the reparandum and its alteration, the corresponding structural interpretation of the alteration will be taken and the correct interpretation.

Grammar rules also play a major role in any robust parsing system. Certain rules can help generate structures for disfluent parts of an utterance. Further work on the fine-tuning of the rules should improve the repair detection capability of the parser. Out of all the rules used in this grammar, the verb phrase rules are the most complicated ones. The

verb phrase rules take a head verb, which is followed by other categories. A head verb can be followed by a noun phrase (NP) constituent (i.e. VP ← V NP) or a preposition phrase (i.e. VP ← V PP) constituent. These constituents that follow the verb are called as compliments. Jurafsky and Martin [23] explain the ways, in which the verb can be sub-categorized based on the compliments it takes. A verb like "want" can take a noun phrase (NP) compliment or an infinite verb phrase (inf-VP) compliment. Example 7-2 shows different compliments the verb "want" can take.

Example 7-2

1. want a book (verb with a NP-compliment (the book))
2. want to go to Toronto (verb with inf-VP compliment (to go))

Two different verb phrase structures can be derived based on the verb "want" are,

1. VP ← V NP
2. VP ← V VP
3. VP ← TO VP

Example 7-3 shows different compliments the verb "fly" can take.

Example 7-3

1. to fly to Toronto (verb with preposition phrase compliment (to Toronto))

The rule derived based on the verb "fly" is,

1. VP ← V PP

After generalization of the verb phrase rules, we have

VP ← V NP

VP ← V VP

VP ← TO VP

VP ← V PP

A verb phrase rule can have only the head verb.

VP ← V

If a user queries the system with "I fly want to fly to Toronto". The user starts with the word "fly" then reframes his query and say "want to fly to Toronto". Based on the generalized set of verb phrase rules, a parser cannot detect the interruption point in the utterance. Complete verb phrase structure formed for "fly want to fly to Toronto" is shown in Figure 7-1.

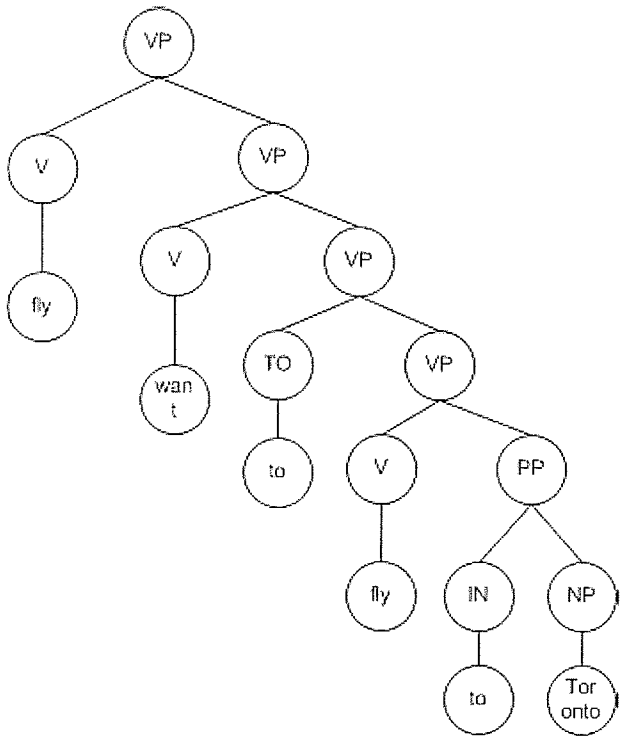


Figure 7-1: Verb Phrase structure for "fly want to Toronto".

If the verb is sub-categorized based on the compliments it can take, the interruption point can be detected. The set of verb phrase rules derived based on sub-categorization of verbs are,

VP ← V-fly PP

VP ← V-want INF-VP

VP ← V-want NP

The verb phrase structure formed for "fly want to fly to Toronto" using the set of rules derived after the verb sub-categorization is shown in Figure 7-2. Since the verb "fly" takes only a PP compliment, the phrase breakage occurs after the first occurrence of the word "fly". The interruption point is detected after the word "fly". The complete VP structure that follows the interruption point represents the alteration. The incomplete VP structure represents the reparandum. Since, an incomplete VP is followed by a complete verb phrase, overlapping of partially parsed structures takes place.

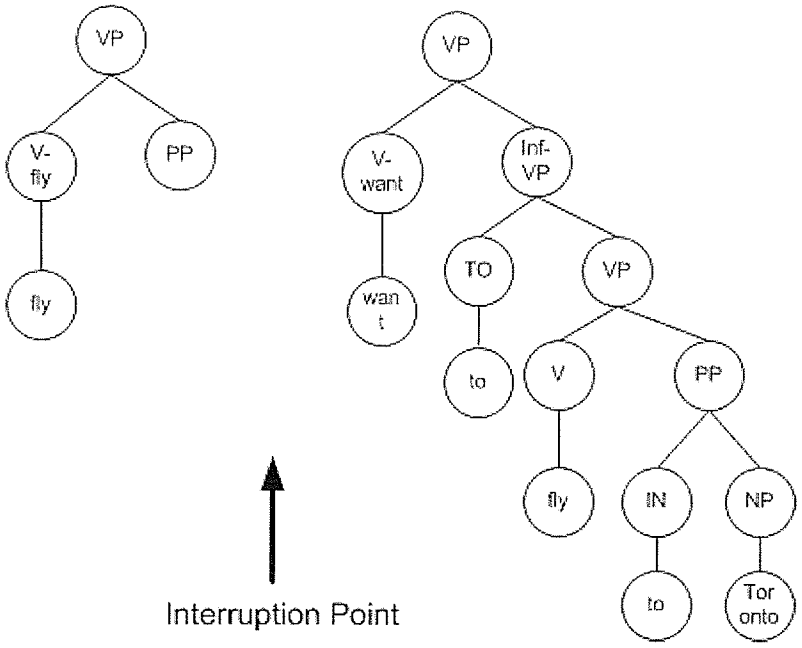


Figure 7-2: Verb phrase structure after verb sub-categorization.

In summary looking at different possible interpretation of the alteration and use of verb sub-categorization can be used to fine tune the integrated system for spoken language processing.

Bibliography

[1] J. Allen. Natural Language Understanding. Addison-Wesley, Redwood City, 1995.

[2] A. Anderson, M. Bader, E. Bard, E. Boyle, G.M. Doherty, S. Garrod, S. Isard, J. Kowtko, J. McAllister, J. Miller, C. Sotillo, H.S. Thompson and R. Weinert. The HCRC Map Task Corpus. *Language and Speech*. Vol 34, pp 351-366,1991.

[3] J. Bear, J. Dowding, and E. Shriberg. Integrating Multiple Knowledge Source for Detection and Correction of Repairs in Human-Computer Dialog. In *Proceedings of the 30th Annual Meeting of the association for Computational Liguistics (ACL-92),* pages 56-63, 1992.

[4] E. R. Blackmer and J. L. Mitton. Theories of Monitoring and Timing of Repairs in Spontaneous Speech. *Cognition*, 39:173-194, 1991.

[5] CALLHOME American English Transcripts. http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC97T14

[6] Context Free Grammar. http://www.laynetworks.com/Context%20Free%20Grammar.htm

[7] M. Core and L. Schubert. Handling Speech Repairs and Other Disruptions through Parser Metarules. Technical report, American Association for Artificial Intelligence (AAAI), 1997.

[8] M. Core and L. Schubert. Implementing Parser Metarules that Handle Speech Repairs and Other Disruptions. In *Proceedings of the 11th International FLAIRS Conference*, pages 283-288, Florida, May 1998.

[9] M. Core and L. Schubert. A Syntactic Framework for Speech Repairs and Other Disruptions. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL-99)*, June 1999.

[10] M. G. Core. Dialog Parsing: From Speech Repairs to Speech Acts. PhD thesis, University of Rochester, New York, 1999.

[11] F. Costa, P. Frasconi, V. Lombardo, and G. Soda. Towards Incremental Parsing of Natural Language using Recursive Neural Networks, *Applied Intelligence*, 19: 9- 25, 2003.

[12] J. Dowding, J. M. Gawron, D. Appelt, J. Bear, J. Cherny, R. Moore, and D. Moran. Gemini: A Natural Language System for Spoken Language Understanding. In Proceedings of the the 31st Annual Meeting of the Association for Computational Linguistics (ACL-93), pages 54-61, Columbus, Ohio, 1993.

[13] J. Earley. An Efficient Context-Free Parsing Algorithm. Communication of the ACM, 13(2):94-102, 1970.

[14] J. Godfrey, E. Holliman and J. McDaniel. "SWITCHBOARD: Telephone Speech Corpus for Research and Development". *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, San Francisco, California, USA, March 1992

[15] P. Heeman and J. Allen. Speech Repairs, Intonational Phrases and Discourse Markers, Modelling Speakers Utterance in Spoken Dialogue. *Computational Linguistics*, 25:527-571, December 1999.

[16] P. Heeman and J. F. Allen. Intonational Boundaries and Discourse Markers: Modeling Spoken Dialogue. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL-97)*, pages 254-61, July 1997.

[17] P. A. Heeman. Speech Repairs, Intonational Phrases and Discourse Markers, Modelling Speakers Utterance in Spoken Dialogue. PhD thesis, University of Rochester, 1997.

[18] P. A. Heeman, K. L. Kim, and J. F. Allen. Combining the Detection and Correction of Speech Repairs. In *Proceedings of the 4th International Conference on Spoken Language Processing*, pages 358-361, Philadelphia, October 1996.

[19] C.T. Hemphill, J.J. Godfrey and G.R. Doddington.. The ATIS Spoken Language Systems pilot corpus. *Proc. DARPA Speech and Natural Language Workshop*, Hidden Valley, pp. 96-101,1990.

[20] D. Hindle. Deterministic Parsing of Syntactic Non-fluencies. In *Proceedings of the 21st Annual Meeting of the Association of Computational Linguistics*, pages 123-128, 1983.

[21] Conversational Interaction and Spoken Dialogue Research Group. The Monroe Corpus. http://www.cs.rochester.edu/research/cisd/resources/monroe.

[22] Conversational Interaction and Spoken Dialog Research Group. The TRAINS Project: Natural Spoken Dialogue and Interactive Planning. http://www.cs.rochester.edu/research/trains.

[23] D. Jurafsky and J.H. Martin. Speech and Language Processing, An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition. Prentice Hall, NJ, 2000.

[24] D. Jurafsky, C. Wooters and G. Tajchman. The Berkely Restaurant Project. In *Proceedings of the International Conference on Spoken Language proceesing-94*, 2139-2142, Yokohama, 1994.

[25] C. Kemke. Generative Connectionist Parsing with Dynamic Neural Networks. In *Proceedings of the Second Workshop on Natural Language Processing and Neural Networks*, pages 31-37, Tokio, Japan, November 2001.

[26] A. Lavie. GLR*: A Robust Grammar-Focussed Parser for Spontaneously Spoken Language. PhD thesis, Carnegie Mellon University, 1996.

[27] W. J. Levelt. Monitoring and Self-repair in Speech. Cognition, 14:41-104, 1983.

[28] R. J. Lickley. Detecting Disfluency in Spontaneous Speech. PhD thesis, University of Edinburgh, Jan 1994.

[29] D. McKelvie. The Syntax of Disfluency in Spontaneous Spoken Language. HCRC Research Paper, May 1998.

[30] H. S. Kurtzman. Ambiguity Resolution in the Human Syntactic Parser: An Experimental Study. *Proceedings of the 10th International Conference on Computational Linguistics*, pages 481-485, Stanford, 1984.

[31] Microsoft. MiPad. http://research.microsoft.com/srg/mipad.aspx.

[32] T. Murase, S. Matsubara, Y. Kato and Y. Inagaki. Incremental CFG Parsing with Statistical Lexical Dependencies. *Proceedings of the 6th Natural Language Processing Pacific Rim Symposium*, pages 351-358, Tokyo, Japan, 2001.

[33] C. Nakatani and J. Hirschberg. A Speech-First Model for Repair Detection and Correction. In *Proceedings of the 31st annual meeting of the Association for Computational Linguistics*, pages 46-53, Columbus, Ohio, 1993.

[34] B. Pellom, W. Ward, and S. Pradhan. The CU Communicator: An Architecture for Dialog Systems. In Proceedings of the International Conference on Spoken Language Proceesing, Beijing, China, 2:723-726, Nov 2000.

[35] E. K. Ringger. A Robust Loose Coupling for Speech Recognition and Natural Language Understanding. PhD thesis, The University of Rochester, may 1995.

[36] G. Sampson. The SUSANNE Analytic Scheme. http://www.grsampson.net/RSue.html.

[37] Santa Barbara Corpus of Spoken American English. Http://www.ldc.upenn.edu/Projects/SBCSAE

[38] E. Shriberg and A. Stolcke. How Far Do Speakers back up Repairs? A Quantitative Model. In *Proceedings of the 5th International Conference on Spoken Language Proceesing,* pages 2183-2186, Sydney, Australia, 1998.

[39] E. Shriberg. Preliminaries to a Theory of Speech Disfluencies. PhD thesis, University of California, Berkely, 1994.

[40] E. shriberg, R. Bates, and A. Stolcke, "A prosody-Only Decision-Tree Model for Disfluency Detection", in Proceedings of the 5th European Conference on Speech Communication and Technology (Eurospeech-97), volume 5, pages 2383-2386, Rhodes, Greece, 1997.

[41] A. Stolcke and E. Shriberg. Statistical Language Modelling for Speech Disfluencies. In *proceedings of the IEEE Conference on Acoustics, Speech and Signal Proceesing*, volume 1, pages 405-408, 1996.

[42] TIMIT Corpus. http://www.ldc.upenn.edu/readme_files/timit.readme.html.

[43] S. Wermter and V. Weber. SCREEN: Learning a Flat Syntactic and Semantic Spoken Language Analysis Using Artificial Neural Networks. *Journal of Artificial Intelligence research*, 6:35-85, 1997.

Appendix -1

In the incremental method, the PSMA is triggered as soon as a disfluency is detected by the parser. 95 test utterances were used to test the integrated system. The first 60 sentences in the appendix have correct output structures generated by the integrated system. In these sentences disfluencies were detected and corrected properly. The remaining 35 sentences in the appendix have incorrect output structures generated by the integrated system.

These sentences are listed below, each followed by the respective output structure as generated by the parser, and a corrected version of the sentence, reconstructed from the parser output, and a final separation line.

Correct Output Structures

```
1  so thatPd is with the diamond ... mine on ... on our rightNn

OUTPUT :::
S[NP[PD[thatPd]]VP[VP[VBP[is]PP[IN[with]NP[DT[the]ADJP[JJ[diamond]]NOM[
NN[mine]]]]]PP[IN[on]NP[PPG[our]NOM[NN[rightNn]]]]]]

thatPd is with the diamond mine on our rightNn

**********************************************************************

2  no , dont go dont go any further than the desert

Previous structure = VP[VBP[go]]
Matching Root = VP
Repair =VP[DO[dont]VP[VBP[go]]]
Matching =VP
No Matching =DO
Previous structure = VP[DO[dont]VP[VBP[go]]]
Matching Root = VP
Repair =VP[DO[dont]VP[VBP[go]]]
Matching =VP
Matching =DO
Matching =dont
Matching =VP
```

```
Matching =VBP
Matching =go
No Matching =
OUTPUT ::: Overlapping Parse Trees :::VP[DO[dont]VP[VBP[go]]]
dont go
OUTPUT :::
VP[DO[dont]VP[VBP[go]ADJP[JJ[any]JJR[further]CS[than]NP[DT[the]NOM[NN[d
esert]]]]]]

dont go any further than the desert

**********************************************************************



3  but I am thatPd is not inIn a straight line

Reparandum = S[NP[PRP[I]]VP[BEM[am]NIL ]]
Repair =S[NP[PD[thatPd]]VP[VBP[is]]]
Matching =S
Matching =NP
No Matching =PD
OUTPUT ::: Overlapping Parse Trees :::S[NP[PD[thatPd]]VP[VBP[is]]]
thatPd is
OUTPUT :::
S[NP[PD[thatPd]]VP[VBP[is]PP[NOT[not]PP[IN[inIn]NP[DT[a]ADJP[JJ[straigh
t]]NOM[NN[line]]]]]]]

thatPd is not inIn a straight line

**********************************************************************


4  Right just just go between those two

OUTPUT :::
VP[ADVP[RB[just]]VP[VBP[go]PP[IN[between]NP[DT[those]NOM[NN[two]]]]]]
just go between those two
**********************************************************************


5  Right , I ... I am at the walled city Right

OUTPUT :::
S[NP[PRP[I]]VP[BEM[am]PP[IN[at]NP[DT[the]ADJP[JJ[walled]]NOM[NN[city]]]
]]]

I am at the walled city

**********************************************************************


6  so I am still ... at the ... ... parallel with the base

Reparandum = PP[IN[at]NP[DT[the]NIL ]]
Repair = ADJP[JJ[parallel]PP[IN[with]NP[DT[the]NOM[NN[base]]]]]
No Matching =ADJP
```

```
Matching Structure :: Partially Activated PP Followed by a Fully
Activated PP
OUTPUT ::: Overlapping Parse Trees
:::ADJP[JJ[parallel]PP[IN[with]NP[DT[the]NOM[NN[base]]]]]
parallel with the base
OUTPUT :::
S[NP[PRP[I]]VP[BEM[am]RB[still]ADJP[JJ[parallel]PP[IN[with]NP[DT[the]NO
M[NN[base]]]]]]

I am still parallel with the base

**********************************************************************


7  the carved wooden pole is rightQl ... at my ... rightQl at the
bottom

Previous structure = PP[QL[rightQl]IN[at]]
Matching Root = PP
Repair =PP[QL[rightQl]IN[at]]
Matching =PP
Matching =QL
Matching =rightQl
Matching =IN
Matching =at
No Matching =
OUTPUT ::: Overlapping Parse Trees :::PP[QL[rightQl]IN[at]]
rightQl at
OUTPUT :::
S[NP[DT[the]ADJP[JJ[carved]ADJP[JJ[wooden]]]NOM[NN[pole]]]VP[VBP[is]PP[
QL[rightQl]IN[at]NP[DT[the]NOM[NN[bottom]]]]]

the carved wooden pole is rightQl at the bottom

**********************************************************************


8  Right then , pass ... pass ... below ... the ghost town

OUTPUT :::
VP[VBP[pass]PP[IN[below]NP[DT[the]ADJP[JJ[ghost]]NOM[NN[town]]]]]

pass below the ghost town

**********************************************************************


9  Right , below the go below the wooden pole

Reparandum = PP[IN[below]NP[DT[the]NIL ]]
Repair
=VP[VP[VBP[go]]PP[IN[below]NP[DT[the]ADJP[JJ[wooden]]NOM[NN[pole]]]]]
No Matching =VP
Matching Structure :: Partially Activated PP Followed by a Fully
Activated PP
OUTPUT ::: Overlapping Parse Trees
:::VP[VP[VBP[go]]PP[IN[below]NP[DT[the]ADJP[JJ[wooden]]NOM[NN[pole]]]]]
```

```
go below the wooden pole
Reparandum = PP[IN[below]NP[DT[the]NIL ]]
Repair =
VP[VBP[go]PP[IN[below]NP[DT[the]ADJP[JJ[wooden]]NOM[NN[pole]]]]]
No Matching =VP
Matching Structure :: Partially Activated PP Followed by a Fully
Activated PP
OUTPUT ::: Overlapping Parse Trees
:::VP[VBP[go]PP[IN[below]NP[DT[the]ADJP[JJ[wooden]]NOM[NN[pole]]]]]
go below the wooden pole
OUTPUT :::
VP[VBP[go]PP[IN[below]NP[DT[the]ADJP[JJ[wooden]]NOM[NN[pole]]]]]

go below the wooden pole
**************************************************************************


10  no , dont go dont go below the wooden pole

Previous structure = VP[VBP[go]]
Matching Root = VP
Repair =VP[DO[dont]VP[VBP[go]]]
Matching =VP
No Matching =DO
Previous structure = VP[DO[dont]VP[VBP[go]]]
Matching Root = VP
Repair =VP[DO[dont]VP[VBP[go]]]
Matching =VP
Matching =DO
Matching =dont
Matching =VP
Matching =VBP
Matching =go
No Matching =
OUTPUT ::: Overlapping Parse Trees :::VP[DO[dont]VP[VBP[go]]]
dont go
OUTPUT :::
VP[DO[dont]VP[VBP[go]PP[IN[below]NP[DT[the]ADJP[JJ[wooden]]NOM[NN[pole]
]]]]]

dont go below the wooden pole

**************************************************************************


11  you got to you have got to pass below the flat rocks

Previous structure = S[NP[PRP[you]]VP[VBP[got]]]
Matching Root = S
Repair =S[NP[PRP[you]]VP[HV[have]VBP[got]IVP[TO[to]VBP[pass]]]]
Matching =S
Matching =NP
Matching =PRP
Matching =you
Matching =VP
No Matching =HV
```

```
OUTPUT ::: Overlapping Parse Trees
:::S[NP[PRP[you]]VP[HV[have]VBP[got]IVP[TO[to]VBP[pass]]]]
you have got to pass
OUTPUT :::
S[NP[PRP[you]]VP[HV[have]VBP[got]IVP[TO[to]VBP[pass]PP[IN[below]NP[DT[t
he]ADJP[JJ[flat]]NOM[NN[rocks]]]]]]]

you have got to pass below the flat rocks

**********************************************************************


12  Right okay , now , go ... go right ... from the diamond mine

OUTPUT :::
VP[VP[VBP[go]ADVP[RB[right]]]PP[IN[from]NP[DT[the]ADJP[JJ[diamond]]NOM[
NN[mine]]]]]

go right from the diamond mine

**********************************************************************


13  and so you areAx going to ... you areAx going to pass by the ...
graveyard asQl well

Reparandum = S[NP[PRP[you]]VP[AUX[areAx]VBP[going]IVP[TO[to]NIL ]]]
Repair =S[NP[PRP[you]]VP[AUX[areAx]VP[VBP[going]]]]
Matching =S
Matching =NP
Matching =PRP
Matching =you
Matching =VP
Matching =AUX
Matching =areAx
No Matching =VP
OUTPUT ::: Overlapping Parse Trees
:::S[NP[PRP[you]]VP[AUX[areAx]VP[VBP[going]]]]
you areAx going
OUTPUT :::
S[NP[PRP[you]]VP[VP[AUX[areAx]VBP[going]IVP[TO[to]VBP[pass]PP[IN[by]NP[
DT[the]NOM[NN[graveyard]]]]]]ADVP[QL[asQl]RB[well]]]]
you areAx going to pass by the graveyard asQl well
**********************************************************************

14  I am ... I am going to hit the roundrocks , no

Reparandum = S[NP[PRP[I]]VP[BEM[am]NIL ]]
Repair =S[NP[PRP[I]]VP[BEM[am]VP[VBP[going]]]]
Matching =S
Matching =NP
Matching =PRP
Matching =I
Matching =VP
Matching =BEM
Matching =am
No Matching =VP
```

```
OUTPUT ::: Overlapping Parse Trees
:::S[NP[PRP[I]]VP[BEM[am]VP[VBP[going]]]]
I am going
OUTPUT :::
AFF[no]
no
OUTPUT :::
S[NP[PRP[I]]VP[BEM[am]VP[VBP[going]IVP[TO[to]VBP[hit]NP[DT[the]NOM[NN[r
oundrocks]]]]]]]
I am going to hit the roundrocks
**********************************************************************

15  where , well , I ... you I dont suppose you ... you you can go
outside the picket fence

Previous structure = ADVP[WRB[where]]
Matching Root = ADVP
Repair =ADVP[RB[well]]
Matching =ADVP
No Matching =RB
Reparandum = ADVP[ADVP[WRB[where]]]
Repair =ADVP[RB[well]]
Matching =ADVP
No Matching =RB
Matching Structure :: Partially Activated ADVP Followed by a Fully
Activated ADVP
OUTPUT ::: Overlapping Parse Trees :::ADVP[RB[well]]
well
Previous structure = NP[PRP[I]]
Matching Root = NP
Repair =NP[PRP[you]]
Matching =NP
Matching =PRP
No Matching =you
OUTPUT ::: Overlapping Parse Trees :::NP[PRP[you]]
you
Previous structure = NP[PRP[you]]
Matching Root = NP
Repair =NP[PRP[I]]
Matching =NP
Matching =PRP
No Matching =I
OUTPUT ::: Overlapping Parse Trees :::NP[PRP[I]]
I
OUTPUT :::
S[NP[ADVP[RB[well]]NP[PRP[I]]]VP[DO[dont]VP[VBP[suppose]S[NP[PRP[you]]V
P[MOD[can]VP[VBP[go]PP[IN[outside]NP[DT[the]NOM[NN[picket]NOM[NN[fence]
]]]]]]]]]

well I dont suppose you can go outside the picket fence

**********************************************************************

16  Right , can you can you can you well can you see the abandoned
cottage

Reparandum = S[MOD[can]NP[PRP[you]]]
```

```
Repair =S[MOD[can]NP[PRP[you]]VP[VBP[see]]]
Matching =S
Matching =MOD
Matching =can
Matching =NP
Matching =PRP
Matching =you
OUTPUT ::: Overlapping Parse Trees
:::S[MOD[can]NP[PRP[you]]VP[VBP[see]]]
can you see
Reparandum = S[MOD[can]NP[PRP[you]]]
Repair =S[MOD[can]NP[PRP[you]]VP[VBP[see]]]
Matching =S
Matching =MOD
Matching =can
Matching =NP
Matching =PRP
Matching =you
OUTPUT ::: Overlapping Parse Trees
:::S[MOD[can]NP[PRP[you]]VP[VBP[see]]]
can you see
Reparandum = S[MOD[can]NP[PRP[you]]]
Repair =S[MOD[can]NP[PRP[you]]VP[VBP[see]]]
Matching =S
Matching =MOD
Matching =can
Matching =NP
Matching =PRP
Matching =you
OUTPUT ::: Overlapping Parse Trees
:::S[MOD[can]NP[PRP[you]]VP[VBP[see]]]
can you see
OUTPUT :::
S[MOD[can]NP[PRP[you]]VP[VBP[see]NP[DT[the]ADJP[JJ[abandoned]]NOM[NN[co
ttage]]]]]

can you see the abandoned cottage

**********************************************************************

17  I would if I need to go ... ... beneath the old mill Right , okay

Reparandum = S[MOD[would]]
Repair =S[NP[PRP[I]]VP[VBP[need]]]
Matching =S
No Matching =NP
Matching Structure :: Partially Activated S Followed by a Fully
Activated S
OUTPUT ::: Overlapping Parse Trees :::S[NP[PRP[I]]VP[VBP[need]]]
I need
Reparandum = S[NP[PRP[I]]VP[MOD[would]NIL ]]
Repair =S[NP[PRP[I]]VP[VBP[need]]]
Matching =S
Matching =NP
Matching =PRP
Matching =I
Matching =VP
```

```
No Matching =VBP
OUTPUT ::: Overlapping Parse Trees :::S[NP[PRP[I]]VP[VBP[need]]]
I need
OUTPUT :::
S[NP[PRP[I]]VP[VBP[need]IVP[TO[to]VBP[go]PP[IN[beneath]NP[DT[the]ADJP[J
J[old]]NOM[NN[mill]]]]]]]
I need to go beneath the old mill
**************************************************************************

18  I am I am going rightRp under the ... the mill okay

Reparandum = S[NP[PRP[I]]VP[BEM[am]NIL ]]
Repair =S[NP[PRP[I]]VP[BEM[am]VP[VBP[going]]]]
Matching =S
Matching =NP
Matching =PRP
Matching =I
Matching =VP
Matching =BEM
Matching =am
No Matching =VP
OUTPUT ::: Overlapping Parse Trees
:::S[NP[PRP[I]]VP[BEM[am]VP[VBP[going]]]]
I am going
OUTPUT :::
S[NP[PRP[I]]VP[VP[BEM[am]VP[VBP[going]PP[RP[rightRp]IN[under]]]]NP[DT[t
he]NOM[NN[mill]]]]]

I am going rightRp under the mill

**************************************************************************

19  uh-huh ... thatPd is fine , so if you you areAx going To the old
mill

OUTPUT :::
S[S[NP[PD[thatPd]]VP[VBP[is]ADJP[JJ[fine]]]]CS[so]CS[if]S[NP[PRP[you]]V
P[AUX[areAx]VBP[going]PP[IN[To]NP[DT[the]ADJP[JJ[old]]NOM[NN[mill]]]]]]
]

thatPd is fine so if you areAx going To the old mill

**************************************************************************

20  like , thatPd is how you go just just go round the top

OUTPUT :::
S[NP[PD[thatPd]]VP[VBP[is]S[VP[ADVP[WRB[how]NP[PRP[you]]VP[VBP[go]ADVP[
RB[just]]]]VP[VBP[go]PP[IN[round]NP[DT[the]NOM[NN[top]]]]]]]]]

thatPd is how you go just go round the top

**************************************************************************

21  you actually go round the side where the where the wheel is

Previous structure = ADVP[WRB[where]]
```

```
Matching Root = ADVP
Repair =ADVP[WRB[where]]
Matching =ADVP
Matching =WRB
Matching =where
No Matching =
OUTPUT ::: Overlapping Parse Trees :::ADVP[WRB[where]]
where
OUTPUT :::
S[NP[PRP[you]]VP[VP[ADVP[RB[actually]]VP[VBP[go]PP[IN[round]NP[DT[the]N
OM[NN[side]]]]]]ADVP[WRB[where]NP[DT[the]NOM[NN[wheel]]]VP[VBP[is]]]]]

you actually go round the side where the wheel is

**********************************************************************

22  uh-huh , where where the steps are and round the on top of the roof
and then you go roundRp ... To your rightNn

Reparandum = PP[IN[round]NP[DT[the]NIL ]]
Repair =PP[IN[on]NN[top]]
Matching =PP
Matching =IN
No Matching =on
OUTPUT ::: Overlapping Parse Trees :::PP[IN[on]NN[top]]
on top
OUTPUT :::
S[S[NP[ADVP[WRB[where]]NP[DT[the]NOM[NN[steps]]]]VP[VBP[are]]]CC[and]S[
PP[IN[on]NP[NP[NN[top]PP[IN[of]NP[DT[the]NOM[NN[roof]]]]]CC[and]NP[ADVP
[RB[then]]NP[PRP[you]]]]]VP[VP[VBP[go]PP[RP[roundRp]IN[To]]]NP[PPG[your
]NOM[NN[rightNn]]]]]]

where the steps are and on top of the roof and then you go roundRp To
your rightNn

**********************************************************************


23  see where this ... the ... the river comes in

Reparandum = NP[DT[this]]
Repair =NP[DT[the]NOM[NN[river]]]
Matching =NP
Matching =DT
No Matching =the
OUTPUT ::: Overlapping Parse Trees :::NP[DT[the]NOM[NN[river]]]
the river
OUTPUT :::
VP[VBP[see]ADVP[WRB[where]NP[DT[the]NOM[NN[river]]]VP[VBP[comes]RP[in]]
]]

see where the river comes in

**********************************************************************

24  follow follow the line of crane bay ... down To ... justQl above
the corner ... of the bay
```

```
OUTPUT :::
VP[VP[VP[VP[VBP[follow]NP[DT[the]NOM[NN[line]]]PP[IN[of]NN[crane]]]NP[N
N[bay]PP[RP[down]IN[To]]]]PP[QL[justQl]IN[above]NP[DT[the]NOM[NN[corner
]]]]]PP[IN[of]NP[DT[the]NOM[NN[bay]]]]]

follow the line of crane bay down To justQl above the corner of the bay

**********************************************************************

25  do you have hills To your ... the left of the dead tree and justQl
up a bit

Reparandum = NP[PPG[your]]
Repair =NP[DT[the]NOM[NN[left]]]
Matching =NP
No Matching =DT
Matching Structure :: Partially Activated NP Followed by a Fully
Activated NP
OUTPUT ::: Overlapping Parse Trees :::NP[DT[the]NOM[NN[left]]]
the left
OUTPUT :::
VP[VP[VP[DO[do]NP[PRP[you]]VP[HV[have]NOM[NN[hills]]]]PP[IN[To]NP[DT[th
e]NOM[NN[left]]]]]PP[PP[IN[of]NP[DT[the]ADJP[JJ[dead]]NOM[NN[tree]]]]CC
[and]PP[QL[justQl]IN[up]NP[DT[a]NOM[NN[bit]]]]]]

do you have hills To the left of the dead tree and justQl up a bit

**********************************************************************

26  pirateship is rightQl downIn the ... the ... the southern

OUTPUT :::
S[NP[NNP[pirateship]]VP[VBP[is]PP[QL[rightQl]IN[downIn]NP[DT[the]NOM[NN
[southern]]]]]]

pirateship is rightQl downIn the southern

**********************************************************************

27  does your ... the river forkVb into one goes into the lagoon and
the other one goes off the page

Reparandum = NP[PPG[your]]
Repair =NP[DT[the]NOM[NN[river]]]
Matching =NP
No Matching =DT
Matching Structure :: Partially Activated NP Followed by a Fully
Activated NP
OUTPUT ::: Overlapping Parse Trees :::NP[DT[the]NOM[NN[river]]]
the river
OUTPUT :::
VP[DO[does]NP[DT[the]NOM[NN[river]]]VP[VBP[forkVb]S[PP[IN[into]NP[PN[on
e]]]VP[VBP[goes]S[PP[IN[into]NP[NP[DT[the]NOM[NN[lagoon]]]CC[and]NP[DT[
the]AP[other]PN[one]]]]VP[VBP[goes]PP[IN[off]NP[DT[the]NOM[NN[page]]]]]
]]]]]
```

```
does the river forkVb into one goes into the lagoon and the other one
goes off the page

**********************************************************************

28  you want to you want ... my path ... to ... be a largeenough c so
as it goes down

Previous structure = S[NP[PRP[you]]VP[VBP[want]]]
Matching Root = S
Repair =S[NP[PRP[you]]VP[VBP[want]]]
Matching =S
Matching =NP
Matching =PRP
Matching =you
Matching =VP
Matching =VBP
Matching =want
No Matching =
OUTPUT ::: Overlapping Parse Trees :::S[NP[PRP[you]]VP[VBP[want]]]
you want
OUTPUT :::
S[S[NP[PRP[you]]VP[VBP[want]NP[PPG[my]NOM[NN[path]]]IVP[TO[to]BE[be]NP[
DT[a]ADJP[JJ[largeenough]]NOM[NN[c]]]]]]CS[so]CS[as]S[NP[PRP[it]]VP[VBP
[goes]RP[down]]]]

you want my path to be a largeenough c so as it goes down

**********************************************************************

29  it ... and it crosses overIn the ... the mountainstream ... and
then goes directly south underneath the dead tree

OUTPUT :::
S[NP[PRP[it]]VP[VP[VBP[crosses]PP[IN[overIn]NP[DT[the]NOM[NN[mountainst
ream]]]]]CC[and]VP[ADVP[RB[then]]VP[VBP[goes]PP[QL[directly]RP[south]PP
[IN[underneath]NP[DT[the]ADJP[JJ[dead]]NOM[NN[tree]]]]]]]]]

it crosses overIn the mountainstream and then goes directly south
underneath the dead tree

**********************************************************************

30  are you on the other side of the river from the ... the hills

OUTPUT :::
VP[VP[VP[VBP[are]NP[PRP[you]]PP[IN[on]NP[DT[the]AP[other]NN[side]]]]PP[
IN[of]NP[DT[the]NOM[NN[river]]]]]PP[IN[from]NP[DT[the]NOM[NN[hills]]]]]

are you on the other side of the river from the hills

**********************************************************************


31  dont ... dont cross the river at thatDt point

OUTPUT :::
```

```
VP[DO[dont]VP[VBP[cross]NP[DT[the]NOM[NN[river]]]PP[IN[at]NP[DT[thatDt]
NOM[NN[point]]]]]]

dont cross the river at thatDt point

**************************************************************************

32  do the ... do a c so that it , it ... it is like a backwards c

Reparandum = S[DO[do]NP[DT[the]NIL ]]
Repair =S[DO[do]NP[DT[a]NOM[NN[c]]]]
Matching =S
Matching =DO
Matching =do
Matching =NP
Matching =DT
No Matching =a
OUTPUT ::: Overlapping Parse Trees :::S[DO[do]NP[DT[a]NOM[NN[c]]]]
do a c
OUTPUT :::
S[S[DO[do]NP[DT[a]NOM[NN[c]]]]CS[so]CS[that]S[NP[PRP[it]]VP[VP[VBP[is]R
P[like]]NP[DT[a]ADJP[JJ[backwards]]NOM[NN[c]]]]]]

do a c so that it is like a backwards c

**************************************************************************

33  Right , come across ... come round ... and ... across the top of
the gate

Previous structure = VP[VBP[come]]
Matching Root = VP
Repair =VP[VBP[come]]
Matching =VP
Matching =VBP
Matching =come
No Matching =
OUTPUT ::: Overlapping Parse Trees :::VP[VBP[come]]
come
OUTPUT :::
VP[VP[VBP[come]PP[IN[round]CC[and]IN[across]NP[DT[the]NOM[NN[top]]]]]PP
[IN[of]NP[DT[the]NOM[NN[gate]]]]]

come round and across the top of the gate

**************************************************************************


34  uh-huh , To To the left of the telephonekiosk

OUTPUT :::
PP[IN[To]NP[NP[DT[the]NOM[NN[left]]]PP[IN[of]NP[DT[the]NOM[NN[telephone
kiosk]]]]]]

To the left of the telephonekiosk

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

```
35  yes , To my what was my left ... as I am looking at it here

Reparandum = NP[PPG[my]]
Repair =NP[WPS[what]]
Matching =NP
No Matching =WPS
Matching Structure :: Partially Activated NP Followed by a Fully
Activated NP
OUTPUT ::: Overlapping Parse Trees :::NP[WPS[what]]
what
OUTPUT :::
S[S[PP[IN[To]NP[WPS[what]]]VP[VBP[was]NP[PPG[my]NOM[NN[left]]]]]CS[as]S
[NP[PRP[I]]VP[BEM[am]VP[VBP[looking]PP[IN[at]NP[PRP[it]]NP[PN[here]]]]]
]]
To what was my left as I am looking at it here

**********************************************************************


36  oh no , I have got a rocketlaunch justQl justQl below the eastlake

OUTPUT :::
S[NP[PRP[I]]VP[VP[HV[have]VBP[got]NP[DT[a]NOM[NN[rocketlaunch]]]]PP[QL[
justQl]IN[below]NP[DT[the]NOM[NN[eastlake]]]]]]
I have got a rocketlaunch justQl below the eastlake
**********************************************************************


37  thatPd is it , Right , so what what would we endVb justQl
underneath the eastlake do we

Reparandum = S[NP[WPS[what]]]
Repair =S[MOD[would]NP[PRP[we]]VP[VBP[endVb]]]
Matching =S
No Matching =MOD
Matching Structure :: Partially Activated S Followed by a Fully
Activated S
OUTPUT ::: Overlapping Parse Trees
:::S[MOD[would]NP[PRP[we]]VP[VBP[endVb]]]
would we endVb
OUTPUT :::
S[S[NP[PD[thatPd]]VP[VBP[is]NP[PRP[it]]]]CS[so]S[MOD[would]NP[PRP[we]]V
P[VBP[endVb]S[PP[QL[justQl]IN[underneath]NP[DT[the]NOM[NN[eastlake]]]]S
[DO[do]NP[PRP[we]]]]]]]

thatPd is it so would we endVb justQl underneath the eastlake do we

**********************************************************************

38  mmhmm , head upRp To the righthand side of thatPd ... but avoiding
the the dead ... tree

OUTPUT :::
S[S[VP[VP[VP[VBP[head]PP[RP[upRp]IN[To]]]NP[DT[the]ADJP[JJ[righthand]]N
OM[NN[side]]]]PP[IN[of]NP[PD[thatPd]]]]]CS[but]S[VP[VBP[avoiding]NP[DT[
the]ADJP[JJ[dead]]NOM[NN[tree]]]]]]
```

```
head upRp To the righthand side of thatPd but avoiding the dead tree

**********************************************************************


39  I have dont have a dead tree

Reparandum = VP[HV[have]]
Repair =VP[DO[dont]VP[HV[have]NP[DT[a]ADJP[JJ[dead]]NOM[NN[tree]]]]]
Matching =VP
No Matching =DO
Matching Structure :: Partially Activated VP Followed by a Fully
Activated VP
OUTPUT ::: Overlapping Parse Trees
:::VP[DO[dont]VP[HV[have]NP[DT[a]ADJP[JJ[dead]]NOM[NN[tree]]]]]
dont have a dead tree
OUTPUT :::
S[NP[PRP[I]]VP[DO[dont]VP[HV[have]NP[DT[a]ADJP[JJ[dead]]NOM[NN[tree]]]]
]]

I dont have a dead tree

**********************************************************************

40  it is farQl ... eh wayQl over on the righthand side about justQl
upRp from the middle of the map

Reparandum = PP[QL[farQl]]
Repair =
PP[QL[wayQl]RP[over]PP[IN[on]NP[NP[NP[DT[the]ADJP[JJ[righthand]]NOM[NN[
side]]]PP[RP[about]PP[QL[justQl]RP[upRp]PP[IN[from]NP[DT[the]NOM[NN[mid
dle]]]]]]]PP[IN[of]NP[DT[the]NOM[NN[map]]]]]]]
wayQl over on the righthand side about justQl upRp from the middle of
the map
Matching =PP
Matching =QL
No Matching =wayQl
OUTPUT ::: Overlapping Parse Trees
:::PP[QL[wayQl]RP[over]PP[IN[on]NP[NP[NP[DT[the]ADJP[JJ[righthand]]NOM[
NN[side]]]PP[RP[about]PP[QL[justQl]RP[upRp]PP[IN[from]NP[DT[the]NOM[NN[
middle]]]]]]]PP[IN[of]NP[DT[the]NOM[NN[map]]]]]]]
wayQl over on the righthand side about justQl upRp from the middle of
the map
OUTPUT :::
S[NP[PRP[it]]VP[VBP[is]PP[QL[wayQl]RP[over]PP[IN[on]NP[NP[NP[DT[the]ADJ
P[JJ[righthand]]NOM[NN[side]]]PP[RP[about]PP[QL[justQl]RP[upRp]PP[IN[fr
om]NP[DT[the]NOM[NN[middle]]]]]]]PP[IN[of]NP[DT[the]NOM[NN[map]]]]]]]]]

it is wayQl over on the righthand side about justQl upRp from the
middle of the map

**********************************************************************

41  Right okay , wellAff thatPd I wellAff I assume thatPd is the same
thing
```

```
Previous structure = NP[PD[thatPd]]
Matching Root = NP
Repair =NP[PRP[I]]
Matching =NP
No Matching =PRP
Reparandum = S[NP[PD[thatPd]]]
Repair =S[NP[PRP[I]]VP[VBP[assume]]]
Matching =S
Matching =NP
No Matching =PRP
OUTPUT ::: Overlapping Parse Trees :::S[NP[PRP[I]]VP[VBP[assume]]]
I assume
OUTPUT :::
S[NP[PRP[I]]VP[VBP[assume]S[NP[PD[thatPd]]VP[VBP[is]NP[DT[the]ADJP[JJ[s
ame]]NOM[NN[thing]]]]]]]

I assume thatPd is the same thing

**********************************************************************

42  and thatPd is my the finish

Reparandum = NP[PPG[my]]
Repair =NP[DT[the]NOM[NN[finish]]]
Matching =NP
No Matching =DT
Matching Structure :: Partially Activated NP Followed by a Fully
Activated NP
OUTPUT ::: Overlapping Parse Trees :::NP[DT[the]NOM[NN[finish]]]
the finish
OUTPUT :::
S[NP[PD[thatPd]]VP[VBP[is]NP[DT[the]NOM[NN[finish]]]]]

thatPd is the finish

**********************************************************************


43  okay , now you ... you need to go parallel To the side of the map

OUTPUT :::
S[NP[PRP[you]]VP[VP[VBP[need]IVP[TO[to]VBP[go]ADJP[JJ[parallel]PP[IN[To
]NP[DT[the]NOM[NN[side]]]]]]]PP[IN[of]NP[DT[the]NOM[NN[map]]]]]]

you need to go parallel To the side of the map

**********************************************************************


44  then you need to you need to bend it roundRp To the ropebridge

Previous structure = S[NP[PRP[you]]VP[VBP[need]]]
Matching Root = S
Repair =S[NP[PRP[you]]VP[VBP[need]]]
Matching =S
Matching =NP
Matching =PRP
```

```
Matching =you
Matching =VP
Matching =VBP
Matching =need
No Matching =
OUTPUT ::: Overlapping Parse Trees :::S[NP[PRP[you]]VP[VBP[need]]]
you need
OUTPUT :::
S[NP[PRP[you]]VP[VP[VP[VBP[need]IVP[TO[to]VBP[bend]NP[PRP[it]]]]PP[RP[r
oundRp]IN[To]]]NP[DT[the]NOM[NN[ropebridge]]]]]

you need to bend it roundRp To the ropebridge

**********************************************************************

45  you areAx going across the ropebridge so your your line should stop
... at the beginning of the ropebridge and come out again at the end

OUTPUT :::
S[S[NP[PRP[you]]VP[AUX[areAx]VBP[going]PP[IN[across]NP[DT[the]NOM[NN[ro
pebridge]]]]]]CS[so]S[NP[PPG[your]NOM[NN[line]]]VP[VP[VP[MOD[should]VP[
VBP[stop]PP[IN[at]NP[DT[the]NOM[NN[beginning]]]]]]PP[IN[of]NP[DT[the]NO
M[NN[ropebridge]]]]]CC[and]VP[VBP[come]RP[out]ADVP[RB[again]]PP[IN[at]N
P[DT[the]NOM[NN[end]]]]]]]]

you areAx going across the ropebridge so your line should stop at the
beginning of the ropebridge and come out again at the end

**********************************************************************

46  okay , we Right , you are at the bottom of the waterfall now okay

Previous structure = NP[PRP[we]]
Matching Root = NP
Repair =NP[PRP[you]]
Matching =NP
Matching =PRP
No Matching =you
OUTPUT ::: Overlapping Parse Trees :::NP[PRP[you]]
you
OUTPUT :::
S[NP[PRP[you]]VP[VP[VP[VBP[are]PP[IN[at]NP[DT[the]NOM[NN[bottom]]]]]PP[
IN[of]NP[DT[the]NOM[NN[waterfall]]]]]ADVP[RB[now]]]]

you are at the bottom of the waterfall now

**********************************************************************

47  okay , wellAff you have ... so you are at the bottom of the
whitemountain now

Reparandum = S[NP[PRP[you]]VP[HV[have]NIL ]]
Repair =S[NP[PRP[you]]VP[VBP[are]]]
Matching =S
Matching =NP
Matching =PRP
Matching =you
```

```
Matching =VP
No Matching =VBP
OUTPUT ::: Overlapping Parse Trees :::S[NP[PRP[you]]VP[VBP[are]]]
you are
OUTPUT :::
S[NP[PRP[you]]VP[VP[VP[VBP[are]PP[IN[at]NP[DT[the]NOM[NN[bottom]]]]]PP[
IN[of]NP[DT[the]NOM[NN[whitemountain]]]]]ADVP[RB[now]]]]

you are at the bottom of the whitemountain now

**********************************************************************

48  go ... go rightRp , and you stop on on nothing

OUTPUT :::
S[S[VP[VBP[go]RP[rightRp]]]CC[and]S[NP[PRP[you]]VP[VBP[stop]PP[IN[on]NP
[PN[nothing]]]]]]

go rightRp and you stop on nothing

**********************************************************************

49  yeah , do you have a ... a wagonwheel justQl To the rightNn of the
... the diamond mine

OUTPUT :::
VP[VP[VP[DO[do]NP[PRP[you]]VP[HV[have]NP[DT[a]NOM[NN[wagonwheel]]]]]PP[
QL[justQl]IN[To]NP[DT[the]NOM[NN[rightNn]]]]]PP[IN[of]NP[DT[the]ADJP[JJ
[diamond]]NOM[NN[mine]]]]]

do you have a wagonwheel justQl To the rightNn of the diamond mine

**********************************************************************

50  okay , To ... rightQl To the edge of the page

Reparandum = PP[IN[To]]
Repair =PP[QL[rightQl]IN[To]]
Matching =PP
No Matching =QL
Matching Structure :: Partially Activated PP Followed by a Fully
Activated PP
OUTPUT ::: Overlapping Parse Trees :::PP[QL[rightQl]IN[To]]
rightQl To
OUTPUT :::
PP[QL[rightQl]IN[To]NP[NP[DT[the]NOM[NN[edge]]]PP[IN[of]NP[DT[the]NOM[N
N[page]]]]]]

rightQl To the edge of the page

**********************************************************************


51  Right , you have got in you are about the riftvalley

Reparandum = S[S[VP[VBP[got]RP[in]]]]
```

```
Repair =
S[NP[PRP[you]]VP[VP[VBP[are]RP[about]]NP[DT[the]NOM[NN[riftvalley]]]]]
Matching =S
No Matching =NP
Matching Structure :: Partially Activated S Followed by a Fully
Activated S
OUTPUT ::: Overlapping Parse Trees
:::S[NP[PRP[you]]VP[VP[VBP[are]RP[about]]NP[DT[the]NOM[NN[riftvalley]]]
]]

you are about the riftvalley
Reparandum = S[NP[PRP[you]]VP[HV[have]NIL ]]
Repair =
S[NP[PRP[you]]VP[VP[VBP[are]RP[about]]NP[DT[the]NOM[NN[riftvalley]]]]]
Matching =S
Matching =NP
Matching =PRP
Matching =you
Matching =VP
No Matching =VP
OUTPUT ::: Overlapping Parse Trees
:::S[NP[PRP[you]]VP[VP[VBP[are]RP[about]]NP[DT[the]NOM[NN[riftvalley]]]
]]
you are about the riftvalley
OUTPUT :::
S[NP[PRP[you]]VP[VP[VBP[are]RP[about]]NP[DT[the]NOM[NN[riftvalley]]]]]

you are about the riftvalley
**************************************************************************

52  oh no aye yeah , you dont have likeIn above the the rapids To its
slightly To its rightNn you dont have a stonecreek

Reparandum = PP[IN[likeIn]]
Repair =PP[IN[above]NP[DT[the]NOM[NN[rapids]]]]
Matching =PP
Matching =IN
No Matching =above
OUTPUT ::: Overlapping Parse Trees
:::PP[IN[above]NP[DT[the]NOM[NN[rapids]]]]
above the rapids
Reparandum = PP[IN[To]NP[PPG[its]NIL ]]
Repair =PP[QL[slightly]IN[To]]
Matching =PP
No Matching =QL
Matching Structure :: Partially Activated PP Followed by a Fully
Activated PP
OUTPUT ::: Overlapping Parse Trees :::PP[QL[slightly]IN[To]]
slightly To
Reparandum = S[NP[PRP[you]]VP[DO[dont]VP[HV[have]NIL ]]]
Repair
=S[PP[IN[above]NP[NP[DT[the]NOM[NN[rapids]]]PP[QL[slightly]IN[To]NP[PPG
[its]NOM[NN[rightNn]]]]]NP[PRP[you]]]VP[DO[dont]VP[HV[have]NP[DT[a]NOM[
NN[stonecreek]]]]]]
Matching =S
No Matching =PP
```

```
Matching Structure :: Partially Activated S Followed by a Fully
Activated S
OUTPUT ::: Overlapping Parse Trees
:::S[PP[IN[above]NP[NP[DT[the]NOM[NN[rapids]]]PP[QL[slightly]IN[To]NP[P
PG[its]NOM[NN[rightNn]]]]]NP[PRP[you]]]VP[DO[dont]VP[HV[have]NP[DT[a]NO
M[NN[stonecreek]]]]]]
above the rapids slightly To its rightNn you dont have a stonecreek
Reparandum = S[NP[PRP[you]]VP[DO[dont]VP[HV[have]NIL ]]]
Repair
=S[PP[IN[above]NP[NP[DT[the]NOM[NN[rapids]]]PP[QL[slightly]IN[To]NP[PPG
[its]NOM[NN[rightNn]]]]]NP[PRP[you]]]VP[DO[dont]VP[HV[have]NP[DT[a]NOM[
NN[stonecreek]]]]]]
Matching =S
No Matching =PP
Matching Structure :: Partially Activated S Followed by a Fully
Activated S
OUTPUT ::: Overlapping Parse Trees
:::S[PP[IN[above]NP[NP[DT[the]NOM[NN[rapids]]]PP[QL[slightly]IN[To]NP[P
PG[its]NOM[NN[rightNn]]]]]NP[PRP[you]]]VP[DO[dont]VP[HV[have]NP[DT[a]NO
M[NN[stonecreek]]]]]]
above the rapids slightly To its rightNn you dont have a stonecreek
OUTPUT :::
S[PP[IN[above]NP[NP[DT[the]NOM[NN[rapids]]]PP[QL[slightly]IN[To]NP[PPG[
its]NOM[NN[rightNn]]]]]NP[PRP[you]]]VP[DO[dont]VP[HV[have]NP[DT[a]NOM[N
N[stonecreek]]]]]]

above the rapids slightly To its rightNn you dont have a stonecreek

**********************************************************************

53  at the moment I am at at the moment I am justQl slightly To the
left of the rocks

Reparandum =
S[PP[IN[at]NP[DT[the]NOM[NN[moment]]]NP[PRP[I]]]VP[BEM[am]NIL ]]
Repair =
S[PP[IN[at]NP[DT[the]NOM[NN[moment]]]NP[PRP[I]]]VP[VP[BEM[am]PP[QL[just
Ql]QL[slightly]IN[To]NP[DT[the]NOM[NN[left]]]]]PP[IN[of]NP[DT[the]NOM[N
N[rocks]]]]]]
Matching =S
Matching =PP
Matching =IN
Matching =at
Matching =NP
Matching =DT
Matching =the
Matching =NOM
Matching =NN
Matching =moment
Matching =NP
Matching =PRP
Matching =I
Matching =VP
No Matching =VP
OUTPUT ::: Overlapping Parse Trees
:::S[PP[IN[at]NP[DT[the]NOM[NN[moment]]]NP[PRP[I]]]VP[VP[BEM[am]PP[QL[j
```

```
ustQl]QL[slightly]IN[To]NP[DT[the]NOM[NN[left]]]]]PP[IN[of]NP[DT[the]NO
M[NN[rocks]]]]]]
at the moment I am justQl slightly To the left of the rocks
OUTPUT :::
S[PP[IN[at]NP[DT[the]NOM[NN[moment]]]NP[PRP[I]]]VP[VP[BEM[am]PP[QL[just
Ql]QL[slightly]IN[To]NP[DT[the]NOM[NN[left]]]]]PP[IN[of]NP[DT[the]NOM[N
N[rocks]]]]]]

at the moment I am justQl slightly To the left of the rocks

**********************************************************************

54  go right under the manned ... go right under your manned fort and
between your stonecreek

Previous structure = VP[VBP[go]ADVP[RB[right]]]
Matching Root = VP
Repair =VP[VBP[go]]
Matching =VP
Matching =VBP
Matching =go
No Matching =
OUTPUT ::: Overlapping Parse Trees :::VP[VBP[go]]
go
OUTPUT :::
VP[VP[VBP[go]ADVP[RB[right]]]PP[PP[IN[under]NP[PPG[your]JJ[manned]NOM[N
N[fort]]]]CC[and]PP[IN[between]NP[PPG[your]NOM[NN[stonecreek]]]]]]

go right under your manned fort and between your stonecreek

**********************************************************************

55  the carved wooden pole is right ... at my ... right at the bottom

Reparandum = PP[IN[at]NP[PPG[my]NIL ]]
Repair =PP[IN[at]NP[DT[the]NOM[NN[bottom]]]]
Matching =PP
Matching =IN
Matching =at
Matching =NP
No Matching =DT
OUTPUT ::: Overlapping Parse Trees
:::PP[IN[at]NP[DT[the]NOM[NN[bottom]]]]
at the bottom
OUTPUT :::
S[NP[DT[the]ADJP[JJ[carved]ADJP[JJ[wooden]]]NOM[NN[pole]]]VP[VP[VBP[is]
ADVP[RB[right]]]PP[IN[at]NP[DT[the]NOM[NN[bottom]]]]]]

the carved wooden pole is right at the bottom

**********************************************************************

56  is it okayJJ is it safeenough to head diagnolly acrossRp

Previous structure = VP[VBP[is]NP[PRP[it]]]
Matching Root = VP
Repair =VP[VBP[is]]
```

```
Matching =VP
Matching =VBP
Matching =is
No Matching =
OUTPUT ::: Overlapping Parse Trees :::VP[VBP[is]]
is
OUTPUT :::
VP[VBP[is]NP[PRP[it]]ADJP[JJ[safeenough]]IVP[TO[to]VBP[head]ADVP[RB[dia
gnolly]]RP[acrossRp]]]

is it safeenough to head diagnolly acrossRp

**********************************************************************

57  oh no , these these are in the middle

OUTPUT :::
S[NP[PD[these]]VP[VP[VBP[are]RP[in]]NP[DT[the]NOM[NN[middle]]]]]

these are in the middle

**********************************************************************

58  I have got some flamingoes downIn here with a quite a smallgappy
bit

Reparandum = NP[DT[a]]
Repair =NP[QL[quite]DT[a]JJ[smallgappy]NN[bit]]
Matching =NP
No Matching =QL
Matching Structure :: Partially Activated NP Followed by a Fully
Activated NP
OUTPUT ::: Overlapping Parse Trees
:::NP[QL[quite]DT[a]JJ[smallgappy]NN[bit]]
quite a smallgappy bit
OUTPUT :::
S[NP[PRP[I]]VP[HV[have]VBP[got]NP[DT[some]NOM[NN[flamingoes]]]PP[IN[dow
nIn]NP[PN[here]]]PP[IN[with]NP[QL[quite]DT[a]JJ[smallgappy]NN[bit]]]]]

I have got some flamingoes downIn here with quite a smallgappy bit

**********************************************************************

59  I I havent got anything but I think that is where your flamingoes
are

OUTPUT :::
S[S[S[NP[PRP[I]]VP[HV[havent]VBP[got]NP[PN[anything]]]]CS[but]S[NP[PRP[
I]]VP[VBP[think]]]]CS[that]S[VP[VBP[is]ADVP[WRB[where]NP[PPG[your]NOM[N
N[flamingoes]]]VP[VBP[are]]]]]]

I havent got anything but I think that is where your flamingoes are

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

```
60  but above justQl above it ... and To the rightNn I have got a
stonecreek

Reparandum = PP[IN[above]]
Repair =PP[QL[justQl]IN[above]]
Matching =PP
No Matching =QL
Matching Structure :: Partially Activated PP Followed by a Fully
Activated PP
OUTPUT ::: Overlapping Parse Trees :::PP[QL[justQl]IN[above]]
justQl above
OUTPUT :::
S[PP[PP[QL[justQl]IN[above]NP[PRP[it]]]CC[and]PP[IN[To]NP[DT[the]NOM[NN
[rightNn]]]NP[PRP[I]]]]VP[HV[have]VBP[got]NP[DT[a]NOM[NN[stonecreek]]]]
]

justQl above it and To the rightNn I have got a stonecreek

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

Incorrect Output Structures

```
1  thatPd is  ... thatPd is correct uh-huh , and ... go below the
diamond mine

OUTPUT :::
S[NP[PD[thatPd]]VP[VP[VBP[is]S[NP[PD[thatPd]]VP[VBP[is]ADJP[JJ[correct]
]]]]CC[and]VP[VBP[go]PP[IN[below]NP[DT[the]ADJP[JJ[diamond]]NOM[NN[mine
]]]]]]]

thatPd is thatPd is correct and go below the diamond mine

**********************************************************************

2  go justRB untilCS you go ... go below the diamond mine until justQl
before the fast fast flowing river

Reparandum = PP[IN[until]]
Repair =PP[QL[justQl]IN[before]]
Matching =PP
No Matching =QL
Matching Structure :: Partially Activated PP Followed by a Fully
Activated PP
OUTPUT ::: Overlapping Parse Trees :::PP[QL[justQl]IN[before]]
justQl before
OUTPUT :::
S[S[VP[VBP[go]ADVP[RB[justRB]]]]CS[untilCS]S[NP[PRP[you]]VP[VP[VBP[go]P
P[IN[below]NP[DT[the]ADJP[JJ[diamond]]NOM[NN[mine]]]]]PP[QL[justQl]IN[b
efore]NP[DT[the]ADJP[JJ[fast]ADJP[JJ[flowing]]]NOM[NN[river]]]]]]]

go justRB untilCS you go below the diamond mine justQl before the fast
flowing river
```

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

3  I have a fast flowing running creek , which is on the far righthand
corner

OUTPUT :::
S[NP[PRP[I]]VP[HV[have]NP[DT[a]ADJP[JJ[fast]ADJP[JJ[flowing]ADJP[JJ[run
ning]]]]NOM[NOM[NN[creek]]RlCl[PR[which]VP[VBP[is]PP[IN[on]NP[DT[the]AD
JP[JJ[far]ADJP[JJ[righthand]]]NOM[NN[corner]]]]]]]]]]

I have a fast flowing running creek which is on the far righthand
corner

**************************************************************************

4  above the rightQl below the buffalo below the buffalo Right

Reparandum = PP[IN[above]NP[DT[the]NIL ]]
Repair =PP[QL[rightQl]IN[below]]
Matching =PP
No Matching =QL
Matching Structure :: Partially Activated PP Followed by a Fully
Activated PP
OUTPUT ::: Overlapping Parse Trees :::PP[QL[rightQl]IN[below]]
rightQl below
OUTPUT :::
PP[QL[rightQl]IN[below]NP[NP[DT[the]NOM[NN[buffalo]]]PP[IN[below]NP[DT[
the]NOM[NN[buffalo]]]]]]

rightQl below the buffalo below the buffalo

**************************************************************************

5  so you need to be ... go upRp ... on a sortof hill

Previous structure = VP[VBP[need]]
Matching Root = VP
Repair =VP[VBP[go]]
Matching =VP
Matching =VBP
No Matching =go
OUTPUT ::: Overlapping Parse Trees :::VP[VBP[go]]
go
OUTPUT :::
S[NP[PRP[you]]VP[VP[VBP[go]PP[RP[upRp]IN[on]]]NP[DT[a]ADJP[JJ[sortof]]N
OM[NN[hill]]]]]

you go upRp on a sortof hill

**************************************************************************

6  so ... Right so , I would I would have to draw a curve then from the
down from the caravanpark

Reparandum = S[NP[PRP[I]]]
Repair
=S[MOD[would]NP[PRP[I]]VP[MOD[would]VP[HV[have]IVP[TO[to]VBP[draw]]]]]
```

```
Matching =S
No Matching =MOD
Matching Structure :: Partially Activated S Followed by a Fully
Activated S
OUTPUT ::: Overlapping Parse Trees
:::S[MOD[would]NP[PRP[I]]VP[MOD[would]VP[HV[have]IVP[TO[to]VBP[draw]]]]
]
would I would have to draw
Reparandum = PP[IN[from]NP[DT[the]NIL ]]
Repair =PP[RP[down]IN[from]]
Matching =PP
No Matching =RP
Matching Structure :: Partially Activated PP Followed by a Fully
Activated PP
OUTPUT ::: Overlapping Parse Trees :::PP[RP[down]IN[from]]
down from
OUTPUT :::
S[MOD[would]NP[PRP[I]]VP[VP[VP[VP[MOD[would]VP[HV[have]IVP[TO[to]VBP[dr
aw]NP[DT[a]NOM[NN[curve]]]]]]ADVP[RB[then]]]PP[RP[down]IN[from]]]NP[DT[
the]NOM[NN[caravanpark]]]]]

would I would have to draw a curve then down from the caravanpark

**********************************************************************

7  you know how the ... the twobits of the crane come in ... cranebay
come in likeIn thatPd ... then into the river
OUTPUT
::::::S[PP[IN[of]NP[DT[the]NOM[NN[crane]]]]VP[VP[VP[VP[VBP[come]PP[RP[i
n]IN[likeIn]]]NP[PD[thatPd]]]ADVP[RB[then]]]PP[IN[into]NP[DT[the]NOM[NN
[river]]]]]]
of the crane come in likeIn thatPd then into the river
OUTPUT :::
S[PP[IN[of]NP[DT[the]NOM[NN[crane]]]]VP[VP[VP[VP[VBP[come]PP[RP[in]IN[l
ikeIn]]]NP[PD[thatPd]]]ADVP[RB[then]]]PP[IN[into]NP[DT[the]NOM[NN[river
]]]]]]

of the crane come in likeIn thatPd then into the river

**********************************************************************

8  wellAff , go upRp to where ... the twobits join nearly ... and to
form the lar ... the river

Previous structure = S[VP[VBP[go]RP[upRp]]]
Matching Root = S
Repair =S[NP[ADVP[WRB[where]]NP[DT[the]NOM[NN[twobits]]]]VP[VBP[join]]]
Matching =S
No Matching =NP
Reparandum = S[S[VP[VBP[go]RP[upRp]]]]
Repair =S[NP[ADVP[WRB[where]]NP[DT[the]NOM[NN[twobits]]]]VP[VBP[join]]]
Matching =S
No Matching =NP
Matching Structure :: Partially Activated S Followed by a Fully
Activated S
OUTPUT ::: Overlapping Parse Trees
:::S[NP[ADVP[WRB[where]]NP[DT[the]NOM[NN[twobits]]]]VP[VBP[join]]]
```

```
where the twobits join
Reparandum = NP[DT[the]]
Repair =NP[DT[the]NOM[NN[river]]]
Matching =NP
Matching =DT
Matching =the
OUTPUT ::: Overlapping Parse Trees :::NP[DT[the]NOM[NN[river]]]
the river
OUTPUT :::
IVP[TO[to]VBP[form]NP[DT[the]NOM[NN[river]]]]
to form the river
OUTPUT :::
CC[and]
and
OUTPUT :::
ADVP[WRB[where]NP[DT[the]NOM[NN[twobits]]]VP[VBP[join]ADVP[RB[nearly]]]
]
where the twobits join nearly
OUTPUT :::
TO[to]
to
OUTPUT :::
VP[VBP[go]RP[upRp]]

go upRp

**************************************************************************

9  Right , now I want you to ... go To your ... left ... and do a
sortof c quite a ... a wide c ... roundJJ some supposedVb

Reparandum =
VP[VP[VP[VBP[want]NP[PRP[you]]IVP[TO[to]VBP[go]PP[IN[To]NP[PPG[your]NOM
[NN[left]]]]]]CC[and]VP[DO[do]NP[DT[a]ADJP[JJ[sortof]]NOM[NN[c]]]NP[QL[
quite]DT[a]JJ[wide]NN[c]]]]NP[ADJP[JJ[roundJJ]]NIL ]]
Repair =VP[VBP[supposedVb]]
Matching =VP
No Matching =VBP
Matching Structure :: Partially Activated VP Followed by a Fully
Activated VP
OUTPUT ::: Overlapping Parse Trees :::VP[VBP[supposedVb]]
supposedVb
OUTPUT :::
S[NP[PRP[I]]VP[VBP[supposedVb]]]

I supposedVb

**************************************************************************

10  there are ... the the hills are on the lefthand side ... my
lefthand side ... which is your lefthand side

OUTPUT :::
S[NP[EX[there]]VP[VBP[are]S[NP[DT[the]NOM[NN[hills]]]VP[VBP[are]PP[IN[o
n]NP[DT[the]ADJP[JJ[lefthand]]NOM[NN[side]]]NP[PPG[my]JJ[lefthand]NOM[N
OM[NN[side]]RlCl[PR[which]VP[VBP[is]NP[PPG[your]JJ[lefthand]NOM[NN[side
]]]]]]]]]]]]
```

```
there are the hills are on the lefthand side my lefthand side which is
your lefthand side

**************************************************************************

11  so at the top of ... justQl before the fork inIn the river ... do a
c ... To the ... the same side

Reparandum = PP[IN[of]]
Repair =PP[QL[justQl]IN[before]]
Matching =PP
No Matching =QL
Matching Structure :: Partially Activated PP Followed by a Fully
Activated PP
OUTPUT ::: Overlapping Parse Trees :::PP[QL[justQl]IN[before]]
justQl before
OUTPUT :::
S[PP[IN[at]NP[NP[NP[DT[the]NOM[NN[top]]]PP[QL[justQl]IN[before]NP[DT[th
e]NOM[NN[fork]]]]]PP[IN[inIn]NP[DT[the]NOM[NN[river]]]]]]S[DO[do]NP[NP[
DT[a]NOM[NN[c]]]PP[IN[To]NP[DT[the]ADJP[JJ[same]]NOM[NN[side]]]]]]]

at the top justQl before the fork inIn the river do a c To the same
side

**************************************************************************

12  erm ... we are start ... starting ... down , thatPd is the end of
thatPd , erm starting above the telephonekiosk

OUTPUT ::: Overlapping Parse Trees
:::S[PP[IN[of]NP[PD[thatPd]]]VP[VBP[starting]PP[IN[above]NP[DT[the]NOM[
NN[telephonekiosk]]]]]]
of thatPd starting above the telephonekiosk
OUTPUT :::
S[PP[IN[of]NP[PD[thatPd]]]VP[VBP[starting]PP[IN[above]NP[DT[the]NOM[NN[
telephonekiosk]]]]]]

of thatPd starting above the telephonekiosk

**************************************************************************

13  erm ... if you come just loop down ... below the stile and round To
the lefthand side

Reparandum = PP[IN[round]]
Repair =PP[IN[To]NP[DT[the]ADJP[JJ[lefthand]]NOM[NN[side]]]]
Matching =PP
Matching =IN
No Matching =To
OUTPUT ::: Overlapping Parse Trees
:::PP[IN[To]NP[DT[the]ADJP[JJ[lefthand]]NOM[NN[side]]]]
To the lefthand side
Reparandum = NP[DT[the]NN[stile]CC[and]]
Repair =PP[IN[To]NP[DT[the]ADJP[JJ[lefthand]]NOM[NN[side]]]]
No Matching =PP
```

```
Matching Structure :: Partially Activated NP Followed by a Fully
Activated NP
OUTPUT ::: Overlapping Parse Trees
:::PP[IN[To]NP[DT[the]ADJP[JJ[lefthand]]NOM[NN[side]]]]
To the lefthand side
Reparandum = PP[PP[RP[down]IN[below]]]
Repair = PP[IN[To]NP[DT[the]ADJP[JJ[lefthand]]NOM[NN[side]]]]
Matching =PP
No Matching =IN
Matching Structure :: Partially Activated PP Followed by a Fully
Activated PP
OUTPUT ::: Overlapping Parse Trees
:::PP[IN[To]NP[DT[the]ADJP[JJ[lefthand]]NOM[NN[side]]]]
To the lefthand side
OUTPUT :::
S[NP[PRP[you]]VP[VBP[come]S[VP[ADVP[RB[just]]VP[VBP[loop]PP[IN[To]NP[DT
[the]ADJP[JJ[lefthand]]NOM[NN[side]]]]]]]]]
you come just loop To the lefthand side
OUTPUT :::
CS[if]
if

**********************************************************************

14  I have got , oh , I have got a great viewpoint ... eh ... ... above
the tree ... eh justQl below the eastlake

Reparandum = VP[HV[have]]
Repair =
VP[VBP[got]S[NP[PRP[I]]VP[HV[have]VBP[got]NP[DT[a]ADJP[JJ[great]]NOM[NN
[viewpoint]]]PP[IN[above]NP[DT[the]NOM[NN[tree]]]]PP[QL[justQl]IN[below
]NP[DT[the]NOM[NN[eastlake]]]]]]]
Matching =VP
No Matching =VBP
Matching Structure :: Partially Activated VP Followed by a Fully
Activated VP
OUTPUT ::: Overlapping Parse Trees
:::VP[VBP[got]S[NP[PRP[I]]VP[HV[have]VBP[got]NP[DT[a]ADJP[JJ[great]]NOM
[NN[viewpoint]]]PP[IN[above]NP[DT[the]NOM[NN[tree]]]]PP[QL[justQl]IN[be
low]NP[DT[the]NOM[NN[eastlake]]]]]]]
got I have got a great viewpoint above the tree justQl below the
eastlake
OUTPUT :::
S[NP[PRP[I]]VP[VBP[got]S[NP[PRP[I]]VP[HV[have]VBP[got]NP[DT[a]ADJP[JJ[g
reat]]NOM[NN[viewpoint]]]PP[IN[above]NP[DT[the]NOM[NN[tree]]]]PP[QL[jus
tQl]IN[below]NP[DT[the]NOM[NN[eastlake]]]]]]]]

I got I have got a great viewpoint above the tree justQl below the
eastlake

**********************************************************************
15  Right , so I am I am I am like on the bank on the bank of the
eastlake

Reparandum = S[NP[PRP[I]]VP[BEM[am]NIL ]]
Repair =S[NP[PRP[I]]VP[BEM[am]PP[RP[like]IN[on]]]]
Matching =S
```

```
Matching =NP
Matching =PRP
Matching =I
Matching =VP
Matching =BEM
Matching =am
No Matching =PP
OUTPUT ::: Overlapping Parse Trees
:::S[NP[PRP[I]]VP[BEM[am]PP[RP[like]IN[on]]]]
I am like on
Reparandum = S[NP[PRP[I]]VP[BEM[am]NIL ]]
Repair =S[NP[PRP[I]]VP[BEM[am]PP[RP[like]IN[on]]]]
Matching =S
Matching =NP
Matching =PRP
Matching =I
Matching =VP
Matching =BEM
Matching =am
No Matching =PP
OUTPUT ::: Overlapping Parse Trees
:::S[NP[PRP[I]]VP[BEM[am]PP[RP[like]IN[on]]]]
I am like on
OUTPUT :::
S[NP[PRP[I]]VP[VP[VP[VP[BEM[am]PP[RP[like]IN[on]]]NP[DT[the]NOM[NN[bank
]]]]PP[IN[on]NP[DT[the]NOM[NN[bank]]]]]PP[IN[of]NP[DT[the]NOM[NN[eastla
ke]]]]]]

I am like on the bank on the bank of the eastlake

**********************************************************************

16  no , I have got I have got an eastlake but I havent got a westlake

Reparandum = S[NP[PRP[I]]VP[HV[have]NIL ]]
Repair =
S[S[VP[VBP[got]S[NP[PRP[I]]VP[HV[have]VBP[got]NP[DT[an]NOM[NN[eastlake]
]]]]]]CS[but]S[NP[PRP[I]]VP[HV[havent]VBP[got]NP[DT[a]NOM[NN[westlake]]
]]]]
Matching =S
No Matching =S
Matching Structure :: Partially Activated S Followed by a Fully
Activated S
OUTPUT ::: Overlapping Parse Trees
:::S[S[VP[VBP[got]S[NP[PRP[I]]VP[HV[have]VBP[got]NP[DT[an]NOM[NN[eastla
ke]]]]]]]CS[but]S[NP[PRP[I]]VP[HV[havent]VBP[got]NP[DT[a]NOM[NN[westlak
e]]]]]]
got I have got an eastlake but I havent got a westlake
OUTPUT :::
S[S[VP[VBP[got]S[NP[PRP[I]]VP[HV[have]VBP[got]NP[DT[an]NOM[NN[eastlake]
]]]]]]CS[but]S[NP[PRP[I]]VP[HV[havent]VBP[got]NP[DT[a]NOM[NN[westlake]]
]]]]

got I have got an eastlake but I havent got a westlake

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

```
17  it is it is directly below thatPd

OUTPUT :::
S[NP[PRP[it]]VP[VBP[is]S[NP[PRP[it]]VP[VBP[is]PP[QL[directly]IN[below]N
P[PD[thatPd]]]]]]]

it is it is directly below thatPd

**************************************************************************

18  and come over gradually To the far To the lefthand side of the tree

Reparandum = PP[IN[To]NP[DT[the]NIL ]]
Repair =
ADJP[JJ[far]PP[IN[To]NP[NP[DT[the]ADJP[JJ[lefthand]]NOM[NN[side]]]PP[IN
[of]NP[DT[the]NOM[NN[tree]]]]]]]
No Matching =ADJP
Matching Structure :: Partially Activated PP Followed by a Fully
Activated PP
OUTPUT ::: Overlapping Parse Trees
:::ADJP[JJ[far]PP[IN[To]NP[NP[DT[the]ADJP[JJ[lefthand]]NOM[NN[side]]]PP
[IN[of]NP[DT[the]NOM[NN[tree]]]]]]]
far To the lefthand side of the tree
OUTPUT :::
ADJP[JJ[far]PP[IN[To]NP[NP[DT[the]ADJP[JJ[lefthand]]NOM[NN[side]]]PP[IN
[of]NP[DT[the]NOM[NN[tree]]]]]]]
far To the lefthand side of the tree
OUTPUT :::
VP[VP[VBP[come]RP[over]]ADVP[RB[gradually]]]

come over gradually

**************************************************************************

19  then you need to go straightQl along for aboutQl untilCS aboutQl
... you are about an inch and a half away from the edge of the map

Reparandum = NP[QLDT[aboutQl]]
Repair =NP[PRP[you]]
Matching =NP
No Matching =PRP
Matching Structure :: Partially Activated NP Followed by a Fully
Activated NP
OUTPUT ::: Overlapping Parse Trees :::NP[PRP[you]]
you
Reparandum = NP[QLDT[aboutQl]]
Repair =NP[PRP[you]]
Matching =NP
No Matching =PRP
Matching Structure :: Partially Activated NP Followed by a Fully
Activated NP
OUTPUT ::: Overlapping Parse Trees :::NP[PRP[you]]
you
Previous structure = VP[VBP[need]]
Matching Root = VP
Repair =
VP[VP[VP[VP[VP[VBP[go]S[PP[QL[straightQl]RP[along]PP[IN[for]NP[PRP[you]
```

```
]]]VP[VBP[are]RP[about]]]]NP[NP[DT[an]NOM[NN[inch]]]CC[and]NP[DT[a]NOM[
NN[half]]]]]PP[RP[away]IN[from]]]NP[DT[the]NOM[NN[edge]]]]PP[IN[of]NP[D
T[the]NOM[NN[map]]]]]
Matching =VP
No Matching =VP
OUTPUT :::
VP[VP[VP[VP[VP[VBP[go]S[PP[QL[straightQl]RP[along]PP[IN[for]NP[PRP[you]
]]]VP[VBP[are]RP[about]]]]NP[NP[DT[an]NOM[NN[inch]]]CC[and]NP[DT[a]NOM[
NN[half]]]]]PP[RP[away]IN[from]]]NP[DT[the]NOM[NN[edge]]]]PP[IN[of]NP[D
T[the]NOM[NN[map]]]]]
go straightQl along for you are about an inch and a half away from the
edge of the map
OUTPUT :::
TO[to]
to
OUTPUT :::
S[NP[PRP[you]]VP[VBP[need]]]
you need

**********************************************************************

20  yeah , so you are at where are you now

Previous structure = VP[VBP[are]]
Matching Root = VP
Repair =VP[ADVP[WRB[where]]VP[VBP[are]]]
Matching =VP
No Matching =ADVP
Previous structure = VP[VBP[are]]
Matching Root = VP
Repair =VP[ADVP[WRB[where]]VP[VBP[are]]]
Matching =VP
No Matching =ADVP
Previous structure = S[NP[PRP[you]]VP[VBP[are]]]
Matching Root = S
Repair =S[VP[ADVP[WRB[where]]VP[VBP[are]]]]
Matching =S
No Matching =VP
Reparandum = S[S[NP[PRP[you]]VP[VBP[are]]]]
Repair =S[VP[ADVP[WRB[where]]VP[VBP[are]]]]
Matching =S
No Matching =VP
Matching Structure :: Partially Activated S Followed by a Fully
Activated S
OUTPUT ::: Overlapping Parse Trees
:::S[VP[ADVP[WRB[where]]VP[VBP[are]]]]
where are
Previous structure = VP[VBP[are]]
Matching Root = VP
Repair =VP[VP[ADVP[WRB[where]]VP[VBP[are]]]NP[PRP[you]]]
Matching =VP
No Matching =VP
Previous structure = S[NP[PRP[you]]VP[VBP[are]]]
Matching Root = S
Repair =S[VP[VP[ADVP[WRB[where]]VP[VBP[are]]]NP[PRP[you]]]]
Matching =S
No Matching =VP
```

```
Reparandum = S[S[NP[PRP[you]]VP[VBP[are]]]]
Repair =S[VP[VP[ADVP[WRB[where]]VP[VBP[are]]]NP[PRP[you]]]]
Matching =S
No Matching =VP
Matching Structure :: Partially Activated S Followed by a Fully
Activated S
OUTPUT ::: Overlapping Parse Trees
:::S[VP[VP[ADVP[WRB[where]]VP[VBP[are]]]NP[PRP[you]]]]
where are you
Previous structure = VP[VBP[are]]
Matching Root = VP
Repair =3
Matching =VP
No Matching =VP
OUTPUT :::
VP[VP[ADVP[WRB[where]]VBP[are]NP[PRP[you]]]ADVP[RB[now]]]
where are you now
OUTPUT :::
IN[at]
at
OUTPUT :::
S[NP[PRP[you]]VP[VBP[are]]]
you are

**********************************************************************

21  so you are justQl ... ... you are now aboutQl twoinches above the
gorillas or the banana tree

Reparandum = NP[QL[justQl]]
Repair =NP[PRP[you]]
Matching =NP
No Matching =PRP
Matching Structure :: Partially Activated NP Followed by a Fully
Activated NP
OUTPUT ::: Overlapping Parse Trees :::NP[PRP[you]]
you
OUTPUT :::
S[NP[PRP[you]]VP[VP[VP[VBP[are]S[NP[PRP[you]]VP[VBP[are]ADVP[RB[now]]]]
]NP[QLDT[aboutQl]NOM[NN[twoinches]]]]PP[IN[above]NP[NP[DT[the]NOM[NN[go
rillas]]]CC[or]NP[DT[the]ADJP[JJ[banana]]NOM[NN[tree]]]]]]]

you are you are now aboutQl twoinches above the gorillas or the banana
tree

**********************************************************************

22  Right okay , To the do I bend it rightQl over untilCS I get To the
fallen pillars

Reparandum = NP[DT[the]]
Repair =S[DO[do]NP[PRP[I]]]
No Matching =S
Matching Structure :: Partially Activated NP Followed by a Fully
Activated NP
OUTPUT ::: Overlapping Parse Trees :::S[DO[do]NP[PRP[I]]]
do I
```

```
Reparandum = S[DO[do]]
Repair =
S[S[NP[PRP[I]]VP[VBP[bend]NP[PRP[it]]PP[QL[rightQl]RP[over]]]]CS[untilC
S]S[NP[PRP[I]]VP[VBP[get]PP[IN[To]NP[DT[the]ADJP[JJ[fallen]]NOM[NN[pill
ars]]]]]]]
Matching =S
No Matching =S
Matching Structure :: Partially Activated S Followed by a Fully
Activated S
OUTPUT ::: Overlapping Parse Trees
:::S[S[NP[PRP[I]]VP[VBP[bend]NP[PRP[it]]PP[QL[rightQl]RP[over]]]]CS[unt
ilCS]S[NP[PRP[I]]VP[VBP[get]PP[IN[To]NP[DT[the]ADJP[JJ[fallen]]NOM[NN[p
illars]]]]]]]
I bend it rightQl over untilCS I get To the fallen pillars
OUTPUT :::
S[S[NP[PRP[I]]VP[VBP[bend]NP[PRP[it]]PP[QL[rightQl]RP[over]]]]CS[untilC
S]S[NP[PRP[I]]VP[VBP[get]PP[IN[To]NP[DT[the]ADJP[JJ[fallen]]NOM[NN[pill
ars]]]]]]]
I bend it rightQl over untilCS I get To the fallen pillars
OUTPUT :::
IN[To]
To

**********************************************************************

23  Right , you are now underneath ... are you at the end of the fallen
pillars

Previous structure = VP[VBP[are]ADVP[RB[now]]]
Matching Root = VP
Repair =VP[VBP[are]]
Matching =VP
Matching =VBP
Matching =are
No Matching =
OUTPUT ::: Overlapping Parse Trees :::VP[VBP[are]]
are
OUTPUT :::
S[NP[PRP[you]]VP[VP[VBP[are]NP[PRP[you]]PP[IN[at]NP[DT[the]NOM[NN[end]]
]]]PP[IN[of]NP[DT[the]ADJP[JJ[fallen]]NOM[NN[pillars]]]]]]

you are you at the end of the fallen pillars

**********************************************************************

24  so you need to go ... loop it down ... and then all the way up To
the fallen pillars

Previous structure = VP[VBP[need]IVP[TO[to]VBP[go]]]
Matching Root = VP
Repair =VP[VBP[loop]]
Matching =VP
Matching =VBP
No Matching =loop
OUTPUT ::: Overlapping Parse Trees :::VP[VBP[loop]]
loop
Previous structure = NP[PRP[it]]
```

```
Matching Root = NP
Repair =NP[ADVP[RB[then]]NP[DPR[all]DT[the]NN[way]]]
Matching =NP
No Matching =ADVP
Reparandum = NP[NP[PRP[it]]]
Repair =NP[ADVP[RB[then]]NP[DPR[all]DT[the]NN[way]]]
Matching =NP
No Matching =ADVP
Matching Structure :: Partially Activated NP Followed by a Fully
Activated NP
OUTPUT ::: Overlapping Parse Trees
:::NP[ADVP[RB[then]]NP[DPR[all]DT[the]NN[way]]]
then all the way
Reparandum = PP[IN[up]]
Repair =PP[IN[To]NP[DT[the]ADJP[JJ[fallen]]NOM[NN[pillars]]]]
Matching =PP
Matching =IN
No Matching =To
OUTPUT ::: Overlapping Parse Trees
:::PP[IN[To]NP[DT[the]ADJP[JJ[fallen]]NOM[NN[pillars]]]]
To the fallen pillars
OUTPUT :::
S[NP[PRP[you]]VP[VBP[loop]NP[ADVP[RB[then]]NP[DPR[all]DT[the]NN[way]]]P
P[IN[To]NP[DT[the]ADJP[JJ[fallen]]NOM[NN[pillars]]]]]]

you loop then all the way To the fallen pillars

**********************************************************************

25  okay , then  , so you are at you are at the stones now

OUTPUT :::
S[NP[PRP[you]]VP[VP[VBP[are]S[PP[IN[at]NP[PRP[you]]]VP[VBP[are]PP[IN[at
]NP[DT[the]NOM[NN[stones]]]]]]]ADVP[RB[now]]]]

you are at you are at the stones now

**********************************************************************


26  you areAx going to go ... and you areAx going to come rightQl
through

OUTPUT :::
S[S[NP[PRP[you]]VP[AUX[areAx]VBP[going]IVP[TO[to]VBP[go]]]]CC[and]S[NP[
PRP[you]]VP[AUX[areAx]VBP[going]IVP[TO[to]VBP[come]PP[QL[rightQl]IN[thr
ough]]]]]]

you areAx going to go and you areAx going to come rightQl through

**********************************************************************

27  okay ... Right , I will have to I will I will have to kindof change
it slightly because I have taken it on the wrong side of the cactus
thing

Reparandum = S[NP[PRP[I]]]
```

```
Repair
=S[MOD[will]NP[PRP[I]]VP[MOD[will]VP[HV[have]IVP[TO[to]RB[kindof]VBP[ch
ange]NP[PRP[it]]]]]]
Matching =S
No Matching =MOD
Matching Structure :: Partially Activated S Followed by a Fully
Activated S
OUTPUT ::: Overlapping Parse Trees
:::S[MOD[will]NP[PRP[I]]VP[MOD[will]VP[HV[have]IVP[TO[to]RB[kindof]VBP[
change]NP[PRP[it]]]]]]
will I will have to kindof change it
Reparandum = S[NP[PRP[I]]VP[MOD[will]VP[HV[have]IVP[TO[to]NIL ]]]]
Repair
=S[MOD[will]NP[PRP[I]]VP[MOD[will]VP[HV[have]IVP[TO[to]RB[kindof]VBP[ch
ange]NP[PRP[it]]]]]]
Matching =S
No Matching =MOD
Matching Structure :: Partially Activated S Followed by a Fully
Activated S
OUTPUT ::: Overlapping Parse Trees
:::S[MOD[will]NP[PRP[I]]VP[MOD[will]VP[HV[have]IVP[TO[to]RB[kindof]VBP[
change]NP[PRP[it]]]]]]
will I will have to kindof change it
OUTPUT :::
S[S[MOD[will]NP[PRP[I]]VP[MOD[will]VP[HV[have]IVP[TO[to]RB[kindof]VBP[c
hange]NP[PRP[it]]]]]]CS[QL[slightly]CS[because]]S[NP[PRP[I]]VP[HV[have]
VBP[taken]NP[PRP[it]]PP[IN[on]NP[DT[the]ADJP[JJ[wrong]]NOM[NN[side]]]]P
P[IN[of]NP[DT[the]ADJP[JJ[cactus]]NOM[NN[thing]]]]]]]

will I will have to kindof change it slightly because I have taken it
on the wrong side of the cactus thing

**********************************************************************

28  okay wellAff , you need to go ... ehm ... ... travel leftRp aboutQL
an inch ... and then go down alongIn the side of the missionary camp

Previous structure = VP[VBP[need]IVP[TO[to]VBP[go]]]
Matching Root = VP
Repair =VP[VBP[travel]]
Matching =VP
Matching =VBP
No Matching =travel
OUTPUT ::: Overlapping Parse Trees :::VP[VBP[travel]]
travel
OUTPUT :::
S[NP[PRP[you]]VP[VP[VP[VP[VP[VBP[travel]RP[leftRp]]NP[QL[aboutQL]DT[an]
NN[inch]]]CC[and]VP[ADVP[RB[then]]VP[VBP[go]PP[RP[down]IN[alongIn]]]]]]N
P[DT[the]NOM[NN[side]]]]PP[IN[of]NP[DT[the]ADJP[JJ[missionary]]NOM[NN[c
amp]]]]]]

you travel leftRp aboutQL an inch and then go down alongIn the side of
the missionary camp

**********************************************************************

29  okay , now , you have you got a some gorillas over on the left
```

```
Reparandum = NP[DT[a]]
Repair =NP[DT[some]NOM[NN[gorillas]]]
Matching =NP
Matching =DT
No Matching =some
OUTPUT ::: Overlapping Parse Trees :::NP[DT[some]NOM[NN[gorillas]]]
some gorillas
OUTPUT :::
S[NP[PRP[you]]VP[VP[HV[have]NP[PRP[you]]VP[VBP[got]NP[DT[some]NOM[NN[go
rillas]]]PP[RP[over]IN[on]]]]NP[DT[the]NOM[NN[left]]]]]

you have you got some gorillas over on the left

**************************************************************************

30  alright thatPd is probably , do you think thatPd is probably the
sameNn

Previous structure = S[NP[PD[thatPd]]VP[VBP[is]ADVP[RB[probably]]]]
Matching Root = S
Repair =S[DO[do]NP[PRP[you]]]
Matching =S
No Matching =DO
Reparandum = S[S[NP[PD[thatPd]]VP[VBP[is]ADVP[RB[probably]]]]]
Repair =S[DO[do]NP[PRP[you]]]
Matching =S
No Matching =DO
Matching Structure :: Partially Activated S Followed by a Fully
Activated S
OUTPUT ::: Overlapping Parse Trees :::S[DO[do]NP[PRP[you]]]
do you
OUTPUT :::
S[NP[PD[thatPd]]VP[VP[VBP[is]S[VP[ADVP[RB[probably]]VP[DO[do]NP[PRP[you
]]VP[VBP[think]S[NP[PD[thatPd]]VP[VBP[is]ADVP[RB[probably]]]]]]]]]NP[DT
[the]NOM[NN[sameNn]]]]]

thatPd is probably do you think thatPd is probably the sameNn

**************************************************************************

31  yeah , wellAff actually you need to then you need to bend upRp
round the side of the gorillas so you are then going along so you areAx
runningVb parallel with the eh lefthand side of the map

Reparandum = S[S[NP[ADVP[RB[actually]]NP[PRP[you]]]VP[VBP[need]]]]
Repair =
S[S[NP[ADVP[RB[then]]NP[PRP[you]]]VP[VP[VP[VBP[need]IVP[TO[to]VBP[bend]
PP[RP[upRp]IN[round]]]]NP[DT[the]NOM[NN[side]]]]PP[IN[of]NP[DT[the]NOM[
NN[gorillas]]]]]]CS[so]S[NP[PRP[you]]VP[VP[VBP[are]S[S[VP[ADVP[RB[then]
]VP[VBP[going]RP[along]]]]CS[so]S[NP[PRP[you]]VP[AUX[areAx]VP[VBP[runni
ngVb]ADJP[JJ[parallel]PP[IN[with]NP[DT[the]ADJP[JJ[lefthand]]NOM[NN[sid
e]]]]]]]]]]PP[IN[of]NP[DT[the]NOM[NN[map]]]]]]]
Matching =S
Matching =S
Matching =NP
Matching =ADVP
```

```
Matching =RB
No Matching =then
OUTPUT ::: Overlapping Parse Trees
:::S[S[NP[ADVP[RB[then]]NP[PRP[you]]]VP[VP[VP[VBP[need]IVP[TO[to]VBP[be
nd]PP[RP[upRp]IN[round]]]]NP[DT[the]NOM[NN[side]]]]PP[IN[of]NP[DT[the]N
OM[NN[gorillas]]]]]]CS[so]S[NP[PRP[you]]VP[VP[VBP[are]S[S[VP[ADVP[RB[th
en]]VP[VBP[going]RP[along]]]]CS[so]S[NP[PRP[you]]VP[AUX[areAx]VP[VBP[ru
nningVb]ADJP[JJ[parallel]PP[IN[with]NP[DT[the]ADJP[JJ[lefthand]]NOM[NN[
side]]]]]]]]]]PP[IN[of]NP[DT[the]NOM[NN[map]]]]]]]
then you need to bend upRp round the side of the gorillas so you are
then going along so you areAx runningVb parallel with the lefthand side
of the map
OUTPUT :::
S[S[NP[ADVP[RB[then]]NP[PRP[you]]]VP[VP[VP[VBP[need]IVP[TO[to]VBP[bend]
PP[RP[upRp]IN[round]]]]NP[DT[the]NOM[NN[side]]]]PP[IN[of]NP[DT[the]NOM[
NN[gorillas]]]]]]CS[so]S[NP[PRP[you]]VP[VP[VBP[are]S[S[VP[ADVP[RB[then]
]VP[VBP[going]RP[along]]]]CS[so]S[NP[PRP[you]]VP[AUX[areAx]VP[VBP[runni
ngVb]ADJP[JJ[parallel]PP[IN[with]NP[DT[the]ADJP[JJ[lefthand]]NOM[NN[sid
e]]]]]]]]]]PP[IN[of]NP[DT[the]NOM[NN[map]]]]]]]

then you need to bend upRp round the side of the gorillas so you are
then going along so you areAx runningVb parallel with the lefthand side
of the map

**********

32  okay , downIn aboutQL an inch ... so you come you have come off the
rope bridge and you drop downIn aboutQL an inch

Previous structure = NP[QL[aboutQL]DT[an]NN[inch]]
Matching Root = NP
Repair =NP[PRP[you]]
Matching =NP
No Matching =PRP
Reparandum = NP[DT[an]NN[inch]]
Repair =NP[PRP[you]]
Matching =NP
No Matching =PRP
Matching Structure :: Partially Activated NP Followed by a Fully
Activated NP
OUTPUT ::: Overlapping Parse Trees :::NP[PRP[you]]
you
OUTPUT :::
S[S[NP[PRP[you]]VP[VBP[come]S[NP[PRP[you]]VP[HV[have]VBP[come]PP[IN[off
]NP[DT[the]ADJP[JJ[rope]]NOM[NN[bridge]]]]]]]CC[and]S[NP[PRP[you]]VP[V
BP[drop]PP[IN[downIn]NP[QL[aboutQL]DT[an]NN[inch]]]]]]
you come you have come off the rope bridge and you drop downIn aboutQL
an inch
OUTPUT :::
QL[aboutQL]
aboutQL
OUTPUT :::
IN[downIn]
downIn

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

```
33  you have got where are you now

Reparandum = VP[HV[have]]
Repair =
VP[VP[VBP[got]S[VP[ADVP[WRB[where]]VBP[are]NP[PRP[you]]]]]ADVP[RB[now]]
]
Matching =VP
No Matching =VP
Matching Structure :: Partially Activated VP Followed by a Fully
Activated VP
OUTPUT ::: Overlapping Parse Trees
:::VP[VP[VBP[got]S[VP[ADVP[WRB[where]]VBP[are]NP[PRP[you]]]]]ADVP[RB[no
w]]]
got where are you now
OUTPUT :::
S[NP[PRP[you]]VP[VP[VBP[got]S[VP[ADVP[WRB[where]]VBP[are]NP[PRP[you]]]]
]ADVP[RB[now]]]]

you got where are you now

**************************************************************************

34  so you want to come down ... straightQl down ... ... untilCS you
are aboutQl ... twoinches

OUTPUT :::
S[S[NP[PRP[you]]VP[VBP[want]IVP[TO[to]VBP[come]RP[down]PP[QL[straightQl
]RP[down]]]]]CS[untilCS]S[NP[PRP[you]]VP[VBP[are]NP[QLDT[aboutQl]NOM[NN
[twoinches]]]]]]

you want to come down straightQl down untilCS you are aboutQl twoinches

**************************************************************************

35  I have got  ... from the bottom I have got saloonbar stonecreek
mannedfort

Reparandum = VP[HV[have]]
Repair =
VP[VBP[got]S[PP[IN[from]NP[DT[the]NOM[NN[bottom]]]NP[PRP[I]]]VP[HV[have
]VBP[got]NOM[NN[saloonbar]NOM[NN[stonecreek]NOM[NN[mannedfort]]]]]]]
Matching =VP
No Matching =VBP
Matching Structure :: Partially Activated VP Followed by a Fully
Activated VP
OUTPUT ::: Overlapping Parse Trees
:::VP[VBP[got]S[PP[IN[from]NP[DT[the]NOM[NN[bottom]]]NP[PRP[I]]]VP[HV[h
ave]VBP[got]NOM[NN[saloonbar]NOM[NN[stonecreek]NOM[NN[mannedfort]]]]]]]
got from the bottom I have got saloonbar stonecreek mannedfort

OUTPUT :::
S[NP[PRP[I]]VP[VBP[got]S[PP[IN[from]NP[DT[the]NOM[NN[bottom]]]NP[PRP[I]
]]VP[HV[have]VBP[got]NOM[NN[saloonbar]NOM[NN[stonecreek]NOM[NN[mannedfo
rt]]]]]]]]

I got from the bottom I have got saloonbar stonecreek mannedfort
```

Appendix-2

Different part of speech tag set used in this integrated system are summarized below. All tags used are based on the HCRC Map Task Corpus tag set [2].

Adjectives
JJ – Adjective
JJR – Comparative Adjectives

Adverb
RB – Adverb
WRB - where

Conjunctions
CC – represent conjunctions like "and", "or"
CS - as

Determiners
DT- Determiner
DPR - Predeterminer

Noun
PRP – Pronoun
NN - noun
EX – Existential there
PD – Demonstratives
WPS – "Who"

Preposition
IN - Preposition
RP- Particles (Prepositions without a noun phrase compliment)

Verb
VBP – verb
HV- Aux verb "have"
BEM – "be" form of the verb
Example: am, be
DO – "do" form of the verb
Example: do, don't
MOD – Modal verbs
Example: can, may, could

Qualifier
QL- qualifier