Department of Computer Science

Technical Report

Undergraduate Dissertation: Generating English Language Based On Formal Grammars

David Graham Perryman

Technical Report 2006-08 **June 2006**

ISSN 1740-9497

Contact Address:
Department of Computer Science
University of Bath
Bath, BA2 7AY
United Kingdom
URL: http://www.cs.bath.ac.uk

ISSN 1740-9497

GENERATING ENGLISH LANGUAGE BASED ON FORMAL GRAMMARS

Submitted by David Graham Perryman

for the degree of

BSc (Hons) Computer Science

of the University of Bath

2006

GENERATING ENGLISH LANGUAGE BASED ON FORMAL GRAMMARS

Submitted by David Graham Perryman

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signature .....

This dissertation may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purposes of consultation.

Signature .....

Abstract

This dissertation presents a method for character-based story generation in English, which is based upon a transformational grammar. Initially the structure of the English language is explored, and types of words are described. Several different structural representations of English are introduced, and appropriate methods selected for the syntactic construction of an English sentence. Some concepts of semantic-driven text generation are investigated, including character-based, and an approach for the program is devised. The implementation phase is detailed, describing how both syntactic and semantic elements are brought together to form final output. Finally a critical analysis of the methods used describes which approaches were more successful than others. It is demonstrated that small phrase grammars can be very expressive, and that character-based story generation is feasible, especially when character goals are introduced.

Acknowledgements

I would like to acknowledge my supervisor Dr Marina De Vos for her help and advice during the course of this project. I would also like to acknowledge my aunt for her proof reading, my parents for their continuing support in my studies, and my fiancee for her patience while I was working on this project.

Contents

Chapter 1

Introduction

The automated generation of English language is a problem that dates back as far as the age of modern computers. Alan Turing devised the "Turing Test" in 1950, which is a test of a machine's capacity to engage in realistic conversation (Turing [1950]). More recently computer game designers explore the area with progressively more accurate simulations of human reactions, in a market where reality is increasingly essential.

The applications of linguistic tools are wide ranging, from grammar checkers in word processing programs (Temperley [2005]), and translation tools (Corb-Bellot [2005]), to text generation in computer games (Cavazza [2005]), and artificial intelligence programming (Mateas [2005]). Therefore any study into the automated generation of language is significant, and has many different potential uses. This project will not attempt to solve all the problems related to linguistics, but will try to explore a simple but elegant approach to text generation.

The cornerstone of realistic reactions is textual generation; to be able to generate syntactically correct language based on the current situation. This project seeks to explore textual generation, starting at the very lowest level involving formal grammars, and progressing into basic character based story generation. Initially the structure of a sentence will have to be investigated, and a suitable representation found. The semantics of sentences will then be explored so that larger blocks of text can be created.

The structure of language is taken for granted in most common use, but is vi-

tal if an automated process is to produce syntactically correct sentences. There have been many different attempts to represent English as a structure, Chomsky [1957] outlines a method for representing the structure of English via a "Transformational Grammar". This is one of many concepts that will be explored to help gain a wider picture of linguistic theory.

Creating semantically viable output is an entirely different challenge from creating syntactically correct sentences. The generator must be aware of the concept of semantics, and the meaning of the different words involved in a sentence. The current state of the system is built up from all the previous sentences that have been generated. Using this state and the different meanings of the words, the program must be able to generate the next sentence. Mateas [2005] uses a storyline "arc" to guide the progression of action. This is one option to help control the way in which sentences are produced, but others will be required to generate smaller groups of sentences.

Due to the wide applicability of linguistics, there are many potential ways that this project could be extended. One of the aims of the project is to allow for extensibility wherever possible, so that further development can be more easily achieved.

This dissertation is organised as follows: In chapter 2 the literature review explores many different relevant sources of information and existing systems. The analysis chapter brings these sources together and selects suitable approaches for this project. The requirements for the system are set out in chapter 4. The design and implementation chapter explains in detail the techniques used when creating the system, and the output from the finished system is reviewed in chapter 6. The approaches that were taken are evaluated in chapter 7, and possible future developments are outlined in the further work chapter. The conclusion explains what has been demonstrated by this project, and what information can be used in future work.

Chapter 2

Literature Review

2.1 Identification of Sources

English language generation requires research into several different areas of theory about the way in which language is constructed, and about the way in which language can be represented using logical structures. These structures can then be represented in a computer, and the language produced.

The first area to research is the study of language from a linguistics viewpoint. When learning English, or indeed any other language, what are the key features that have to be grasped? It is important to understand the fundamental facts about languages before attempting to represent them mathematically. Finding out the different ways that linguists break down a language will be valuable when trying to represent a language using mathematical structures.

The next area to research are the logical structures that may have the power to represent a language. Which are the most suitable out of the many different mathematical theories about the nature of language? The study of the different types of grammar is essential to the construction of a system to generate language, because grammars describe languages (in the mathematical sense).

It is also important to look at existing systems which have attempted to perform similar tasks. What approaches have been taken, and how successful have these approaches proved? The analysis of other systems may help to clarify the strengths and weaknesses of the mathematics that they are based upon, and

therefore show which theories work better when put into an active environment.

2.2 Study of the English Language

"Knowing grammar, and 'knowing about' grammar are two different things" Newby [1987]

Most people implicitly know their native language, but actually knowing about how that language is constructed is a different thing. It is as if the language is automatically acquired, but the brain conceals the complexities of how it is constructed. When studying English, an important fact to establish is that there are many different forms of English; different dialects, accents and styles throughout the world. However, all these variations of the language must share a common core of some kind, otherwise communication between people of different regions would be almost impossible. It is this "Standard English" that is to be studied, as it contains the fundamentals of the language.

The building block of any (western) language is its alphabet: the letters that are used to construct all the more complicated structures of the language. These are built into phonemes, the basic sounds of the language. The study of how phonemes are combined into words is outside the scope of this project, but it is significant to acknowledge their existence.

2.2.1 Words and Morphemes

Morphemes are combinations of one or more letters, and words are combinations of one or more morphemes.

Examples:
1. Play
2. Play +s
3. Play +ing
4. Play +ful +ly

In these examples 'Play' is a morpheme, but also a word. 'Play' is therefore a free morpheme (as opposed to a bound morpheme), as it can exist alone. In (2)

an ‘s’ is added to the end of ‘Play’ to get ‘Plays’. The application of this morpheme to ‘Play’, changes it from the verb infinitive to the 3rd person singular form. Adding the morpheme ‘ing’ in (3) changes the verb to the imperfect form, and in (4), two morphemes are applied, and the result is an adverb, changing the type of word entirely.

The combining of morphemes to form different words is called morphology, and it is clearly an important part of the study of language, as it can change the meaning and tense of words. Even if the correct free morphemes are selected, in the correct order, if the wrong bound morphemes are applied, then the sentence will not be grammatically correct. There are some clear rules that can be applied to words in order to alter them in the desired manner, but there are also many different irregularities in the English language, so morphology will have to be handled carefully.

2.2.2 Types of Words

There are 10 different types of words as identified in Newby [1987]:

Nouns, Verbs, Adjectives, Adverbs, Articles, Determiners, Pronouns, Prepositions, Conjunctions, Interjections.

All of these words must be considered when constructing a sentence, but the most important words are the verbs and nouns. Verbs being the actions within a sentence, and nouns being the objects that the actions are being performed upon. These elements form the underlying structure of the sentence, and many of other words fall into place once the verbs and nouns are identified.

Verbs & Auxiliaries

There are four main forms of English verbs as identified by Zandvoort [1975]:

a) The stem = infinitive
b) The stem +ing = imperfect
c) The stem +sibilant (usually ‘s’) = 3rd person singular
d) The stem +dental (usually ‘ed’) = past tense

Most verbs in English have these different forms; there are some that are irregular, these will have to be handled explicitly. As well as this regular type of verb, there is also a class of verb which is referred to as the auxiliary verbs. These are used to change the mood or the tense of other verbs in the same sentence (Zandvoort [1975]). The set of auxiliary verbs can also be broken down further, in Lasnik [2000] a list of 'modal' auxiliaries is given:

may, might, will, would, can, could, must, shall, should

The verbs "to have" and "to be" are treated as special cases, as they can appear in addition to any modal auxiliary in a sentence. As can be seen from the examples 5-8 up to three auxiliary verbs can be in a sentence at any one time.

Examples:
5. He slept
6. He might sleep
7. He might have slept
8. He might have been sleeping

The order of these auxiliaries is fixed to "modal-have-be" (Lasnik [2000]), but any one of them can be removed and the sentence is still valid. It is important to see that the addition of the auxiliary verbs within the sentences changes the form of the verb that they are affecting. This change is explored further in section 5.5.2.

Nouns & Pronouns

There are two different types of noun, common nouns, and proper nouns (Bach [1974]). Common nouns require that a determiner or article be placed before them to make any sense, where as proper nouns can be used alone.

Examples:
9. The monkey
10. Some monkeys
11. Fred

In examples 9 and 10, "monkey" is a common noun; it is not valid to say, "Mon-

key climbed the tree". However in example 11, "Fred" is a proper noun; it is valid to say, "Fred climbed the tree". This is a relatively trivial problem to solve when generating the language, but it does add complexity to the eventual grammar.

Pronouns, as identified by Zandvoort [1975], have three different types; first person (I), second person (you), and third person (he, she, it). Pronouns are used in place of nouns, when talking about an object that is already defined. It is important when using pronouns that it is clear who is being referred to, otherwise sentences may become ambiguous.

Adjectives & Adverbs

Adjectives are words which are usually used to describe a (common) noun that follows them. Adjectives are inserted between the article and the noun and, as shown by the examples 12-14, any number of adjectives can be used at the same time; there is theoretically no limit. However sentences become unreadable if too many are used.

Examples:
12. The red bus
13. The bright red bus
14. The big bright red bus

It is possible to produce some bizarre sentences if random insertion of adjectives is applied, because the adjectives are closely related to the noun they are describing. For example, it is not valid to say "the red idea", because "idea" is an abstract noun, and only concrete nouns can have a colour. When adding adjectives into a sentence it will be necessary to ensure that they are linked to the object that they are describing in some way.

Adverbs are similar to adjectives, usually formed by adding the morpheme 'ly' to the end (Aarts [1998]) of an adjective.

"Adverbs are used to modify a verb, an adjective, or another adverb" Aarts [1998]

Although they can be applied to adjectives and adverbs, adverbs are usually applied to verbs, however not all adverbs can be applied to all verbs. The sen-

Phrase Type	Head	Example
Noun Phrase	Noun	**the children** in class 5
Verb Phrase	Verb	**play** the piano
Adjective Phrase	Adjective	**delighted** to meet you
Adverb Phrase	Adverb	**very** quickly
Prepositional Phrase	Preposition	**in** the garden

table 2.3.1 different types of phrase (Aarts [1998])

tence "he accidentally thought" is not valid, because it is not possible to think something accidentally. Therefore when choosing which adverbs to insert, it is important that they relate to the verb correctly, otherwise the sentence will not make any sense.

Remaining word types

Prepositions are words that fit in-between the verb and the noun in a sentence, like 'to', 'out', or 'on', (Zandvoort [1975]). These words are linked to the verb that precedes them, and only prepositions that agree with the verb can be used in a sentence. Interjections are words like 'oh', or 'ah', which can occur in many different places, they are not very important to this project, as they do not form part of the structure of a sentence. Conjunctions are words like 'and', or 'but', and can be used to join different clauses together to form larger sentences.

2.2.3 Phrases

Phrases are elementary collections of words. There are several different types of phrase, based around the different word types:

Noun phrase, Verb phrase, Adjective Phrase, Adverb Phrase, Prepositional Phrase Zandvoort [1975]

Each of the phrases has a "head" word , which is the main word in the phrase, and the rest of the phrase is based around that word. An elementary noun phrase consists of just the noun, with article, and optional adjectives. An elementary verb phrase consists of the verb, any auxiliary verbs, and possibly a noun phrase, or prepositional phrase that follows it. Table 2.3.1 shows these different types of phrases, and gives examples of them.

The structure of phrases can be quite complicated, and one of the aims of this project is to produce a grammar which can adequately define this structure.

2.2.4 Clauses

Phrases are only small portions of text, centered around the 'head' word, but clauses consist of one or more phrases which are combined to form more complex statements. The different elements that can be in a clause are outlined in Newby [1987] as:

Subject - Noun Phrase
Verb - Verb Phrase
Direct Object - Noun Phrase
Indirect Object - Noun Phrase
Subject Compliment - Noun Phrase / Adjective Phrase
Object Compliment - Noun Phrase / Adjective Phrase
Adverbial - Adverb Phrase / Prepositional Phrase

The verb phrase is the only compulsory element in a clause, and is either extensive (which means the subject does something to the object) or intensive (which means that the subject is the same entity as the object). Some verbs are intransitive, which means that there is no object in the clause. The Subject and Object compliments are phrases which are related to the subject and object respectively, and inform the user about what, or who the entity is. The adverbial phrase is an additional phrase usually giving extra information about when or where the clause is taking place.

2.2.5 Sentences

Sentences are formed from collections of clauses, there are two main ways in which sentences can be formed: co-ordination and subordination (Traugott [1972]). Co-ordination of clauses is joining two sentences together using a conjunctive word, such as 'and', 'but', or 'however'. This is a very simple way of forming a more complicated sentence from two clauses, but the clauses have to be semantically linked in some way, or the resulting sentence will not make sense. When using a conjunction it is possible to use pronouns in the second clause, because usually

the subject of the sentence will be defined within the first clause.
The other way to form sentences is by subordination of clauses; this is where another clause takes the place of the noun, or adverbial phrase, within a clause. For example the clause "The cat chased the mouse" can be altered by using the clause "the mouse from the barn" in place of the second noun phrase, forming "The cat chased the mouse from the barn".

There are four main types of sentence that are identified in Newby [1987]:
Statements, Commands, Exclamations, Questions

Statements are used to convey information of some kind; they state facts. They are the most common type of sentence, and therefore will be the focus of study for most of this project. Commands are used to tell someone what to do, and typically start with the verb. Exclamations are usually said by a person, and are not necessarily very well formed. The Question, or interrogative class of sentences are quite important, there are four different kinds of interrogative sentences according to Aarts [1998]:

Yes/no, Alternative, Wh-, Tag Questions

Yes/no questions require a response of yes or no, they are formed by switching the noun and verb at the front of the sentence (15,16). This is a very simple way of creating a question from a statement style of sentence.

Examples:
15. This is your book
16. Is this your book?
17. Is this your book, or hers?
18. This is your book, isn't it?

Alternative questions give a list of options for the answer (17). Wh- style questions are sentences that start with what/when/why/where. Tag questions are questions formed by adding a tag to the end of a statement, which questions its validity, therefore making the whole sentence a question (18).

2.3 Study of Syntax

"Syntax is the study of the principals and processes by which sentences are constructed in particular languages" Chomsky [1957]

Syntax is a very broad topic, it covers all possible studies into the grammars of languages. This includes looking for grammars to languages outside of linguistics to explain other phenomena, which may have an underlying mathematical structure. This study attempts to find a formal method to define a set of sentences that are in a language, and therefore to exclude, or reject, the sentences that are not in that language.

2.3.1 Statistical Analysis

One plausible approach to solving this problem is statistical analysis; that is calculating the probability of each word appearing in a sentence, and working out what chance a word has of following another word. This kind of approach is outlined in Harris [1982], where a language is defined as a set of sentences, and each word in the language has a chance of appearing in each sentence. Some words may have zero chance of appearing in a sentence if other words are already present, and some words may have a 100% chance of appearing next in a sentence if another word is there. To construct such a system would require gathering information about the valid sentences of a language, and then performing some analysis of those sentences to calculate the statistics of the system. However this approach is dismissed in Chomsky [1957], using the following examples:

19. Colourless green ideas sleep furiously
20. Furiously sleep ideas green colourless

Chomsky argues that English is an infinite language, there are infinite possibly ways of combining words into sentences. Therefore it is impossible to analyse every possible sentence, and so the chance of generating a sentence like 19 would be the same as 20. All of the words are the same, but 19 is grammatically correct, and in some far-fetched reality could be a sentence. In Chomsky [1957] it is argued that a statistical approach replaces the very slim chance of 19 with zero chance, which is incorrect (since it excludes a sentence that is valid), therefore invalidating the approach.

2.3.2 Finite State Machine

Another approach to defining valid sentences is to use a finite state machine to produce all valid sentences in a language. When a word is output the machine moves from one state to the next. The machine would have a starting state, and one or more end states. Languages that are generated this way are called finite state languages, and can be enhanced by adding a probability of following each path of the machine, this creates a finite state Markov process.

However Chomsky also excludes this approach from being powerful enough to produce complicated languages in Chomsky [1957] because it cannot handle applying more than one rule at any one instance. Therefore it is not powerful enough to describe natural languages.

An extension to the finite state machine method for language processing is the "augmented transition network". In this approach, additional tests are performed before progressing from one state to another. This greatly increases the power of the machine. However, it still lacks the ability to apply multiple rules at the same time.

2.3.3 Phrase Structure Grammars

A phrase structure grammar, also known as a context free (CF) grammar, is defined in Gross [1970] by:

1) A finite terminal vocabulary
2) A finite auxiliary vocabulary
3) An axiom (start symbol in the auxiliary vocabulary)
4) A finite number of rules from the auxiliary set to the auxiliary + terminal set

A phrase structure grammar is defined by having a set of symbols (in the case of a natural language these are mainly the words) (2), with one of these symbols being the start symbol (3). The provided rules (4) are applied one at a time, transforming the start symbol into different sets of the symbols. The rules are applied until it is not possible to apply any more rules, i.e. when all the symbols that are remaining are terminal symbols (1).

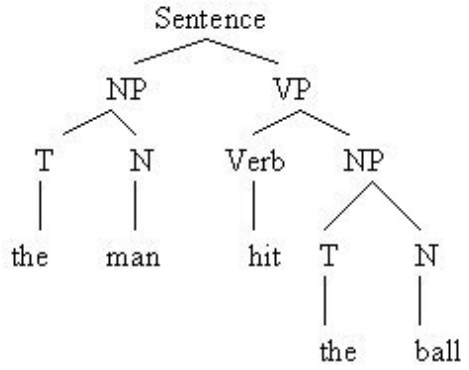

Figure 2.1: example output from a phrase structure grammar

Phrase structure grammars are very powerful, and capable of producing many different derivations, depending on the set of rules provided. It is helpful to be able to display derivations in a tree format, so that it is clear how the result was achieved. An example tree structure is shown in figure 2.1.

Although very powerful, phrase structure grammars are also limited, because they are unable to handle linked structures. The best demonstration of this is returning to the morphology of auxiliary verbs mentioned in section 2.2.1. In Lasnik [2000] it is shown that "to have" and "+en" enter the sentence together, but the +en (which is a past morpheme) is attached to the next verb in the sentence (after the auxiliary). Furthermore when adding the verb "to be" into sentence, the morpheme "+ing" enters as well, attached to the verb that follows "to be".

"This represents a "cross-serial dependency". Phrase Structure rules cannot in general deal with these dependencies." Lasnik [2000]

If phrase structure grammars cannot deal with cross serial dependencies, then a different device is needed to produce such features as can be observed in the auxiliary verbs.

2.3.4 Transformational Grammars

A transformational grammar is a phrase structure grammar, with another set of rules that are applied to the terminal string that is produced. These new rules, called transformations alter the tree that is produced by the phrase grammar,

and create new derivations from it. This is a powerful concept because it allows the cross dependencies to be handled by re-arranging the tree after the sentence has been produced (Lasnik [2000]). Transformational grammars also allow the phrase grammar to become less complicated. This is because some of the rules that would have been in the production rules of the phrase grammar can be represented more easily as transformations.

Examples:
21. The dog is barking
22. The dog barked

Examples 21-22 are a demonstration of a change that can be made when using a transformational grammar instead of a phrase structure grammar. In a phrase structure grammar, both these sentences would be different representations, and be produced by a slightly different set of production rules. However, in a transformational grammar both these sentences can be produced from the same underlying phrase structure representation. If example 21 was generated by the phrase structure, then a transformation could be defined to change the tense of the statement, and generate example 22. These ideas are outlined in Chomsky [1957].

However, these sentences seem linked more fundamentally than just being transformations of the same phrase grammar; they are both part of a family of sentences that contain the concept of a dog, and it barking. The underlying structure to the sentences is referred to as the "deep structure". It relates to the meaning of the sentence, and although it is unpronounceable, it is theoretically there (Fowler [1971]). There is a relationship between the deep structure and the semantics of the produced sentence (Bach [1974]), however it is difficult to capture, and not part of this project. The final output of the transformational grammar is the "surface structure"; this is the pronounceable part of the sentence, and the aim of the production, but it contains both semantic and syntactic content, not pure semantics.

A transformational rule contains a structural analysis (SA), and a structural change (SC). To be able to alter the structure of a production of the phrase grammar, some structural analysis must be performed on the tree that is produced. This is a pattern matching exercise, as outlined by Lasnik [2000].

Example:
23. SA: X - en - V - Y SC: X1 - X3 - X2 # - X4

The example 23 is a transformational rule which moves an 'en' to the other side of a verb, and then binds it. The program has to look for the pattern X - en - V - Y, where X and Y are any term, en is the morpheme 'en' (which would be added with the 'to have' auxiliary) and V, which is a verb. Once this pattern has been found, the 2nd and 3rd nodes (in the phrase grammar tree) are switched, and a word boundary placed after the 3rd node. This word boundary binds the 'en' morpheme to the verb.

There are four elementary operations that a transformation can perform on the tree of a phrase structure grammar, which are combined to form more complicated operations. While performing these changes, the transformation should try to preserve as much of the structural information as possible, so that the maximum amount can be known about the phrase structure derivation. The four rules, as given by Lasnik [2000], are:

a) Adjunction of one term to another (left or right)
b) Deletion of a term, or sequence of terms
c) Adjunction of new material (that was not in the structure before)
d) Permutation (changing the order of two items)

Using these four alterations it is possible to do numerous amounts of transformations, some of which are outlined in Chomsky [1957]. Some transformations are obligatory (the rule must be applied if the structural analysis is found), and some are optional. The fact that it is necessary to alter the tree to produce the final sentence means that any program designed to use a transformational grammar will have to construct a tree structure (created by the phrase structure grammar), which can be passed to the transformational rules, and manipulated to form the new sentences.

2.3.5 Lexical Insertion

When building up the phrase structure much of the structure is dependent upon the verb that is being used. Because some verbs are transitive, and others in-

transitive, some verbs require a noun follow them (be the object in the sentence) and others explicitly do not allow a noun follow. Furthermore the nouns that are chosen to occupy the subject and object have to be of the correct type for the sentence to make sense (Thomas [1974]).

Examples:
24. The party was fantastic
25. The fantastic was party

While example 24 is valid, Clearly example 25 is not a correctly formed sentence, because the verb 'to be', requires a concrete subject, which is a noun that is physical object. Nouns can be classified into many different categories. Things that are concrete or abstract, animate or inanimate, animals or humans, and many more besides. The best way to identify which nouns are appropriate is for the verb to specify which forms are acceptable, somewhere in its definition (Fowler [1971]).

If a verb specifies that it must be followed by a concrete noun, then that also means it can be followed by any objects that are sub categories of concrete, such as animal or human, animate or inanimate. This means that when selecting nouns to fill the object, the ideal structure is a tree, where everything below the concrete node satisfies the condition of concrete. It is possible that a noun might satisfy several different categories at the same time, and then may have to be added to more than one of the nodes, if they are not sub-categories of each other. The process of selecting and inserting the correct noun into the sentence is called lexical insertion. However, if a verb specifies that it does not allow a noun to follow it, then this will have to be taken into account within the phrase structure grammar, before the lexical insertion takes place.

It may also be necessary for verbs to specify which prepositions (if any) are allowed (or required) to follow it, because if an incorrect preposition follows the verb, then the sentence is incorrect. This could be achieved in a similar way, with the rules being taken into account as part of the phrase structure grammar. The preposition used may affect the type of noun that is needed at the lexical insertion phase.

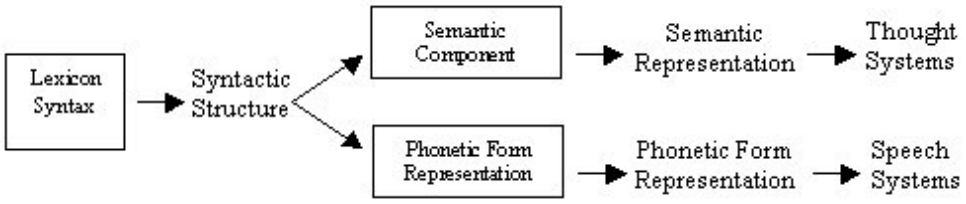

Figure 2.2: diagrammatic representation of the minimalist program, Radford [2004]

2.3.6 The Minimalist Program

Another slightly different approach to the problem of generating sentences is the minimalist program. This is essentially a new look at the underlying structure of sentences from the semantic and syntactic viewpoint. It was designed because of the complexity that had developed in the traditional approach, and to attempt to model the pattern of human thought more accurately (Radford [2004]). A diagrammatic representation of the minimalist program is show in figure 2.2.

The derivation begins with the numeration of the key objects that are going to be talked about in the sentence e.g. the verbs and nouns, then a structure is produced containing these objects. At some point in the process there is a split between the Logical Form (LF) and the Phonetic Form (PF). The logical form contains all of the semantic information, and the phonetic form contains all of the syntactic information. The point at which this split occurs is sometimes called 'spell out' (Cook [1996]).

The minimalist program attempts to break down the problem into smaller, more easily solved problems, and has been developed based upon the way that children learn language. One of Chomsky's aims as he was working on the minimalist program was to try to create a "universal grammar"; one that defined why a language was learnable, and linked all forms of human grammars (Radford [2004]). This means that some of the concepts are more general than just the study of English, and that the minimalist program is more closely linked to psychology, and linguistic theory.

2.3.7 Word Grammar

Word grammar is a grammar that does not use phrase structure as its base; it is based entirely upon dependencies between words in a sentence. Each word that can appear in a sentence is analysed, and the dependencies of that word calculated. For any particular word to appear in a sentence, all the dependencies associated with the word must be satisfied. This may take several passes to achieve, because the changes made to satisfy one word might invalidate another word (Hudson [2005]).

This technique is significant because it does not use phrase structure, which is what the vast majority of language studies are based upon, and also because it does not use a 'surface structure' and 'deep structure' system. Instead, the collection of words is progressively parsed until all the dependencies are satisfied. This takes the representation from the semantic to the syntactic. The fact that this approach is successful means that phrases are not fundamentally tied to their phrase structure definitions, and that flexibility outside of phrase structure is possible. However, this structure does mean creating full definitions for all the words that are used, which involves quite a large and complicated data store.

2.3.8 Tree Adjoining Grammar

Another grammatical technique is called the tree adjoining grammar. This builds up a language based upon trees that are generated from a phrase grammar. A set of initial trees and a set of auxiliary trees are combined using substitution and adjunction, to form derived trees that are in the language (Joshi [1997]). Using this approach, the language is broken down into smaller phrases that are combined using substitution and adjunction to form more complex sentences. This approach can be used to create a model for English (it is used in Cavazza [2005]), and is not very different from a transformational grammar. However, it lacks some of the expressiveness of a full transformational grammar, because the primary operations are substitutions and adjunctions, where as transformations can manipulate the structure of the nodes in more complex ways.

2.4 Study of Existing Systems

There are several different types of system that use grammars to analyse the structure of language, apart from simply generating text. Many word processing programs have grammar checkers built into them that analyse the text written and attempt to work out if it is a valid sentence. Translation programs attempt to translate text from one natural language to another, which involves reading in the input text, performing some semantic analysis, and reproducing it in a different syntax. There are numerous sentence generator programs available on the Internet, most of limited functionality, but there are several approaches taken and examining them is appropriate.

When trying to generate linked sentences, and groups of sentences that are based around the same area, the main source of examples are interactive computer games. Some new multiplayer games have interactive characters that are entirely AI driven, and respond based upon their relationships with the other characters within the environment. This type of game is where most of the current development into sentence production is happening, because of the market value of such games.

2.4.1 Grammar Checkers

"The Link Grammar Parser is a syntactic parser of English, based on link grammar, an original theory of English syntax." Temperley [2005]

One of the grammar checkers that is available on the Internet is the 'link grammar parser'; it uses a method of textual analysis called link parsing.

"Think of words as blocks with connectors coming out. There are different types of connectors; connectors may also point to the right or to the left. A left-pointing connector connects with a right-pointing connector of the same type on another word." Temperley [2005]

Essentially this method analyses the text, and matches the words with words from the dictionary. The words that are in the included dictionary have rules attached to them that describe what 'links' to other words they are allowed, or are required, to have. This approach seems to work quite well, but is probably not

applicable to text generation. For each word that is in the dictionary, the rules have to be explicitly defined, which means that the system is not easily extensible. This system also fails to take into account the links between the different forms of verbs, and of morphology (each different word is separately defined). Not allowing for concepts like these would limit the power of a text generation system, and result in a large amount of coding to produce all the links required. However, using a system such as this could be useful to check the output of the system, to verify if the sentences produced are in correct English.

In Naber [2003] there is a description of a 'rule based' grammar checker, which is a grammar checker that breaks the text into tags which have a particular word type, and checks those tags against a set of rules. Naber states:

"It turns out there are basically three ways to implement a grammar checker:
Syntax-based checking
Statistics-based checking
Rule-based checking" Naber [2003]

Syntax based checking involves parsing the whole text, and comparing each sentence against a grammar. If the text does not match the grammar, then it is incorrect. The main drawback of this approach is that it requires a full grammar of the language to be written. Statistics based checking looks at the text, and breaks it into simple phrases, assigning each word in the phrase with a word type (e.g. noun or verb). It uses statistics to calculate the likelihood of the words appearing in the order that they have been written; if it is a very low probability then the chances are that an error has been made. The main problem with this approach is that sometimes unlikely sentences are grammatically correct, and sometimes likely looking sentences are in fact incorrect.

Rule based checking tags each word in the text with a word type, and compares the sequences of word types against error rules that it contains. If it matches one of the rules, then there is an error. This means that a grammar does not have to be defined to cover the whole language, and also extra rules can easily be added to account for previously undefined errors.

The concept of holding error rules to check if something is incorrect is important; grammars can be simplified by using this technique.

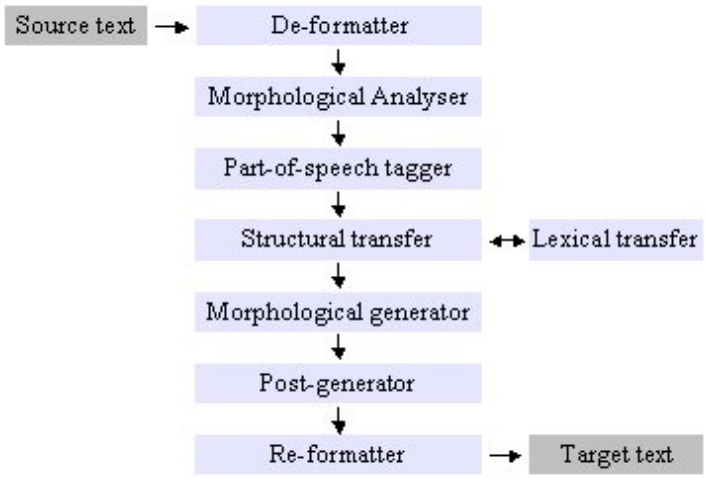

Figure 2.3: the eight modules of the MT system, Apertium [2005]

2.4.2 Language Analysis Tools

"Apertium is designed to translate between related languages" Corb-Bellot [2005]

Apertium is an open source language translation tool, which is designed to translate between two closely related languages. This type of translation is called "shallow transfer" because there is little semantic analysis required, the languages being closely linked. The type of architecture used within this system is called the Machine Translation (MT) system, and can be seen in Figure 2.3.

The modules of importance in this system are the structural transfer, and the lexical transfer. The structural transfer module breaks down the source text into pieces that can be directly translated, some just single words, others simple phrases. The pieces are then passed to the lexical transfer module, which is a dictionary of translations, and converted into the target language. The morphological generator handles any morphemes that may have been entered into the system by translation into the new language. The remainder of the modules are concerned with parsing the original text, and restoring its form once it has been translated.

Because this system is designed to translate between two fairly similar languages, the analysis of the text is not very strong. When creating a program to generate text, the second half of this system has to be produced, from the lexical transfer module downwards. However, there is no source language, so the concept of using translatable phrases is not be applicable. It is be possible to generate text based

upon a dictionary that contains lists of acceptable phrases, but this would limit the power of such a system, restricting the amount of possible sentences that could be produced. A system based upon words rather than phrases would be more powerful.

There is an English grammar tool called the linGO Grammar (linguistic online grammar), this is a project run at Stanford University to produce a grammar to describe the English language (Copestake [2000]). It uses a head driven phrase structure grammar, and a 'lexical knowledge base'. A head driven phrase structure grammar (HPSG) is a phrase structure grammar that is based around 'head' words in each phrase e.g. the verb in a verb phrase defines how the phrase is structured (Sag [2001]). This is an attempt to simplify the process by removing some of the elements from a transformational grammar, and placing them in the phrase structure, which is organised into hierarchies of different kinds of phrase.

The lexical knowledge base is a very important part of the system; it is a structure that contains encyclopaedic knowledge of the English language. This enables the grammar to produce sentences that have semantic meaning as well as syntactic correctness. Without a lexical knowledge base, the grammar would be unable to combine ideas in a coherent way, and produce sentences that have meaning.

2.4.3 Existing Random Generators

There are numerous different scripts and programs on the Internet that generate text in a random, or pseudo random, manner. Most of the programs are fairly basic, but there are two main techniques used when creating random generators.

'Phrase Grammar' based

The phrase grammar based generators use a grammar whose components are phrases, rather than words. These phrases are interchanged with each other to produce sentences, and groups of sentences. There are also generators that simply switch in and out the verbs and nouns from a 'script', meaning that the text that is generated is always the same apart from the names and actions. The more advanced versions have numerous different variables and different sentence structures. These can generate some advanced passages (see Appendix A.1 for a sample taken from Zelenski [1999]).

Although some of the more complicated grammars can generate plausible passages, this method of generation is fundamentally limited to the structure that exists within the grammar. Without creating a very complicated grammar, there is no way that sentences being produced can depend upon the sentences that have already been produced, i.e. no linking between sentences, and the majority of the logical analysis of linguistics is not taken into account. Therefore, although this type of program produces acceptable results (within their domain), there is little scope for enhancement.

Phrase Structure based

The phrase structure based generators use a phrase structure grammar to model the structure of a sentence, and then randomly inserts words into the positions in the grammar to form sentences. This is a more powerful model of producing sentences than the phrase grammar based systems, because sentences are produced in a free-form manner, not constrained by any 'script'. However, this method usually has the side effect of producing completely unintelligible sentences, due to the random insertion. It is also difficult to link sentences together, as usually each sentence is just as random as the preceding sentences. A good example of this type of generator can be found at Kelly [1998].

The generator at Kelly [1998] proves that sentences can be created using the phrase structure method, however it uses quite a limited dictionary in its examples to ensure that valid sentences are produced. This makes it not much different from the phrase grammar version. The two areas that need to be improved using this technique are ensuring that the correct nouns are inserted (so the sentence makes sense), and linking subsequent sentences together. Using an advanced system of lexical insertion fixes the issues with invalid nouns. Linking subsequent sentences together requires a higher form of semantic analysis, to ensure the sentences are about the same subject or topic.

2.4.4 Multiple Sentence Generation

One of the aims of this project is to explore the way in which sentences are linked together, and generate multiple sentences at the same time. This is connected to the semantics of the sentences that are produced, and the concept has been

explored in the creation of interactive video games. Characters react in a realistic way when the user interacts with them, based upon the current circumstances in the game.

A method of generating speech between two characters is outlined in Cavazza [2005]. This method is based around the characters that are communicating and events that are happening in the game. It uses the affinity between the characters, the goals of the conversation, the events taking place, and the roles of the characters to build a semantic picture of the sentence that is to be produced. This semantic picture is then made into a sentence by using a 'Tree-Adjoining Grammar'.

Example of a semantic level structure from Cavazza [2005]:
(?interrogative) (?actor) (?take-part) (?event :type:party) (?event ?property)

The examples outlined in this paper are set in quite a limited framework with quite a specific goal, however the principal is shown. Introducing characters into the system, and the idea that these characters can have affinities, means that their actions can be partially defined by these affinities and current setting. This is slightly removed from the context of characters speaking to each other, but the principals of having some global 'setting' which helps keep the sentences linked within that setting are useful.

"Façade is an artificial intelligence-based art/research experiment in electronic narrative" Mateas [2005]

Façade is a game where the user enters a story as themself, and interacts with the characters involved, changing the way in which the story evolves. The game is focused around interacting with the other characters in the system, and most of the play is focused on the behaviours and actions of these characters. The system understands natural language, and the characters respond and reply to what is being said. The game is governed by the situation that is presented to the player, and the storyline follows an arc, but is not fully predetermined. The system is controlled by a 'beat' that is part of the program that determines what should happen next in the story, and what the characters next actions should be.

Façade is a real example of character and situation based conversation gener-

ation. Although it is quite constrained in the setting that is given to the player, and the character goals are fairly static, it shows that this technique is viable.

2.5 Conclusions and Plan

From the research done it is possible to derive several tasks to be considered when attempting to create a system to generate language:

- Constructing a viable phase structure grammar
- Lexical insertion of the correct nouns / prepositions
- Morphology, changing the tense of verbs
- Adding transformational rules to create a transformational grammar
- Creating some governing rules to link sentences together

The first task is to construct a framework that allows the creation of a phrase structure grammar, which will ultimately describe a sentence in the English language. This needs to produce a phrase structure tree, so that transformational rules can be applied later in the process. This structure can be created without reference to the generation of natural language, because it is based purely on the rules of phrase structure grammars, therefore it can be made using very simple test grammars.

Once such a framework has been created, a basic description of the English sentence has to be applied to it. The rules for lexical insertion can then be created, to ensure that the correct combinations of nouns and prepositions are used with the correct verbs. These rules may be quite complex, but should be made in as flexible a way as possible, so that the system remains extensible. Morphology can be considered when creating the transformational rules that apply after the phrase structure had been calculated, transformations can be defined to change the tense of a passage, or sentence.

Transformations could also be defined to change the meaning of the sentence that is being generated; examples such as the negation transformation, and the question transformation are given in Lasnik [2000]. When the transformations are being defined, testing will have to be done to ensure that transformations do

not result in generating invalid sentences when applied simultaneously.

During the process of creating the sentence generator, entering the produced text into a grammar checker could check the 'correctness' of the sentences that are produced. All the sentences that the program could create should be correct.

When the system is capable of producing an acceptable number of different sentences, then some governing rules can be created, so that multiple sentences can be generated that are linked in some way. These links could include a location, common characters, time of day, weather, or other situation variables. These ideas could be enhanced by including relationships between the characters, and by giving the characters goals to achieve.

Chapter 3

Analysis

There are several considerations that must be made when attempting to create a system that generates language. From the study of different techniques and existing systems, the following are the main areas of concern:

3.1 Platform

The very first problem to analyse is what platform to develop on. There are numerous different issues that could be discussed about the differences between programming languages, and many different languages that could potentially be used. A primary concern is portability, so that the program can be run on different platforms and it is also a good idea to choose a language that is relatively 'main stream' so that the ideas within the project can be easily transferred. The two languages that meet these criteria are Java and C.

Java is preferable to C because of several reasons: Java is object oriented, and this is more applicable to the problem, because words can very easily be viewed as objects, and sentences viewed as objects that contain word objects. This provides a good model of the system, and would be much harder to achieve in C. The program will involve many string operations, because the fundamental blocks of sentences are words, therefore the chosen language has to be good at handling strings. Although C has a string library, it is much easier to handle strings in Java; this will help to simplify the code. Although C will probably run faster than Java, speed is not critical to this project, and the program will not be large enough to slow the system down significantly. There is also a free foundation

version of Borland Jbuilder available (an IDE for Java), which will help to develop and debug the program. Using Java also opens the possibility of creating an applet from the final program, so that the results can be put online.

3.2 Framework

There are many different theories and strategies that have been looked at to design the core of an English grammar generator. Ideally the solution would be as simple as possible, but also as flexible as possible, allowing the simple addition of rules or words to expand the capabilities. The generation of English is inherently complicated; by breaking it down into very simple parts it is possible to simplify the structure. The reason that a transformational grammar is the preferred option is that there are several different levels of complexity, and breaking the grammar apart allows for each level to be simpler than it would be otherwise.

Techniques that have one level of grammar are more complicated, because they have to handle the whole language in one process, using a system based on transformational ideas will mean that different areas of the system can be developed separately, and therefore made simpler and more robust. The project will be based around a transformational framework, however completely disregarding ideas from other techniques would not be constructive, and this project is intended to be flexible enough to incorporate different methods where appropriate.

3.3 Phrase Structure Grammar

The fundamental building block of most grammars (including transformational) is the phrase structure grammar. This outlines the structure of each phrase, and how the phrases are connected to form sentences, as described in section 2.3.3. A phrase structure grammar will have to be constructed for the system, which will produce the elemental structure of the sentence. This phrase structure grammar will have to be constructed in a way, which makes it easy to modify the way it functions, so that an appropriate grammar can be developed using it.

A phrase grammar essentially consists of a set of production rules, and a start symbol, these are the variable components of the grammar. The production rules are repeatedly applied to the start symbol until no more rules can be used, and

the result is a terminal string in the language that the grammar describes. The rules and start symbol must be stored in way that makes it easy to alter them to change the grammar. This means that a solution such as a recursive descent grammar (where each rule is a function that calls the sub functions, and returns the result) is not appropriate, because it is not easily modified. The phrase grammar must produce a tree of the resulting sentence, so transformations can be applied to the sentence after it has been produced. This means that the code to generate the nodes of the tree, and apply rules to the symbols must be generic, so the rules remain easy to modify, but strong enough to produce a tree.

Because the phrase structure grammar is generic to all grammars, and should work well with any input that is given, it can be tested, and made fully operational, without specifically considering a grammar for English sentences.

3.4 Lexical Knowledge Base

It will be necessary to create a knowledge base to contain words that are to be used in the sentences. This is so that the words can be categorised into different types beyond the scope of a phrase grammar, since words can fall into many different categories, which a phrase grammar cannot represent without duplication of data. The main reason for requiring a knowledge base is to store nouns, and the different types of noun. These nouns will have to be matched against verbs with compatible definitions in the lexical insertion phase. The categories and sub categories of the nouns means that a tree structure is an appropriate representation, where all the words belong to the sets that are defined above them in the tree.

Verbs (not modals or auxiliaries) will have to be stored in the knowledge base, along with definitions of acceptable ways of using them. There are essentially three different types of verb; transitive, intensive and intransitive. These different types work differently in a sentence, and the type of the verb will have to be defined before the structure is generated. Acceptable ways of using verbs include optional, or compulsory prepositions, and the types of noun that act as the object or subject. These definitions will have to be stored in some kind of generic structure, which will be easy to edit, so the system is extensible, and will allow for minimum duplication of data between similar verbs. It may be appropriate

to develop templates for different kind of verbs, if obvious groups appear while the system is being developed.

3.5 Lexical Insertion

The lexical insertion process should take place after the phrase structure has been generated. The purpose of this process is to insert nouns (and prepositions where appropriate) into the sentence that agree with the verb that is at the centre of their clause, as described in section 2.3.5. If it is not already specified, then the verb will have to be selected (of the correct type), and nouns matched against it by referring to the lexical knowledge base.

It is important that this process be as simple as possible, because it will be expanded on when further development is done on linking sentences. Nouns will have to be chosen, not only because they agree with the verb, but also because they agree with the context of the sentence.

3.6 Transformational Rules

After a phrase grammar has been generated, transformational rules can be applied, that will alter the sentence, to ensure its syntactic correctness, and also to generate further meanings based on the phrase grammar sentence (see section 2.3.4). There are several elemental steps that can be combined to create a transformation; these steps will have to be created as functions to be called when performing a transformation. A transformation has two parts: the structural analysis, and the structural change. The structural analysis looks for a place in the tree that the transformation can apply, this is just a tree walk operation, and a generic function can be constructed. The structural change is a combination of the elementary operations performed on the tree, it may be possible to create a generic function that can handle all types of transformation, or one function for each transformation may be required. A generic function would be preferable, because then only the structural analysis and the structural change will need to be stored, and the transformations will be easy to amend and easy to read.

Transformations can also be defined to change the meaning of the sentence that is being generated; e.g. changing tense, negation, and creating questions. Some

transformations will also handle elementary morphology. If different tenses of verbs, and plurals for nouns, are stored in the lexical knowledge base then much of the morphology will be a simple case of switching in the words.

When the transformations are being defined, testing will have to be done to ensure that transformations do not result in generating incorrect sentences unexpectedly, when being applied simultaneously.

3.7 Developing Phrase Grammar

Once the other areas have been implemented, then the original grammar can be expanded to include more possible sentences. This has to be done after the other sections are completed, because the full transformational grammar will be required to develop the grammar properly.

One of the first areas that will be expanded on are auxiliary verbs, these will require transformations and morphologies to be implemented, because of the cross serial dependencies that occur when auxiliary verbs are added. Sentences will be expanded into multiple sentences by using joining words, and also by inserting sentences as sub clauses. This will require generating semantic links between the clauses, and mean passing variables and objects into the sentence generator from the previous sentence. This should be done in such a way that will allow for arbitrary variables to be used to 'seed' the sentences, so that multiple sentences about the same subject can be generated. This may require the development of a semantic level representation of the sentence, which is sent to the generator to be made into syntax.

3.8 Links Between Concepts

To be able to create more than one sentence about a concept, then there must be some way of knowing what is an appropriate sentence to generate. It will be possible to generate a sentence based on the same objects as the previous one, but there needs to be a way of filtering actions and objects out of all possible words in the system. This will require some sort of linked data structure, where the program can filter all the words, based upon the setting and the previous sentence(s).

This is essentially the creation of some encyclopaedic knowledge that the program can use to generate linked sentences; it may be possible to create it as an addition to the lexical knowledge base. The scope of this structure is very large, there are infinite possibilities of linked words, and it will grow as new words are added to the language. Therefore this will have to be limited to a few scenarios to try to show that the concept works.

3.9 Global Settings and Rules

As well as the program being aware of linked words and situations, it will also be necessary for it to know about some more general concepts that can link sentences together, the most obvious is characters. If objects are created to store information about each character in the system, then these characters can interact with each other in the sentences. The information about the characters that is generated can be stored in the objects, and that information used to help generate further sentences. It may even be possible to store an affinity rating between characters, which will govern how they should react to actions by the other characters.

Other information that may be useful when generating sentences is the setting information: where the action is taking place. This will probably be one of the categories in the lexical knowledge base. It may also be possible to store the time of day, weather conditions, and any other information that may be relevant to the sentences. If all these concepts are correctly integrated, it will greatly increase the power of the sentences, allowing them to be much more meaningful. The information should be stored in some global variables, and used when the generating semantics. The semantic generation may be significant enough to require its own module for selecting the words that are sent to the sentence generator, based upon all the situation variables that are currently in use.

3.10 Test Plan

It is quite difficult to measure the 'correctness' of the text that is produced from the system. During the process of creating the sentence generator, entering the produced text into a grammar checker can determine the grammatical 'correctness' of the sentences. However, there is also the semantic correctness to be

considered, especially when generating more than one sentence at a time. This is almost impossible to verify, because the verification process is just as hard (if not harder) than the generation of the sentences. The reader should be able to tell if the text makes sense simply by reading it; this is the best test for semantic correctness that is available.

Chapter 4

Requirements Specification

4.1 Functional Requirements

4.1.1 A grammar must be created that is suitable for representing a sentence in the English language. This does not need to be very complex initially, but will need to be developed during the initial stages of the project.

4.1.2 A dictionary of words will be required, that can be used to create sentences from the grammar. This does not need to be large to start with, but it does need to be well structured, as it will be expanded greatly, when the program becomes more powerful. This dictionary must contain all the information required to perform lexical insertion.

4.1.3 The program must take the grammar and the dictionary of words, and combine them to create sentences, which are then output to the screen.

4.1.4 The process of lexical insertion must take place, to insert the correct types of nouns and prepositions. This must use the lexical knowledge stored in the dictionary, and will occur after the generation of structure.

4.1.5 A transformational grammar must be applied to the structure that has been generated, to allow for more types of sentence to be produced. This grammar must be written in a compact manner, so adding new rules and amending existing ones is an easy operation.

4.1.6 The program must be enhanced so that it is capable of generating linked sentences. This will be expanded to generate several sentences at once, which are all connected by some common element(s).

4.1.7 The program must use character objects to create further text, interacting the characters to create the foundations of a story.

4.1.8 This program does not require a significant user interface, because the primary function is to produce text in a random manner. This only requires the user to press "go" and the text to be produced. There may be a limited amount of interface, for testing purposes, that controls the type of sentences that can be produced, but there is no plan to have any options in the final program.

4.1.9 The aim of this project is not to produce every possible sentence in the English language. However, all the sentences that are produced must be correct, both syntactically, and semantically.

4.2 Non-Functional Requirements

4.2.1 The system must output text to the screen, which can then be read by the user and copied to a grammar checker or printed. This should allow for the output to be checked in any manner necessary.

4.2.2 The system must be able to generate new text when the user presses the button again; this must be a simple process. This is to allow several examples to be generated easily, so an evaluation of the system can be made.

4.2.3 The system must not produce errors; text must always be output. Any errors that occur must be internal only, and handled appropriately. Even if errors occur, the system must still produce "correct" textual output (ideally no errors should occur).

4.2.4 The system must be constructed in a way that makes it simple for a future developer to enhance it, by simple changing of the rules, and insertion of new words into the lexical knowledge base.

Chapter 5

Design and Implementation

The system is broken down into several sections, each handling a different part of the text generation process, and passing the output to the next section. The first part involves the creation of clauses, and ensures the syntactic correctness of the text, the second part of the program handles the semantics of creating multiple clauses together, and building up more complicated text.

5.1 Phrase Structure Grammar

The first section of the first part of the system is the phrase structure grammar. The inputs of this section are the production rules of the grammar, and the output is a tree, with those rules (initially randomly) combined. When designing this part of the system, there are essentially two problems to be solved: how to represent the rules in a simple way, so that they can be altered relatively easily, and then how to combine them to produce a tree. It is also necessary to design a tree structure that is appropriate to represent the output.

The production rules in a context free grammar are of the form $A \rightarrow XYZ$, where A is the symbol that is to be replaced, and XYZ are symbols to be inserted. This means that the when storing a rule, it has two components, an initial symbol, and a collection of rewrite symbols. It is possible to create a grammar that is of the form $A \rightarrow XY$, meaning there are always two rewrite symbols, which would generate a binary tree. However, this reduces the expressiveness of the grammar, and would make the transformational phase more difficult. Therefore, an ordered collection of rewrite symbols will be used. A class definition for

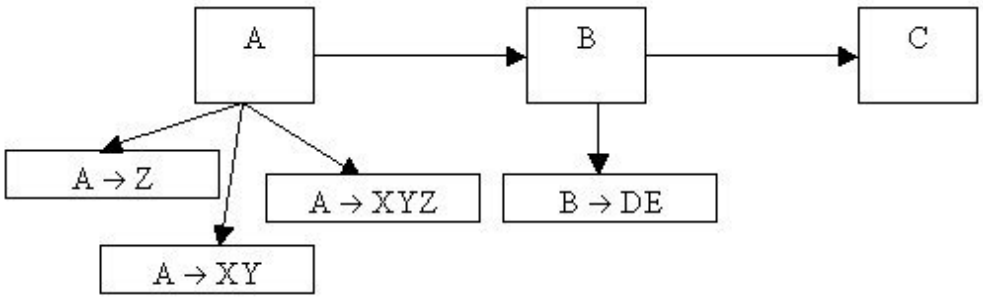

Figure 5.1: grammar rules stored in a structure where A,B and C are initial symbols, and D,E,X,Y and Z are various combinations of rewrite symbols.

a rule (in Java) looks like:

```
//a class to hold a production rule
private class rule
{
   public String initSymbol;
   public String[] rewriteSymbols;
}
```

When the rules are created, they are stored in an array, where they can be searched for when looking for a rule to apply. An enhancement to this sees the rules be stored in a linked list structure, where they are sorted by initial symbol. This is useful because when looking for a rule to apply, the program has to find all the options that exist for a particular initial symbol. An intermediate structure will be used which contains an initial symbol, and a collection of all the rules that relate to that initial symbol. The rule structure after this optimisation can be seen in figure 5.1.

To create a tree based on the grammar that has been given, the program starts with the start symbol (which is also specified), and applies the production rules until it is not possible to apply any more rules. At this point a terminal string has been reached, which is a member of the language that the grammar describes. At each application of a rule, the program creates a tree node, with the initial symbol attached to it, and one child for each symbol in the rewrite symbols. in this way a complete tree of the derivation is produced.

The basic tree node has to have a symbol, that is printed out to represent the node, and a collection of child nodes. To begin the derivation, all that has to be

done is to create a node for the start symbol, then find a rule to apply to that node. When a rule is selected, child nodes are created for each symbol in the rewrite symbols of that rule, and then rules are applied to each of the nodes that were created.

When the derivation is completed, both the tree structure and sentence are output to the screen. Only the leaf nodes of the tree are printed when displaying the sentence, but all the nodes must be printed, and correctly spaced, when printing the tree.

Sample output from the initial phrase grammar generator:
The Monkey killed Burt

```
S
  NP
    PN
      Burt
  VP
    V
      killed
  NP
    CN
      Monkey
    D
      The
```

Where S = Sentence, NP = Noun Phrase, VP = Verb Phrase, PN = Proper Noun, V = Verb, CN = Common Noun and D = Determiner. As can be seen from this derivation, the phrase grammar can produce sentences, but not of a very high quality. The full grammar that was used to produce this deviation can be seen in the appendix A.2, along with the syntax that was used to input it into the system. This syntax demonstrates the compact nature of the representation developed, and shows how easily it can describe a grammar.

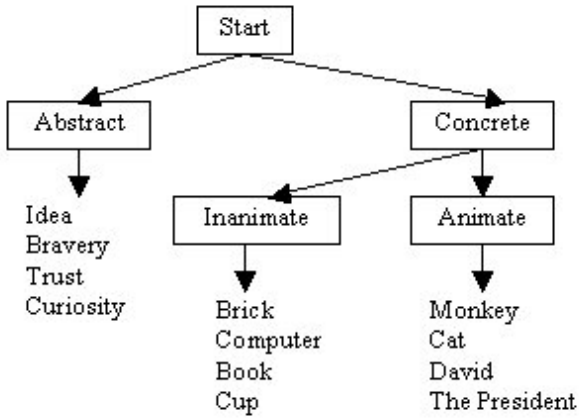

Figure 5.2: example noun structure

5.2 Lexical Knowledge Base

The knowledge base contains nouns, verbs, adverbs, adjectives, and all the mechanisms to store and retrieve them. There are two types of noun: common and proper, and nouns are grouped into categories, such as real or imaginary, animate or inanimate. These categories are tree-like in nature, because objects fall into many categories and sub categories. Verbs fall into 3 different types and specify which types of noun can occupy the position as object and subject in the sentence, and also what prepositions are applicable. This is done in the most flexible, and readable way, so it can be changed easily.

A noun object has a type, a category, a singular string, and a plural string. These noun objects are sorted into a tree of categories; this tree is constructed before any nouns are entered, as adding nodes to it dynamically may cause nouns to be entered into the wrong place. This means that the categories are defined explicitly beforehand, so that a tree can be built. This will start with fairly crude categories, and be developed as the project progresses. An example noun structure can be seen in figure 5.2.

Verbs have essentially got 4 different forms in English, infinitive, imperfect, 3rd person singular, and past. There are 3 different types of verb transitive, intransitive and intensive, but some verbs fit into all of the categories above. Verb rules describe the structure of the sentence that can surround the verb and are in the following different forms:

1) $NounType + verb + NounType$
2) $NounType + verb + Preposition + NounType$
3) $NounType + verb$
4) $Verb + NounType$
5) $Verb + Preposition + NounType$

Functions that match these forms create rule objects, which are stored in the verb object. The rules are then stored in three different lists within the verb, one for transitive rules, one for rules with only an object noun, and one for rules with only a subject noun. This makes it easier to identify what the verb is compatible with, without having to examine every rule closely (verbs are given a type depending on which lists have rules defined in them).

For common verbs, the imperfect, 3rd person singular, and past forms can be calculated from the infinitive form, therefore a default constructor for the verb can be used, which only takes the infinitive, and the other forms are calculated as described below.

Imperfect = Infinitive + "ing"
3rd Person Sing = Infinitive + "s"
Past = Infinitive + "ed"

5.3 Lexical Insertion

Lexical insertion is the process where the verbs and nouns that are in the phrase structure are given values. The first word that has to be established is the verb, because the verb determines what types of noun can act as the subject and object in the phrase. Once the verb has been placed, the surrounding nouns must be inserted. To enable this, the nouns must be in some way linked to the verb. This is done using a verb clause, where a verb phrase is the central child, and noun phrases are left and right children. Figure 5.3 shows the layout of a verb clause.

If a verb is always part of a verb clause, then it is always possible to find the object and subject nouns that are associated with it. The initial functionality is that the program locates the verb node, and assigns it to a verb from the lexical knowledge base, then locates the nouns that are associated with the verb, and

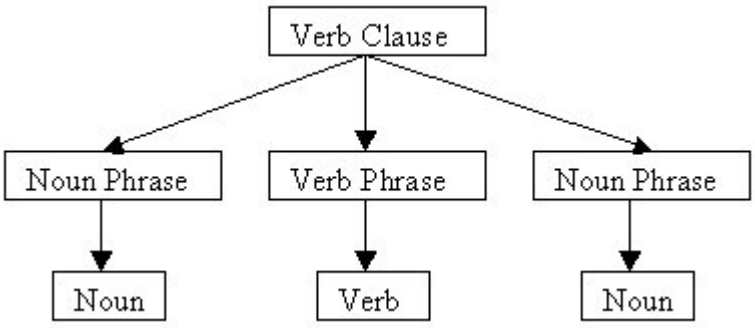

Figure 5.3: verb clause structure

assigns them to compatible nouns from the lexical knowledge base. If no match is found, then a different verb is chosen. This method is not very advanced, and may be changed once further development on the grammar has been done.

The revised phrase structure grammar can be seen in appendix section A.3. As can be seen, the definitions of verbs and nouns have been replaced by constants which denote the type of word that is required to fill the node. The lexical insertion phase then adds nodes to the tree, which are attached to the word objects in the lexical knowledge base. It is done this way so that morphology can be applied to the words in the transformational section, and a link to the word object is required to find the different forms of that word. There are also three different definitions for a verb clause, each requiring a different type of verb. This produces command type phrases, where the verb comes first, primitive action type phrases, where there is no subject of the action, and transitive phrases, where there is an object that performs an action on the subject.

5.4 Transformations

5.4.1 Structural Analysis

As described in section 2.3.4, There are two parts of a transformation: the structural analysis, and the structural change. The structural analysis describes the structure of the tree that the transformation will apply to, and the structural change describes the change in the structure that the transformation will apply. This is a sample transformational rule:
SA: X - en - V - Y SC: X1 - X3 - X2 # - X4

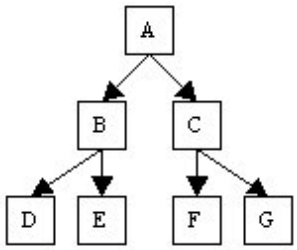

Figure 5.4: example tree structure

The rules are stored as objects, and in a list inside the transformational grammar, they are then checked for in order, and if a match is found for the structural analysis, then the structural change is applied.

The valid structural analyses for the tree in figure 5.4 are the following:
{A},{B,C},{D,E,C},{B,F,G},{D,E,F,G}

That is, all the possible ways of deriving the tree, in any order possible. Furthermore, a match would be found if searching for E,F, or E,C (a subset of the valid analyses). The list of given symbols are matched against the tree in such a way as to obey these rules.

The first node to be found is a node that matches the first symbol in the list, as this will provide the starting point of the sequence. This node can be located anywhere in the tree, and there could be multiple matches of the same rule in a tree, so every node is checked for the first symbol. Once the symbol has been identified, the second symbol is searched for. For a match, the second symbol must be found on the next branch immediately to the right of the first symbol e.g. for E in the example tree, possible second symbol matches are C and F. To find this branch, it is necessary to find the node where there is a branch to the right, immediately above the first symbol node. This node to the right is the top node in a chain of possible match symbols for the second symbol e.g. for E on the example tree, the top of the chain is C, but for D, the top of the chain is E. This chain is worked through by looking at the first child of each of the children of the top node. If a match is found somewhere in this chain, then the same matching process is performed on the next symbol in the structural analysis, but starting with the node that was just identified as a match. If a match is not found for any of the symbols, then the system just returns to searching for the first symbol again.

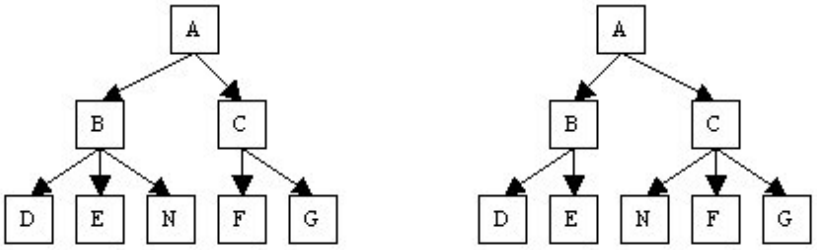

Figure 5.5: different ways of adjoining node N between nodes E and F

The matched nodes are stored in a list, so that they can be manipulated in the structural analysis. The pseudo code for this algorithm can be found in the appendix section A.4. This algorithm is complicated by the fact that it is possible to declare a structural analysis containing an "any node" symbol.

5.4.2 Structural Change

The first problem when designing an algorithm to perform the structural change is working out a syntax that is suitable for describing the change that is required. When inserting the new node 'N' into the tree previously shown to make D,E,N,F,G across the bottom, there are several possible places to add N. The two most obvious ones are show in figure 5.5.

It is important to be able to represent the difference between the two trees in the transformation description; otherwise there is no way to create a general case method for performing the structural change. Although the difference between these trees may seem unimportant, it may have a large effect on the subsequent sections of the program that look for groups of words attached to the same branch of the tree.

Examples:
1. SA: D - E - F - G SC: X1 - X2 - N - X3 - X4
2. SA: D - E - F - G SC: X1 - X2 - N* - X3 - X4
3. SA: D - E - F - G SC: X1 - X2 - *N - X3 - X4

The syntax in examples 1,2 and 3 is adequate to do this job; the first example would insert the node N in-between nodes E and F (presumably attached to the 'A' node, as that is the closest common link). The second example uses a

star to represent a right branch to the X3 (the F) node. This would attach the N node immediately to the left of the F node. The third example uses a similar notation to represent a left branch join to the node to the right.

There are several different operations that individual nodes in the tree can have applied to them:

- Insert: a new node is added to the tree, and inserted in the appropriate position
- Delete: a node is removed from the tree
- Branch Left: a node is attached to the left of the node to the right of it
- Branch Right: a node is attached to the right of the node to the left of it
- Swap: 2 nodes switch places
- Nothing: the node stays in the same place

Because this is a general case algorithm, the action to perform on any particular node is determined by the program, and derived from the structural change that is given. A structure is required to hold more information about the nodes that are to be altered, so that when analysis is done on the structural change, and is compared to the structural analysis, there is a place to put the information that is derived. The information that is stored is the operation to be performed, the original location of the node, the target location of the node, the adjustment that is currently required to get the node to its target location, and a flag to indicate if the node is a new node.

This information is derived by comparing the structural analysis with the structural change, and a list of structures is created that is in the original order, with any new nodes added to the end of the list. This list is then manipulated to re-order the nodes into the correct order, and perform changes to the tree at the same time. It is important to realise that the operations cannot be performed all at the same time, because some operations are inherently linked to each other; they have to be performed in the correct order. Also, operations that are performed will affect the other operations by changing the positions of the nodes involved. It is very difficult to calculate what the 'correct' order is. The best approach to this problem is to make the operations incremental, and perform them

repeatedly until all the nodes are satisfied, or there are no more clear operations that can be performed. The pseudo code for this algorithm is:

```
delete any nodes that need deleting

while (changesAreBeingMade)
{
   perform "insert" where new nodes have been added

   perform any branching operations, new nodes or not

   perform swapping of nodes that are still not in the correct position
}
```

Deletion of nodes is done at the beginning, because the operation does not depend on the other nodes; the same nodes will always be deleted. The insert and branch operations move the nodes into the correct target positions, and examine the surrounding nodes to see if they are the correct target nodes. If the surrounding nodes are correct nodes for the operation to be performed, then the changes are made to the tree, and the operation removed from the node. Otherwise the operation is not performed, and the procedure attempted again when further changes have been made. When the swapping procedure is reached for the first time, the nodes should have been re-arranged enough so that the nodes which need to be swapped "pair up" (each node will point to the others position as its target position). These nodes can then be switched, and the other operations attempted again.

This procedure appears to work for the primary transformations that are required; it may need to be altered if there are any enhancements to the transformational process.

5.5 Enhancements to Verb Clause Generation

At this point the program is capable of generating entirely random-based verb clauses, which are short sentences. This is quite powerful, and can be enhanced

further simply by increasing the size of the dictionary. However, the program lacks some of the functionality that is required for the next phase of the project. To be able to add some semantic connection between the sentences it is necessary to generate these sentences based upon some underlying collection of concepts simply generating random sentences will never allow for semantic level connections. Given a collection of verbs and or nouns, the program should be able to generate a sentence based on those words.

5.5.1 Seeding the Verb Clause

To be able to seed sentences, the structure of the sentence that is produced must match the words that have been provided. The type of verb or noun that is being used must be taken into account; if a sentence structure is produced that is incorrect for the words that are to be inserted, then either the words will not be inserted, or the sentence will be malformed.

The phrase structure grammar section of the program is rewritten in a way that allows sentences to be generated by seeding. Although the random system was very flexible, and easy to alter, it cannot be maintained. Rather than a rule-based, evaluation style of generation, where rules are continuously applied to a tree until it is not possible to apply any more, the program is written in a fixed way, so that the given words can be taken into account at each stage. This follows a more "recursive descent" style of program, where a function is written for each element in the sentence, and the function calls other functions that describe the sub-elements of that element. Each function handles any random application of rules that still occurs, if there are options which are left open. The most important objects in a sentence are the verb and two nouns, so these are the words that are given to the phrase structure grammar, to be the basis of the sentence generation. If one of the words is set to "null" then a random structure will be generated for that element. The given words are inserted onto the tree during the phrase structure generation, and not at the lexical insertion phase.

The main influences that the seed words have over the phrase structure of the sentence are: the type of verb (noun-first, transitive or noun-last), and the types of nouns (proper or common). When deciding which elements are in the sentence, the type of the verb is crucial. The verb rules are examined to find out which type of structures the verb uses, and an appropriate one is selected. How many

nouns have been provided is also taken into account at this stage; if two nouns have been provided, then a transitive rule is chosen, or there will be no space for them both. The type of the nouns specify if a determiner is appropriate or not.

The lexical insertion is also altered to reflect these changes, as now it is possible for nouns to be fixed, and a verb to be variable. Verb rules are chosen more carefully.

5.5.2 Tenses

The tense of a verb clause that is produced is important because to generate two connected sentences they must be of the same tense, in order to make sense. The three main tenses are past, present and future. It is possible to specify the tense when a verb clause is generated. This allows for the higher semantic layer to have more control over the form of sentence that is generated, whilst still not being concerned about the details of syntax.

The tense is tied to the phrase structure of the sentence and the form of the verb that is finally produced. Although the form of the verb is not directly manipulated in the Phrase Structure Grammar, the insertion of the symbols '[en]', '[ing]', '[p]' and '[pr]' after a verb changes its form when the transformations are applied later on in the process. The structure generated is classified by tense, and the phrase structure process altered to produce a structure of the tense specified. The classification of tense by patterns of auxiliary verbs is as follows, where M = Modal verb, and V = Verb:

Past: M have V [en]
M have been V [ing]
Has been V [ing]
Has V [en]
V [p]

Present: M be V [ing]
Is V [ing]
V

Future: M V

When the object noun is singular and the tense is present, without auxiliary verbs, then the verb takes on a new form, which is the singular present form. Because this case cannot be detected until quite late on in the process, this rule is implemented in the transformational phase. The node [sing] is inserted after the verb to indicate that it needs to be changed into the singular tense.

5.5.3 Irregular Verb "to be"

Most verbs are quite similar in use, and are represented in verb rules of the form $nounType + preposition + nounType$, where all three components are optional. This is captured by the verb-rule class, which was created to store information in the lexical knowledge base. However, the "to be" verb needs more specification than that, because it is not correct to have sentences of the form:

Noun [sing] is Noun [pl] e.g. the man is the monkeys

A singular object cannot be described as a plural object. To be able to express this concept, the verb rule class includes the "quantity" of a noun as an optional part of the rule. If the quantity is known, then the correct form of the noun can be inserted at the lexical insertion phase. The quantity of a noun also affects the determiner that can precede a common noun, "a" and "the" can precede a singular noun, but only "the" can precede a plural noun. This means that the determiner can only be specified once the quantity is known, so the insertion of determiners takes place in the lexical insertion phase, not the phrase structure phase.

Another irregular form of the verb "to be" is the plural present tense form "are". Most other verbs use the infinitive form for plural present tense such as "they eat" and "they go", but "they be" is incorrect, instead it is "they are". The definition of a verb does not allow for this irregular form, although the definition could be expanded to include the irregularity, it would have to be defined for all the verbs. Instead, to include the form "are" a transformation is used that inserts it when the tense is present and the object noun is plural. It is difficult to create a rule at an earlier stage, because the quantity of a noun is not known until after the lexical insertion phase.

5.5.4 Adverbs

Adverbs often set the tone in a sentence; the action that is being performed is given an emotion by the adverb. In order to be able to control what type of emotion is applied to the clause, a parameter is sent to the grammar to specify what type of adverb to insert (if any). The adverbs are classified into 5 categories:

- POSITIVE e.g. happily, enthusiastically
- NEGATIVE e.g. angrily, reluctantly
- CONTENTIOUS e.g. nearly
- TIME e.g. quickly, slowly
- OTHER e.g. carelessly, carefully

Selecting one of these types helps to set the tone of the sentence; these categories are generated by a higher semantic layer, and sent to the phrase structure grammar.

5.5.5 Modals

Initially modal verbs were specified at the phrase structure phase. This meant that they were chosen at random, and had little to do with the rest of the sentence. To be able to control these more closely, they are classified and a parameter sent to the Phrase Structure Grammar to indicate which style of sentence should be created.

DEFAULT this returns the minimum modals; using "will" for future tense, and using no modals for the other tenses

DEFINITE modals mean that the sentence is definitely going to happen like "will", "shall", "must"

INDEFINITE modals mean that the sentence is not definitely going to happen, like "might", "could", "may"

5.5.6 Negations

To be able to have full semantic control over the sentences generated, negation is specified independently from the standard compulsory transformations. To achieve this separation, the negation rules exist in a separate class from the standard ones. This class extends the transformational grammar class, and overrides the add-rules method. An instance of this class is created, and the negation rules applied when required to a phrase grammar, in addition to the standard transformation class.

5.5.7 Adjectives

Initially, whether or not the phrase grammar inserts adjectives before the nouns was random in the system. The adjectives are now categorised by the type of noun that they can apply to, so no inappropriate adjectives can be selected.

Example:
The red ideas

There are some situations where having adjectives is not appropriate, for example when creating a clause to say what day it is, and there are also situations where an adjective is necessary, like when describing the difference between two objects. In order to be able to specify when an adjective can appear, a parameter is sent to the phrase structure generator which can be set to true, false, or random to specify whether an adjective is compulsory, not required, or to choose at random to insert one or not. Further control may be added by categorising the adjectives into types, so an adjective type of "colour" could be specified to add a colour to an object. However so far that level of control is not necessary.

5.5.8 Nouns and Pronouns

There are some nouns that are always singular or always plural. Words like "clever" which is always singular need to be identified because the quantity of a noun can affect the form of the verb that is used. There is a field in the noun class that can be set to singular, plural, or any, to specify that a noun is always singular, always plural, or has both forms. The "quantity" of a noun is looked at in the lexical insertion phase, before the insertion takes place, to ensure that

it agrees with the verb rule that is being applied.

Pronouns help to break up the use of a character in the text. Repeating the characters name every time an action is being performed becomes very hard to read. Because the system does not use 'I', 'you', or 'we' forms, there are essentially four different types of pronoun each that has three different forms:

He, She, It, They
Him, Her, It, Them
Himself, Herself, Itself, Themselves

The first form is the most useful, because they are substituted for the object noun. The second form is more difficult to implement because it describes the second most recent noun that has been used, and the third form is only used when the subject noun is the same as the object noun, so is simpler to implement. Essentially inserting pronouns can be performed at the lexical level, by keeping track of the character that has last been used, and inserting the appropriate pronoun if that character is immediately used again. However, the implementation of pronouns requires character objects to be introduced into the system.

5.5.9 The Third Noun

During the development of the system, it became clear that the original phrase structure was not powerful enough to produce all the clauses required. There were some things that could not be expressed properly:

Examples:
X asked Y about Z
X stole Y from Z
X played Y with Z

The ideas that are conveyed in these phrases could not be captured in the original verb clause phrase structure. This was not good because part of the specification of this project says that the phrase structure should be powerful enough to capture as many different ideas as possible, and the program was clearly falling short in these examples. To solve this problem the phrase structure was changed, so that another noun could be added to the verb clause. It also required an ad-

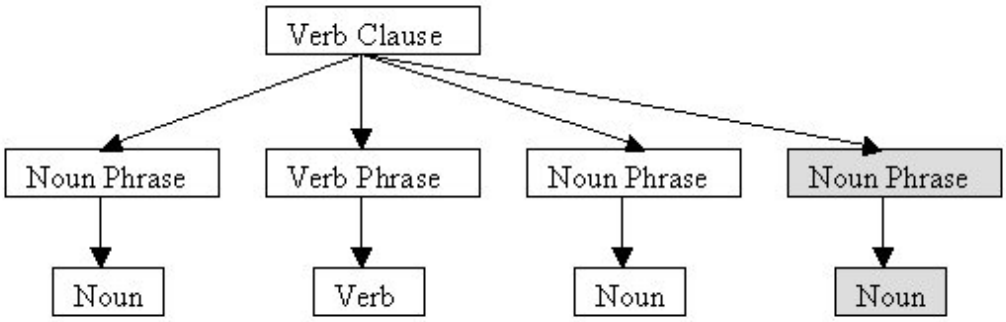

Figure 5.6: the new verb clause structure

ditional preposition to be inserted in between the two noun phrases. Figure 5.6 shows the new verb clause structure.

To allow for this change a new verb type was added and the verb rule class altered so that it accepts rules of the new type. The lexical insertion phase was also changed to make it capable of inserting three nouns, instead of two. The algorithm that selects verbs based on the various nouns available was changed so that it considers the third noun when making the choice. Making the necessary changes ensured that the power of the clauses were greatly increased, and helps to enhance the final output of the program.

5.6 Creating Fundamental Clauses

The second part of the system is concerned with the semantic connections between sentences that are generated. The types of clause that can be generated are classified, and these classifications are used in the semantic layer of the system. The fundamental types of clause are:

- Descriptive:
  X has features
  X is [animate / job/ description]
- Active:
  X does some action
- Passive:
  X does some passive action

It is useful to make a distinction between active and passive actions; this helps to control the flow of action much better. Passive actions are actions that do not directly affect the other characters, like sitting, standing or walking. Introducing passive actions to "pad out" the actions slows the story down, and makes it more realistic.

- Interactive:
  X does something to Y
  A sentence where two characters can interact

- Emotions / Relationships:
  X thinks ...
  X does not like ...
  X hates/loves Y

- Approach / Entrance Clauses:
  X enters the room
  X approaches Y

- Settings:
  It is raining
  It is Tuesday
  Other time-based clauses like:
    Later on ...
    The next day

These different types of clauses are used to build up larger portions of text. Knowing what type of clause to generate next and being able to send parameters to create that clause are the building blocks of the semantic part of the system.

5.7 Characters and Locations

To be able to write about different characters in a story, there is a special class called character. A character has all the attributes of a noun, and sits in place of a noun in a clause, so it makes sense that the character class be a descendant of the noun class. A character also has special properties within the system, such as location and emotion. Characters are reoccurring nouns that link clauses together and actions that take place involving characters change the attributes

associated with them, e.g. adding descriptions to the character as they are generated or changing affinities.

Locations are also noun-based, they are the places in which action is happening and therefore are also reoccurring nouns as far as the "scene" in the story extends. Actions can also be linked to locations, so that actions that are more likely if the characters are in a particular location can be specified. Therefore, locations are also descended from nouns, but have other special properties.

5.7.1 Special Character Properties

A character is able to store the location that the character is currently in. This means that it is easy to make sure two characters are in the same location before creating interactions between them. It also enables the proper generation of "entrance" and "leaving" clauses where appropriate, and ensures that the program obeys the semantics of these clauses.

A character stores the type of entity that the character is, such as human, male, female, or some kind of animal, depending on what has been specified. Keeping this information is important because some of the clauses generated need to know the type of the entity, and inserting the correct pronoun also requires knowledge of the type. Characters are also capable of storing the current action that they are performing. This is useful because it is used to prevent characters doing the same action twice and also to help determine what the next action might be, allowing other characters to "join in" with the action that is being performed. Values for the current "active" action, the current "passive" action, and the character(s) that the action is being performed with, are all stored. This means that the program knows two different actions that the character is doing at the current moment, and this is used to help determine the next action.

5.7.2 Character Emotions

An important factor in deciding what a character should do next is the emotion of the character. Emotions can change over time, depending on what actions are performed on, or by the character, therefore it is a dynamic factor that helps give the characters suitable actions to perform. So that an action can be chosen based on an emotion, the actions are all assigned emotional attributes, so that correct

actions can be matched to the feelings of the characters at any point in the scene. To do this the property "emotion" is associated with every verb object, as part of the lexical knowledge base. Some actions have no particular emotion attached to them, and are given a "neutral" value.

There are five different emotions that have been added to the system:

- 2 - Very Happy
- 1 - Happy
- 0 - Neutral
- -1 - Sad
- -2 - Angry

If the size of the dictionary was significantly increased then a larger number of emotions may be required, but currently there are insufficient verbs to allow for many emotional states. When an action is performed the emotional value of that action is added to the emotion of the character. So that characters do not get stuck on one emotion, it is possible to perform actions of plus or minus one from the current emotion, and it is always possible to perform neutral actions, because there are a number of actions that have no applicable emotion.

Closely related to character emotions is the concept of character relationships, which are the emotions between two particular characters. When performing an interaction between these two characters, their relationship is taken into account. This allows for different characters to have different relationships that build up as the story progresses. Significant events, like character goals, have a bigger impact on character relationships than standard actions.

5.7.3 Special Location Properties

Depending on what location a character is in, they are able to do different things and interact with different elements of the location. There are several different properties that a location has:

- Special nouns
- Special verbs

- Special characters

Special nouns are essentially the scenery in a particular scene, characters can interact with the scenery and this makes the location seem more "real". Special verbs are actions that characters can take because they are in that location, like buying items from a shop, or ordering food at a cafe. Special characters are characters that only appear in that location and help the action along, like a bar tender or shop keeper. At any time the verbs and nouns can be chosen from the global lexical knowledge base or from the special location-oriented set, this means that common actions are only defined once. The combination of these three properties help to make sure the locations play a role in determining what the characters are doing and what they will do next.

5.8 Scenes and Interactions

Once clauses were classified into various types, different clauses could be generated and output at the same time to form larger blocks of text. This text has to be semantically linked in some way or it would not make any sense. To do this it is controlled by a higher class that is aware of concepts like characters and locations within the system.

The first attempt at creating linked clauses involved very simplistic character and location lists, along with a fixed pattern of types of clause that could be generated for each "scene". This had limited success, see Appendix section A.5 for sample output that used this technique. It did create linked clauses, but the structure was very rigid, and the actions that took place were not related to each other.

In order to get more dynamic scenes to occur it was necessary to find an alternative to fixed scene structure. It is also useful to have the action history stored, so that previous actions can be taken into account when deciding what the next action should be. A scene is defined to be all the action that takes place in a certain location therefore, it is tied to the properties of the location, and the actions that occur are also linked. To be able to break the fixed structure of a scene, different types of clause are assigned different probabilities of occurring, which change as each new clause is generated, and the history of the scene evolves. There are three different stages to a scene:

- The Beginning
  Descriptions of the location
  Characters enter
  Actions of small significance

- The Middle
  Actions and reactions
  Possibility of more descriptions

- The End
  Characters leave
  Some "ending" actions

At each stage the probabilities of different things happening are different, and as each stage progresses the chance of going to the next stage increases. The boundaries between these stages are made fuzzy by allowing for characters to enter and leave in the middle stage, which breaks up the actions more. The concept of actions and reactions is quite a powerful one. Once an action has been performed, a reaction to that action can be calculated by looking at the changes in the characters due to the action, and by directly reacting or by altering the probabilities of the next clauses being generated.

A "story" class that contains several scene objects and a list of character objects, controls the final story. Different scenes are created using different locations, with different actions and different probabilities associated with the various types of clause. The first and last scenes are special, the first being when the characters are created, and first enter the story and the last being where the characters exit the story and something significant happens to them.

5.8.1 Interactions

The idea of splitting a scene into three different sections worked reasonably well, the order of the actions created was chronological, and things made more sense than just randomly picking sentences. However, the actions that were being generated in the middle part of the scene were still random actions, with no real connection to things that preceded or followed. Introducing another stage called an "interaction" is an attempt to try and manage the flow of the actions better.

An interaction is a meeting between two characters which is multiple sentences long. It starts with one character approaching the other and ends with one of the characters leaving. In the middle there are various passive actions, active actions, and interactions, which are controlled by the relationship between the two characters that are interacting. Interactions take place during the middle stage of a scene, and the characters that interact are chosen from the list of characters that have entered the scene in the beginning stage.

5.8.2 Different types of Scene

There are different types of scene which are built up to make a story. A beginning scene where characters are introduced, and mainly passive actions take place, and middle scene where more actions take place, and an ending scene where more significant things happen. There is also a summary paragraph where the story gets summarised, and concluded. These different types of scene are created by making a new class, which inherits from the original scene class, and changing the weights attached to the various clauses in the different stages of the scene. However, the finalising paragraph is significantly different from the other scenes and has a completely different structure. It does not inherit from the base scene class and the clauses generated are based upon the final state of the characters that remain in the scene and the relationships between them. See section 5.9.2 for further information on the final paragraph.

5.8.3 Dynamic Allocation of Scenes

Rather than having static number of scenes, beginning then middle then end, the different types of scene are dynamically generated, so that there can be many different scenes in the final story. The story always begins with one or more "beginning" scenes, and the same characters taken from those scenes into the middle scenes. The number of middle scenes is random; it depends on how the story is progressing. If many important events have taken place, then there are fewer scenes than if the story is fairly slow moving. This allows for each story to have significant events, the story being as long as it takes for things to unfold.

5.9 Character Goals

The output for the program at this point can be see in appendix A.6, the story is beginning to take shape, it is no longer a random selection of sentences and there is some "flow" and relation between them. However, it is still just a collection of pointless actions and there is no real direction to the story, because none of the characters have any purpose; they are just performing arbitrary actions.

The introduction of character goals is an attempt to try and make the story more realistic, giving the characters something that they want to achieve. This also adds another factor to the emotions of the characters, making them happy if they achieve their goals, and sad if they do not. The only goal that is in the system is quite simplistic; simply the desire to own an object. Each character is generated with an object that they want to posses, and during interactions they try to obtain that object. This means that another character will have the object, and the character will have to try and persuade them to hand it over, or take it from them. This concept helps to control the actions that are created, and give them more meaning.

5.9.1 Special Events

A character goal is a significant event, when a character achieves their goal, it has a large effect on their emotions and their relationship to the person that helped them. If a character is denied their goal, then it will have a negative effect on their emotions and they may seek revenge on the person who denied them. These are special events in the system; the other main special event is when one character kills another. This will only happen when one character is very angry with the other, and it is significant because it means that the character that is killed cannot perform any further actions, and will not take place in the remainder of the story. The deceased character is added to a list of "old characters" and kept until the end. If a character kills another, then from that point onwards it will be possible for the police to enter the story and try to arrest them. If a character is arrested then they will also be added to the old characters list, and cannot perform any further actions in the scene.

5.9.2 Final Paragraph

As introduced in section 5.8.2, the final paragraph is a small summary of what happened in the story, and a final action for the characters that are involved. It is designed to try and give a some kind of "closure" to the story, instead of ending with everybody leaving the final scene. To be able to output details of the significant events in the story, the program has to look closely at the characters that are left in the story, and work out what has happened to them. There are several different possible ending states for characters:

- Killed someone and in prison
  This means that a character was killed, and then the police arrested the killer, so both characters are in the "old characters" list.
- Killed someone, then got killed
  A character was killed, but then the killer was also killed before they were arrested, so both characters are in the "old characters" list.
- Killed someone and on the run
  A character was killed, and the police have failed to arrest the killer.
- Love
  Two characters are in love with each other.
- Friends
  Two characters are good friends with each other.
- Hate
  Two characters hate each other.
- Goal Summary
  Failing all the other options, a summary of whether the character achieved their goal or not will be output.

By checking for these cases in this order, the program works out what to output about a character to summarise their activities. Characters that have been killed are only mentioned as part of the killers ending state, as they cannot have any further actions. Characters are only ever mentioned once in the final paragraph, the program starts with the list of all the characters, and removes them as they are mentioned. This avoids writing X loves Y, and then later Y loves X, because

both X and Y are removed from the list the first time. It is possible for a character to not be mentioned in the summary, if they have not tried to obtain their goal at any point then nothing will be printed for the final case. However, it is very difficult for there to be nothing to be said in the final paragraph, because of the dynamic scene allocation, the story will continue further if no incidents have occurred.

Once the ending state has been determined for any particular character then a sentence is output describing this state, and a further sentence to describe what a character is doing at the end of the story. This provides a more open ending for the story, and allows the characters a chance to do one last action.

5.10 Expanding the Dictionary

The final task that was done was to expand the dictionary that the story is generated from. This gave the program much more expressiveness, however a larger possibility for randomness could have resulted in less realistic stories being generated. The words had to be added into the correct places, the correct categories and types; otherwise the final program would produce incorrect clauses. There are five main types of word that are in the dictionary, each with different properties that have to be set when adding new words.

- Names:
  It was simple to add more names to the program; names are simply stored as a list of strings, specifying male or female for each. There are no significant restrictions to increasing the volume of possible names in the system.

- Adjectives:
  These are words that describe nouns, when adding a new adjective a category of noun must be given that the adjective applies to. Applying an adjective to an incorrect noun type will not make sense, so care must be taken when adding adjectives.

- Adverbs:
  These are words that are added to the verb to give the clause some kind of emotion. Each adverb has an adverb category associated with it, if the wrong category is chosen, then it may be used in the wrong context.

- Nouns:

  There are many different types of noun in the system; they are used to ensure the correct nouns are chosen when using a verb. If a noun is put in the wrong noun category then it may be used incorrectly. Nouns also require the specification of singular or plural forms, and whether the noun is a common noun or a proper noun. It is also possible to specify that a noun is always plural or singular.

- Verbs:

  Verbs are much more difficult to add than the other words. There are five different verb forms that have to be specified, a type to be specified, and a field that indicates which emotion applies to the verb. More importantly, for a verb to be used in a clause, each possible usage must have a verb rule associated with it. Verb rules are added to the verb and specify the valid combinations of noun types and prepositions that can be used. It is the creation of verb rules that makes them difficult to add to the system.

Chapter 6

Testing

The system can be viewed in two parts, the first part produces syntactically correct sentences, and the second part produces semantically correct text, which comprises of the output from the first part. Because of the nature of this system, it is quite difficult to create a concrete testing strategy for the final program. The only tangible output is the story that is produced. The requirements state that the system must generate "correct" English at all times. The test plan is to enter the text into a grammar-checking program to verify that the output is syntactically correct, but formal verification of the semantics of the text is almost impossible.

Verification that the final output is correct is just one element of the system; most of the other requirements focus on the different aspects of the program, and the way in which they work at a lower level. Testing was done as the system was developed on each of the components as they were added to the program. Most of the testing involved producing elementary output and analysing it for any syntactic errors. If errors were located then the program was altered to correct the problem, and the output retested. It was often necessary to focus on particular parts of the system, by altering the probabilities to force the program down a specific route.

6.1 The Grammar

The specification states that a grammar must be created to represent a sentence, and this grammar must be used to create the textual output. Initially

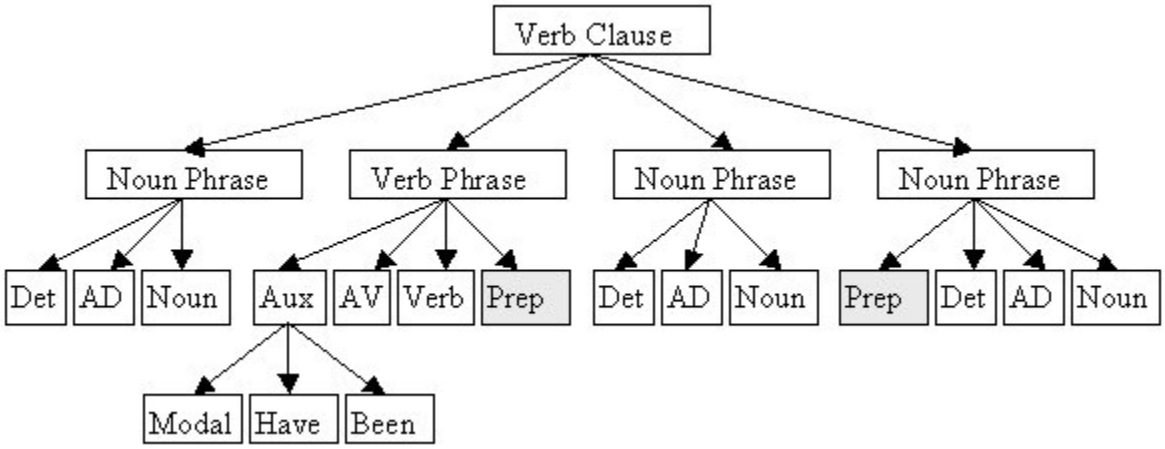

Figure 6.1: the structure of the final grammar where, Det = determiner, AD = adjective, Aux = auxiliary verb, AV = adverb, Prep = preposition

the grammar rules were stored in a grammar structure within the phrase structure grammar class, however later this was altered so that seeded sentences were possible. This means that the grammar is stored in a fixed format represented by different functions. Figure 6.1 shows the full the Grammar; the prepositions are added at the lexical insertion stage, once a verb rule has been selected. All the other nodes are optional, although a noun phrase must have a noun, a verb phrase must have a verb, and there must be at least one child of the auxiliary node. There is always a verb phrase, and at least one noun phrase in a verb clause.

Testing whether or not this grammar fulfils the specification is not very easy. It can be shown that all the sentences eventually produced are "correct". There are five sample output stories in the appendix section A.7, and all of the sentences are valid, have no errors when passed through a grammar checker, it is therefore reasonable to conclude that the structure is valid for English. The grammar is capable of generating a range of different structures, all derived from the base structure; because of this it is quite expressive.

Examples:

1. The shopkeeper drinks
2. Holly was angry
3. Emma punches Holly
4. Harry swiftly leaves the bar
5. The park keeper walks by the river.
6. Connor asks Lucy about the string

These different examples, which are all taken from the sample outputs in the appendix section A.7, demonstrate the expressiveness of the grammar. Example 1 is a very simple clause with one noun and one verb, but example 6 uses three different nouns and a preposition to construct a more complicated sentence. The grammar certainly does not cover the entire of the English language, but that was not the aim of the project. Whether the expressiveness of the grammar is sufficiently large to capture a broad enough range of concepts is not an easy question to answer. However, the variety of different sentences that can be produced, and the fact that it is possible to get the kind of outputs shown in the appendix, indicate that this simple structure is sufficient for the scope of the project.

6.2 Lexical

The specification states that there should be a dictionary of words, which are stored in a well-structured way, and that these words should be inserted into the structure that is created by the grammar. The lexical knowledge base stores almost all of the words that are in the system. The four main categories of word are verbs, nouns, adjectives and adverbs. These are all stored in the lexical knowledge base, and methods are provided that allow the other areas of the system to get words based on category, type, or quantity where applicable.

Testing whether something is stored in a well-structured manner is quite difficult to do. The verbs, adjectives and adverbs are all stored in objects, and those objects stored in lists. Methods exist which allow the words to be created or added in one line, and methods exists that select words from the lists based on given parameters. Nouns are stored in a different manner, there is a tree of different types, and nouns are stored attached to the node of the tree that corresponds to their type. It is done this way because of the way in which nouns must be selected to match with verb rules. Similar methods exist to help add and select nouns. The words must be stored in a well-structured manner because otherwise selecting a word from the lexical knowledge base would be very difficult, and the rest of the system would have to be more complicated to compensate. However, the selection and addition of words are trivial operations, therefore the structure of the lexical knowledge base is sufficient for the system.

The lexical insertion phase takes words from the lexical knowledge base, and inserts them into the grammar that was produced by the previous phase. This is essentially what the specification states; to take words from the lexical knowledge base, and insert them according to the rules that are attached to them. The rules are verb rules, which are attached to each verb in the system, and define what types of noun can be used with the verb, and which prepositions are applicable. The psuedo code for the selection algorithm is shown below. It shows that the verb rules of the selected verb are used to determine which nouns to insert into the phrase structure. This satisfies the requirements of this part of the system.

```
While (nounsNotFound)
   {
     nounsNotFound = false


     Verb v = findAVerb(givenClauseType)
     VerbRule vr = v.getRule(givenClauseInformation)


     If noun1 is needed
          Try to find a noun that matches vr.noun1Type
          If noun1 is not found then NounsNotFound = true


     If noun2 is needed
          Try to find a noun that matches vr.noun2Type
          If noun2 is not found then NounsNotFound = true


     If noun3 is needed
          Try to find a noun that matches vr.noun3Type
          If noun3 is not found then NounsNotFound = true
   }
```

6.3 Transformational Grammar

The transformational section of the system is responsible for altering the tense of clauses, and changing the verb forms to make sure they are correct. There are

also several compulsory transformations such as affix hopping, and optional rules such as negations. The specification states that a transformational grammar must be applied so that different sentences can be created from the same underlying clause. When verbs are inserted into the phrase structure, the infinitive is the default verb form. The verb form is only changed in the transformational phase of the program.

Examples:
7. They sit on some chairs
8. He sits on a chair
9. He is sitting on a chair

These examples are all taken from the sample output in the appendix, and demonstrate that the transformational grammar is changing the form of the verb, depending on the tense of the sentence, and the types of the nouns that are used around it. In example 7 the verb is in infinitive form, but in example 8 it is in 3rd person singular form, and in 9 it is in imperfect form.

Examples:
10. Adam gives her the sandwich
11. Emma doesn't give her the banana

These two examples demonstrate the use of the negation transformations, which change the meaning of the sentence to mean the opposite. In this case it inserts the word "doesn't" into the example 11, and changes the form of the verb to the infinitive. The rule that performs this task is shown below.

addRule(new String[] "NS","VP", new String[] "0","doesn't*","1");

Transformational rules have two parts, the structural analysis, and the structural change. In this case the structural analysis is "NS","VP", which means it is looking for a singular noun, followed by a verb phrase, and the structural change is "0","doesn't*","1", which means it will insert the word "doesn't" in between words 0 and 1. This implementation of transformational rules satisfies the requirements because it allows for different sentences to be generated from the same clause structure, and it uses a compact rule representation, which allows for easy modification and addition of rules.

6.4 Linked Sentences

The specification states that the program must be enhanced to produce many sentences at the same time, and to use character objects to store information to link the sentences together. It is clear to see from the sample outputs that many sentences are being produced simultaneously however, the semantic links between the sentences are not as obvious.

Examples:
12. Samuel wants some pears
13. Lucy has a pear
14. Samuel wants some pears and Lucy has a pear

In these examples, two clauses are generated which have the noun "pear" in common, and then they are joined with the word "and" in example 14. The facts that "Samuel wants pear", from example 12, and "Lucy has pear", from example 13, are then attached to the character objects, and the a few sentences later this allows for:

Examples:
15. Samuel asks her for the pear
16. Lucy doesn't give him the pear
17. Samuel asks her for the pear but Lucy doesn't give him the pear.

Because the information is stored against the two characters, the character Samuel is defined as wanting a pear, and he knows that Lucy has one. He makes a request in example 15, and depending on the relationship between them, she can accept or refuse. In this case it is a refusal in example 16. These two examples are then combined using the word "but" in example 17. The information about the interaction is also stored, so now the character Samuel knows that he wants a pear, Lucy has a pear, but she will not give it to him. This leads onto the next example:

Example:
18. Samuel steals the pear from her and then he leaves the cafe.

Example 18 demonstrates some of the semantic links between the sentences, and how the system uses the character objects to store information about events and

facts that have happened. This works well for small pieces of text, but when creating larger amounts of text it is more difficult to maintain a "flow" based on this kind of simple information. Other information is stored, like the location that the action is taking place, and who is currently in the scene to help focus the action, rather than rely on random actions to generate interesting scenarios. In the next example the noun "park" is used to link the clauses together, and the fact that the characters are in the same location allows them to interact.

Examples:
19. Emma is in the park
20. Holly enters the park
21. Emma goes up to Holly

After the examples 19 and 20, both character objects Emma and Holly have their locations set to "park". This allows for the example 21 occur, where an interaction begins between the two characters. The semantic connection between these clauses is clear to see, and adding concepts like locations to the program helps the "flow" and allows for more text to be produced.

6.5 Non-Functional Elements

There are several non-functional requirements in the specification: the system must produce text when the button is pressed, and that text must be selectable so that it can be copied into a grammar checker. This can be observed by running the program; Figure 6.2 shows the interface. When the "go" button is pressed, the output is produced into the text box that covers most of the screen. This text can easily be selected and copied to a different application. This satisfies the requirements.

There is no facility for the system to produce errors, whatever occurs; the text is always output to the screen. This is what is required because even if there are errors in the output, it must be displayed so that the problem can be resolved. The final requirement is that the dictionary can be altered by future development to increase the power of the system in a simple way. The dictionary is stored in the lexical knowledge base in a well-structured and simple manner. This can be edited in a straight forward way because the entries are compact, and helper

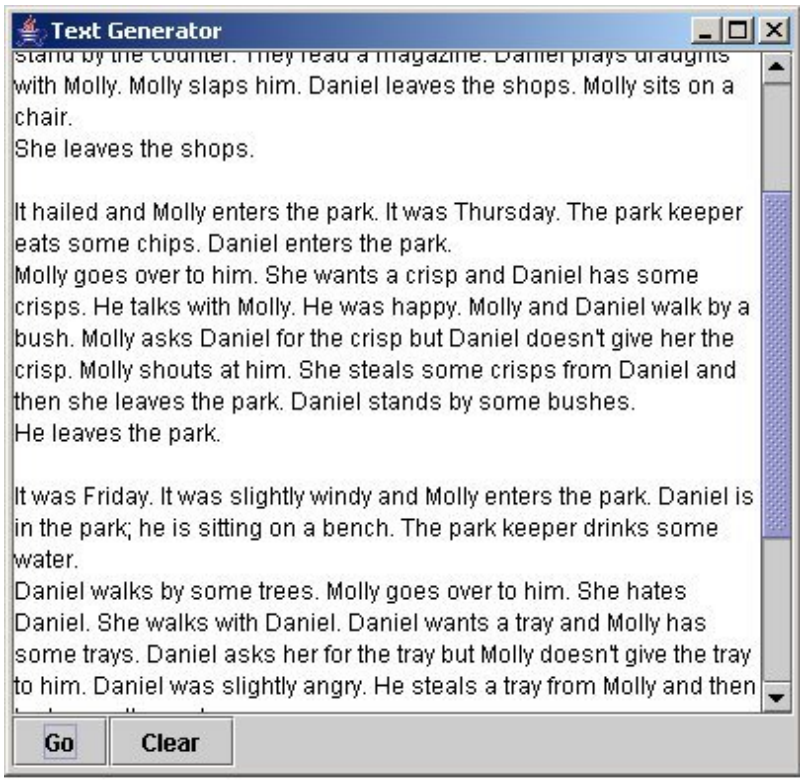

Figure 6.2: the interface

functions are provided to add and select words. This satisfies the requirements of the system.

Chapter 7

Critical Evaluation

A large amount of the project was created in accordance to the original plan, and works well. Not everything was as straightforward as originally intended however, alternative solutions were created, and the finished system was completed in good time and fulfills all of the requirements. This chapter evaluates each area of the system, and outlines the benefits and drawbacks of final program.

7.1 Phrase Structure Grammar

The original plan for the phrase structure grammar was to have a very flexible grammar to represent the structure of a sentence. Being able to alter the grammar easily meant that the grammar could then extended as the project progressed, and the power of the sentences enhanced as improved solutions became apparent. It was not possible to develop the grammar fully until a significant portion of the system was complete, because the text needs to go through the lexical insertion, and transformational phases before it can be output. Therefore, these areas had to be complete before full development of the grammar was possible. The initial phrase structure grammar was developed according to this plan and the solution was good, because it was very flexible, and required little input from an external class in order to generate a tree.

Sample Grammar:
addRule(“S”, new String[]“NP”,“VP”,“NP”);
addRule(“NP”, new String[]“D”,“CN”);
addRule(“NP”, new String[]“D”,“AD”,“CN”);

addRule(“NP”, new String[]“PN”);

This sample was taken from the original phrase structure grammar; it is very easy to construct a phrase grammar using rules like these. The first string is the initial symbol, and the array of strings represents the rewrite symbols. The program would automatically organise the rules, and then select applicable rules at random, when requested, to create a terminal string in the language that the grammar describes. This was very extendable, reliable, and satisfied the requirements.

The original version remained whilst the rest of the sentence generation phase of the project was developed. However, when beginning the second phase of the project, generating multiple semantically linked sentences, “seeded sentences” were required. The sentence structure needed to depend on some optional given words. If a sentence that uses a transitive verb is required, then the system must generate the phrase structure for a transitive sentence. If the structure is chosen at random, then there is no guarantee that it will match the given verb. This problem meant that the phrase generator had to be completely rewritten into a recursive descent style of program, where each element in the grammar was written as a function. Parameters could then be passed to each stage of the generation, and taken into account when deciding what structure to create. This sacrificed a lot of the flexibility of the older version, because altering a grammar that is stored as functions is much more difficult than altering a rule-based grammar. However, the grammar had already been developed to a good standard, so it was not necessary to alter it significantly after the system had been changed.

Ultimately the phrase grammar section of the system ended up with a lot of parameters that are passed to specific elements of the grammar, to control what is generated. This allows the higher semantic level of the program to have a good control over what is created, whilst still not being concerned about the actual syntax of the sentence. When parameters are left blank, then the grammar randomly chooses what to generate, so the semantic level does not need to know every detail of the sentence that is being created. This system lacks the flexibility that was initially intended, because it is not rule-based, but has the power that is required to allow the semantic layer to create sentences, because of the parameters.

7.2 Lexical Insertion

The lexical insertion phase is designed to insert words from the lexical knowledge base into a tree that has been produced by the phrase structure grammar. Although this is essentially a trivial operation, it was difficult to design a generic way to perform this task because of the links between the words. In a clause, the verb defines what types of noun can precede and follow it. This means that when inserting a noun, the verb to which that noun "belongs" must be known. Therefore the lexical insertion phase must have knowledge of the structure of the grammar, so that it can decide how the nouns relate to the verb. Knowledge of the grammar is also used when inserting prepositions and determiners.

Because the lexical insertion algorithm is aware of the structure of the phrase grammar, whenever the grammar is altered significantly, the lexical insertion must also be altered to reflect that change. This is not ideal because it means that the lexical insertion is not a general algorithm for insertion, but fixed for each possible phrase structure grammar. It may be possible to merge the phrase grammar with the lexical insertion, so that any change would be contained in the same method. However, this would break the encapsulation of the system, and make the final method more complicated. Alternatively it may be possible to allow the lexical insertion to be aware of the structure of the grammar, in such a way that any changes would be automatically picked up when searching for words. However, this would be quite difficult because the grammar is stored in a recursive descent style, rather than rule based.

Despite its limitations, the lexical phase insertion performs the task that it is supposed to do. Once an analysis of the tree is performed, and the words located, it selects a suitable verb from the lexical knowledge base, and matches the nouns around it so that the sentence will make sense. This is done in a reliable way, so that the remainder of the system does not need to worry about getting the correct words for the sentence.

7.3 Transformational Grammar

The transformational phase of the project is designed to perform transformations on the final tree so that verb forms can be changed, and different sentences can

be created from the same clause structure. There were three different problems to be solved when creating this part of the system: Creating a compact representation of a transformational rule, creating a structural analysis algorithm, and creating a structural change algorithm.

The representation of the rules is important because the transformational section is designed to be as flexible as possible. This means that it has to be easy to see the rules that are currently being used, and to add or amend them. It is quite difficult to make a simple representation for the rules, because they contain a lot of information.

addRule(new String[] “NPL”,“VP”, new String[] “0”,“doesn’t*”,“1”);

Using the ‘*’ symbol to represent the direction to adjoin is a key concept in the design, and greatly increases the amount of information that can be contained in a rule. The representation that is in the final system is good because it contains all the information necessary, in a compact manner.

The algorithm to search for the structural analysis is essentially a tree walk algorithm. It looks through the tree to find the first symbol, and then walks to the right to try and find matches for the other symbols. The algorithm was written according to the original ideas, and works well. Although this is not a complicated algorithm it is very important, because the whole transformational process relies upon being able to identify where there is a structural match.

The structural change algorithm is more complicated than the structural analysis algorithm. To make the system as flexible as possible the structural change algorithm must handle all possible kinds of structural change. There are four fundamental structural changes that can be performed on the nodes: delete, insert, branch, and swap. The algorithm works out where these operations need to be performed to get from the structural analysis, to the structural change. This is a complicated procedure involving rechecking the tree every time an operation is performed, in case more operations have been made possible. This algorithm is good, because it deals with general case transformations, making the whole transformational section more powerful. It may be possible to enhance this algorithm to handle more types of operation, but it is sufficient to allow all the required transformations to be created.

7.4 Lexical Knowledge Base

The lexical knowledge base is designed to hold all of the words that the system needs to create sentences with. It also contains all of the semantic information associated with those words, such as category, emotion, and verb rules. These words and rules are all stored in lists of objects, and there are methods that select from, and add to, these lists. This works well, and is a good system. However, in the original plan the lexical knowledge base was to become an external data source, so that it could be more easily modified.

If the lexical knowledge base was moved to some external data source, it would become much harder to alter the way it was stored, and add extra information. It would almost certainly become more complicated, especially if it was put into a relational database. Therefore, ideally, the structure of the lexical knowledge base needs to be finished before it is moved to some external data source. Because the lexical knowledge base was being developed all the way though the project, there was no obvious time to implement this change.

It would be possible to change the system to use an external data source, but it would require significant changes in the code, and it would mean that further development on the lexical knowledge base would be more difficult. However, it would also allow for the possibility of changing the data source without altering the code, and to have different dictionaries for different purposes. Despite the fact that it does not use an external data source, the lexical knowledge base still performs the tasks that are required in a well-structured way.

7.5 Linked Clauses

Linked clauses are clauses that are joined to form one sentence using a connecting word like “and” or “but”. To be able to generate clauses that are joined in this way, the clauses must be semantically linked. In the original plan linked clauses were to be generated as part of the grammar. However, when investigating the possibility of creating linked clauses it became apparent that ensuring the necessary semantic link was very difficult. This means that any linked clauses that are created in the final output are separately generated, and then put together afterwards.

The fact that linked clauses are not part of the grammar means that it is considerably less powerful. It means that any linked clauses are effectively in a hard coded format, and that most of the sentences are single clauses. Possible Further work in this area is described in section 8.1.

7.6 Scenes and Interactions

A scene is a series of sentences that all take place in the same location, and an interaction is a series of sentences that happen between two characters. Neither of these ideas were in the original plan, they were both developed when it became apparent where the system was lacking. They are good innovations that help to improve the "flow" of the sentences that are produced, and give them some chronology.

Once sentences could be produced with semantic links, such as common characters and locations, the output was just a collection of randomly generated actions involving the characters that existed. This lacked "flow" and did not make very easy reading. The first attempt to organise the sentences was to create a template describing the order in which clauses could occur. This was very fixed and made the story predictable and repetitive. A scene is an alternative approach to introducing flow, based on assigning probabilities to different types of sentence that change depending on what sentences are produced. This is a much better approach because of the dynamic selection of sentences, but in an ordered fashion.

Although scenes greatly improved the chronology of the text produced, the actions that characters performed were still quite arbitrary, and there was no real link between any of the actions that were taken. Interactions are an extension to the scenes idea. If two characters are in a scene together, then they can have an interaction, which is a series of actions or interactions that happen between them. The clauses generated in an interaction are also based upon probabilities. Interactions are good because they give a link between the actions that are generated, and help to structure the story further.

7.7 Character Goals

Using scenes and interactions to improve the "flow" of the text being produced was quite successful, the text became much more readable, and it was possible to follow what was happening. However, all of the actions being performed were still arbitrary and pointless, there was no direction to the story. Character goals give each character in the story something that they want to achieve, which gives their actions some purpose.

The character goals that are implemented are quite simplistic, so their affect on the text produced is not large. However it does make the story more interesting, and also affects the characters emotions, which helps to determine their other actions. These simple goals demonstrate the concept of goals and indicate that if they were enhanced the story could be greatly improved.

7.8 Final Comments

Ultimately the project went well, all of the different areas of the system were successfully implemented, and the final program produces some interesting output. Whenever there was a problem or something went wrong, a solution was found that was satisfactory. A contributing factor to the systems success is that each element is kept as simple as possible, all the problems are broken down into small steps, and each area is encapsulated. Keeping the program as simple as possible allowed the gradual build up of ideas to create a fully functional system.

The final output is good, it demonstrates several good ideas. However the full power of the underlying grammar is not reflected in the text produced. The development that went into the sentence generator enabled more tenses and types of action than those that are present in the final output of the system. This is because the clauses used as part of a scene are limited in order to provide a good semantic link between them. Most of the sentences are in the present tense, and most of them are actions or interactions. To be able to fully reflect the power of the underlying grammar, the range of sentences used in scenes must be increased.

Throughout the project there were two contradictory methods for generating text that had to be considered. Initially all the generation was done entirely by

randomly selecting what to do, which generated completely random pieces of text with no real meaning. It is clear that this method is not useful for generating text with any semantic connections. The opposite approach is to create a template for the text that is to be generated, and to follow that template. This leads to very structured text, which can have deep semantic meaning, however it is very static and each story is almost the same. Neither approach is satisfactory, it is necessary to find a method that has elements of random generation but done in a structured way, so that semantics can play a role. This is one of the key issues involved with generating text, the balance between random and structure. This project does not rely entirely on either approach, and tries to find the middle ground between the two.

Chapter 8

Further Work

Although this project is a good demonstration of text generation on its own, it is very extensible; there are many possibilities for enhancing the system in various different directions. This section attempts to outline some of the ideas for further work on this project.

8.1 Improve Linked Clauses

Linked clauses have not been properly implemented because of the difficulties in creating semantic connections between the different clauses. To be able to join two clauses together into one sentence, the semantic link has to be very strong. This means that it is difficult to create a general case algorithm for generating linked clauses, but a series of specific cases is not expressive enough. However, if linked clauses are properly implemented then the sentences would be much more expressive. This could be done by increasing the power of the grammar to include linked clauses. However, changing the grammar would mean changing a lot of the rest of the system and would require the semantic link to be established at quite a low level. This is because the grammar is the lowest layer in the system and the semantic phase uses it to generate clauses.

Alternatively linked clauses could be created by adding a new layer in between the grammar and the scene. This new layer would handle the semantics between the clauses, so the current grammar would stay intact. Considerable testing and development of this layer would be necessary to get consistently good results, and the higher semantic layer would also have to be altered to reflect the changes

made.

Linked Clauses could also be improved by analysing the clauses that are currently produced, and inserting the joining words where possible. This would involve looking at all the sentences as the scene produces them, and comparing adjacent pairs, to see where they could be joined to form one sentence. This could be done as a simple pattern matching exercise, if the program knew which types of sentence were compatible and which joining words appropriate. This is the most straightforward solution however, catching clauses after they are generated is not very elegant and ideally linked clauses should be constructed at an earlier phase in the system.

8.2 Goals and Sub-Goals

The concept of character goals was introduced to try and give the story some direction and give the characters a purpose. The current goals are only very simplistic and could be enhanced in several different ways. If character goals are developed greatly, then they could form the basis of a much more complicated story, where different characters interact with each other trying to achieve their goals.

Currently the only goal in the system is to try and acquire an object, and this goal is set at the beginning when the characters are created. This goal could easily be changed to include doing anything that is in the dictionary, like wanting to eat something, or to buy something. This would simply mean storing a verb as well as a noun to represent the goal. Goals could be further enhanced by involving specific locations and other characters.

Goals could also be broken down into a series of sub-goals which all have to be achieved. The characters could have multiple goals at the same time, so their actions could be fulfilling aspects of multiple goals simultaneously. Some sub goals could have to be done in a specific order, and others in any order. This system of goals and sub-goals could be built into a complicated structure resulting in every action being driven by one of the characters goals. This would give the story a strong direction, and eliminate any arbitrary actions.

Another possible enhancement is to implement dynamic goals. The goals of a character could change over time depending on what happens in the story. This means that the goals of a character could build up over time, adding or removing sub goals, or changing depending on what the character does. Combining these ideas would create a powerful goals system that could drive the story without the need for predetermined plots or structured interactions.

8.3 Enhanced Interactions

One of the key areas of the system are interactions, which is where most of the actions between characters are generated, and where goals are used. The interactions are where the real story happens; the rest of the text is just setting the scene. Therefore, improving the interactions will improve the important parts of the text. Currently the interactions are a series of actions, interactions and passive actions that happen involving the two characters that are interacting. The clauses are chosen based upon probabilities that are assigned to each type of event happening and these probabilities change as the interaction progresses.

Because the actions and interactions are chosen at random, there is only a weak semantic link between the consecutive sentences, the characters that are common. The interactions would "flow" better if the link between the consecutive clauses were stronger. To achieve this, the way that interactions take place has to change to take into account more factors when choosing the next action. It will have to know what type of clause is likely to follow what has just happened, and what type of action is likely, depending on what has happened so far. To be able to produce better passive actions the interaction would have to know what the location is, and be able to know what characters are likely to be doing in that location. It is difficult to generate strong semantic connections and keep enough randomness to give variation, but to get better interactions more logic has to be put in place to generate more appropriate clauses.

Another way of improving interactions is to allow different characters to join in, which increases the complexity of the interaction considerably. If more than two characters are interacting at the same time, then there are many different possibilities of who interacts with whom. This would be difficult to do, but it would greatly increase the power of interactions, and goals could be designed to

include multiple characters at the same time. To do this the interactions would have to take into account the relationships between all of the characters, and each of their goals. Implementing this change would break the constraints of only ever having two characters interacting at the same time, and allow the action to be more free.

8.4 Description Engine

At the moment the system is very action oriented, most of the sentences produced involve characters doing something. To help break up the actions, and flesh out the story, descriptive sentences could be introduced, which describe what is happening in more detail. To be able to add descriptions, the system would need to keep track of more information about objects and actions, so that appropriate descriptions could be selected.

There are several different things that the system could describe; the most obvious is the scenery that is around the action. To be able to describe the scenery the program must be able to know about the location and to build up descriptions as they are created, to ensure that contradictory descriptions are not generated. When deciding what to describe, the previous clauses could be taken into account, so that something relevant can be described. It would also be possible to describe any nouns that are in the scene, and build up descriptions of them in a similar manner. Characters already have descriptions attached to them, but they are not very advanced. It may also be possible to link the descriptions to adverbs that could be inserted into some of the action sentences. This would help to describe the action better, and make the text less sterile.

Descriptions could also be used to introduce nouns into a scene, if something gets described, then characters can interact with the object. This would make the location seem more real, and create action that did not always revolve around the characters.

8.5 Other Enhancements

There are many other smaller enhancements that could be made to the system; one possible enhancement is a title generator. A title generator would create a

title for the story based on its content. It would have to find the key events and main characters in the story, and come up with one line to sum it up. This could very simple, and produce very self-explanatory titles, or it could be more complicated and try to create more cryptic or interesting titles.

It may be possible to create a plot generator to help guide the story along a certain path. A plot generator would create elementary story lines that characters would follow, to help give the story more direction. The plots would have to be quite flexible, so that the same story would not get generated twice. This idea is an alternative to character-based story telling, as the plot would already be defined, and the characters would just be following it. However, it could be incorporated into the system along with the character-based interaction, still allowing the characters to interact freely, but guiding them along the underlying storyline.

Another possible enhancement to the program is to improve the dictionary by adding many more words to it. Most of the time spent doing this would be in finding verbs, and creating verb rules so that the verbs could be used properly. This would be quite a time consuming job because the verbs need to be tested fully to make sure that the rules are correct, but the program would become a lot more expressive. The dictionary could also be moved to an external data source, so that it could be modified without having the change the code.

To include more aspects of the grammar in the final output, different parts of the text could be generated in different tenses. Characters could have "flash back" parts of the story, where the text is generated in the past tense, or the goals of the characters could be written in the future tense. Integrating the different tenses into the final output would make the story more interesting, and help to break up some of the present tense actions.

Chapter 9

Conclusions

This project has been quite successful, and achieved what was set out to achieve. It demonstrates that relatively compact grammars can be very expressive if used in the correct way. The grammar that plays the central role in the clause generation is quite small, and does not cover a very large fraction of the English language. However, the system is capable of producing a wide range of different concepts. This shows that it is not necessary to try and define the entire language to have a successful language generator.

The text that the program produces shows that character-based story generation does work. Despite the fact that this system is only a limited example of the possibilities of character-based story generation, it demonstrates that generating text with no plot to follow can produce interesting results. Although there is no fixed progression of activities, the characters tend to interact with each other in a way that leads to some form of conclusion, which is different for each story that is produced.

Although only a small amount of time was spent working on it, this project shows the potential of character goals in this style of story generation. Even by adding simplistic goals the story gains much more direction, and the actions have more purpose.

Despite the fact that this project is only a limited example, it shows that it is possible to create a story generator based upon formal grammars. This means that with more work in the area, it should be possible to create much more complicated systems to generate much more realistic scenarios. The primary ap-

plication for these ideas might be the computer games industry, where games could be made such that the story is different every time the game is played.

One of the issues that became apparent through the course of this project is how to manage the amount of random generation to allow in the system. It is necessary to have a certain amount of structure in place, or the text is entirely random, and the semantics falls apart, but too much structure leads to very rigid text with little variation to keep it interesting. Maintaining a good balance between random and structure is the key to generating larger amounts of text. The scenes and interactions in this project attempt to find a balance between the two, and the output demonstrates the level of success that is achieved.

References

Dr Bas Aarts. The internet grammar of english, 1998. URL `http://www.ucl.ac.uk/internet-grammar/frames/contents.htm`.

Emmon Bach. *Syntactic Theory*. Holt, Rinehart and Winston, 1974.

M Cavazza. Dialogue generation in character-based interactive storytelling. In *AAAI First Annual Artificial Intelligence and Interactive Digital Entertainment Conference, Marina del Rey, California, USA*, 2005.

Noam Chomsky. *Syntactic Structures*. Moulton & Co., 1957.

V.J. Cook. *Chomskys Universal Grammar 2nd Edition*. Blackwell Publishers, 1996.

Ann Copestake. An open source grammar development environment and broad-coverageenglish grammar using hpsg. In *LREC 2000 2nd International Conference on Language Resources & Evaluation*, 2000.

Antonio M. Corb-Bellot. An open-source shallow-transfer machine translation engine. In *Proceedings of the European Association for Machine Translation, 10th Annual Conference, Budapest 2005*. Transducens group, Departament de Llenguatges i Sistemes Informtics, 2005.

Roger Fowler. *An Introduction to Transformational Syntax*. Routledge, 1971.

M Gross. *Introduction to Formal Grammars*. Springer-Verlag New York Inc, 1970.

Zellig Harris. *A Grammar of English on Mathematical Principals*. John Wiley & Sons Inc, 1982.

Richard Hudson. Word grammar, 2005. URL `http://www.phon.ucl.ac.uk/home/dick/wg.htm`.

Aravind K. Joshi. Tree-adjoining grammars. *Handbook of formal languages, vol. 3: beyond words*, 1997.

Charles I Kelly. Fun with randomly-generated sentences, 1998. URL `http://www.manythings.org/rs/`.

Howard Lasnik. *Syntactic Structures Revisited*. The MIT Press, 2000.

Michael Mateas. Faade: An experiment in building a fully-realized interactive drama. In *Game Developers Conference, Game Design track, March 2003*. Literature, Communication and Culture and College of Computing, Georgia Tech, 2005.

Daniel Naber. *A Rule-Based Style and Grammar Checker*. PhD thesis, Bielefeld University, 2003.

Michael Newby. *The Structure of English: A Handbook of English Grammar*. Cambridge University Press, 1987.

Andrew Radford. *Minimalist Syntax: Exploring the Structure of English*. Cambridge University Press, 2004.

Ivan Sag. Head-driven phrase structure grammar, 2001. URL `http://hpsg.stanford.edu/ideas.html`.

Daniel Temperley. Link grammar, 2005. URL `http://www.link.cs.cmu.edu/link/`.

Owen Thomas. *Transformational Grammar and the Teacher of English*. Holt, Rinehart and Winston, 1974.

Elizabeth Traugott. *The History of English Syntax*. Holt, Rinehart and Winston, 1972.

Alan Turing. Computing machinery and intelligence. *Mind 49*, 1950.

R.W. Zandvoort. *A Handbook Of English Grammar 7th Edition*. Longman, 1975.

Julie Zelenski. The random sentence generator, 1999. URL `http://www-cs-faculty.stanford.edu/ zelenski/rsg/`.

Appendix A

Appendix

A.1 Sample Grammar

Sample grammar from a phrase grammar based generator, can be seen demonstrated at The Random Sentence Generator Zelenski [1999].
{ (start)
The (object) (verb) tonight. ;
}

{ (object)
waves ;
big yellow flowers ;
slugs ;
}

{ (verb)
sigh (adverb) ;
portend like (object) ;
die (adverb) ;
}

{ (adverb)
warily ;
grumpily ;
}

A.2 Initial Phrase Structure Grammar

```
addRule("S", new String[]{"NP","VP","NP"});

addRule("NP", new String[]{"D","NN"});
addRule("NP", new String[]{"D","AD","NN"});
addRule("NP", new String[]{"PN"});

addRule("NN", new String[]{"Monkey"});
addRule("NN", new String[]{"Cat"});
addRule("NN", new String[]{"Fish"});

addRule("D", new String[]{"The"});
addRule("D", new String[]{"A"});

addRule("AD", new String[]{"Red"});
addRule("AD", new String[]{"Black"});
addRule("AD", new String[]{"Dead"});
addRule("AD", new String[]{"Killer"});

addRule("PN", new String[]{"Fred"});
addRule("PN", new String[]{"Burt"});

addRule("VP", new String[]{"AV","V"});
addRule("VP", new String[]{"V"});

addRule("AV", new String[]{"quickly"});
addRule("AV", new String[]{"slowly"});
addRule("AV", new String[]{"nearly"});

addRule("V", new String[]{"ate"});
addRule("V", new String[]{"killed"});
addRule("V", new String[]{"stole"});
addRule("V", new String[]{"hid"});
```

A.3 Phrase Structure Grammar After Lexical Insertion

```
addRule(verb.VERBCLAUSE, new String[]{"NP","TVP","NP"});
addRule(verb.VERBCLAUSE, new String[]{"NP","NPVP"});
addRule(verb.VERBCLAUSE, new String[]{"NLVP","NP"});

addRule("NP", new String[]{"D",noun.COMMONNOUN });
addRule("NP", new String[]{"D","AD",noun.COMMONNOUN});
addRule("NP", new String[]{noun.PROPERNOUN });

addRule("D", new String[]{"the"});
addRule("D", new String[]{"a"});

addRule("AD", new String[]{"red"});
addRule("AD", new String[]{"black"});
addRule("AD", new String[]{"dead"});
addRule("AD", new String[]{"killer"});

addRule("TVP", new String[]{"AV",verb.TRANSITIVEVERB });
addRule("TVP", new String[]{verb.TRANSITIVEVERB});

addRule("NPVP", new String[]{"AV",verb.NOUNFIRSTVERB });
addRule("NPVP", new String[]{verb.NOUNFIRSTVERB});

addRule("NLVP", new String[]{verb.NOUNLASTVERB});

addRule("AV", new String[]{"quickly"});
addRule("AV", new String[]{"slowly"});
addRule("AV", new String[]{"graciously"});
addRule("AV", new String[]{"nearly"});
```

A.4 Code for Structural Analysis Matching

```
CheckRule ( treeNode t ) {
        if (match found on symbol 0 ) {
            first node in match list = t;
            if ( checkNextNode(t,1) )
              perform structural change
         }


       for (all of the children of this node)
            recursively call CheckRule ( childNode )
    }


boolean checkNextNode ( treeNode t, int SAIndex ) {
        treeNode commonParent = t.parent;
        treeNode childNode = t;


        while (There is not a right branch) {
            if (no more parents) return false
            go up one parent
         }


        childNode = first child to the right of the commonParent


        //now check for the second symbol
        while (childNode does not match the symbol) {
            if (run out of children) return false
            childNode = first child of childNode
         }


        add childNode to match list
        if (run out of symbols to check) return true


        return checkNextNode(childNode, SAIndex + 1)
    }
```

A.5 Initial Scene-Based Output

Ernie is a mouse. Ernie has red eyes.
Burt is a fish. Burt has a yellow tail.
Burt doesn't like Ernie.
it was wednesday and it was windy.
Burt is in the Bar. Burt is hiding the pianos.
Ernie enters the Bar and Burt and Ernie play with the lions and Then Burt and Ernie hide from the cats.
Ernie leaves the Bar.

it was monday and it snowed.
Burt is in the Cafe. Burt is stealing curries.
Ernie goes to the Cafe and Burt and Ernie steal the apples and Then Burt and Ernie play with a brick.
Ernie leaves the Cafe.

The End

A.6 Secondary Scene-Based Output

it was tuesday. it was windy and Ernie is in the Cafe, he is sitting by a table. he is a man and he is clever. he is clever. he steals a table. Burt enters the Cafe. he goes up to Ernie. Burt and Ernie drink. Ernie laughs with Burt. he argues with Burt. Burt and Ernie steal curry. Burt leaves the Cafe. Ernie hates him. he was angry.
he stands by a table.

it rained and Ernie goes to the Bar. the bar tender drinks. he is a boy and he is cunning. Jane is in the Bar, she is sitting on a chair. she eats. she is a woman and she is insane. she is devious. Burt is in the Bar, he is sitting by the door. he was upset. Ernie sits by the door. Jane goes over to him. Ernie argues with her. Jane and Ernie walk by the bar. they stand by the door. Ernie plays with

Jane. Jane and Ernie sit on some chairs. Ernie argues with Jane. Jane and Ernie stand by the door. they walk by the door. Jane shouts at Ernie. Jane and Ernie sit by the bar. Ernie laughs with Jane. Jane and Ernie stand by the bar. they drink some coke. Ernie leaves the Bar. Jane hates him. she was angry.
Burt walks by the bar. Jane walks by the bar.

it was tuesday. Ernie is in the Park, he is sitting by the river. the park keeper drinks some beer. he is a big boy and he is rather stupid. Burt is in the Park, he is sitting by a bush. he is a big boy and he has orange hair. he is rather blind. the park keeper stands by some bushes. he goes over to Ernie. Ernie talks to him. the park keeper laughs at him. the park keeper and Ernie drink. they eat. the park keeper laughs with Ernie. he leaves the Park. Ernie is friends with him. Burt goes over to him. he doesn't like Ernie. he sits next to Ernie. Burt and Ernie stand by a bush. Burt shouts at Ernie. he argues with Ernie. Ernie shouts at him. Burt and Ernie walk by the river. Ernie argues with Burt. Burt was angry. Burt and Ernie eat some apples. Burt leaves the Park. Ernie stands by a path. he walks by a path.
he stands by a path. he walks by some trees. he stands by some trees.

The End

A.7 Test Results

A.7.1 Test Output 1

Emma is in the park; she is standing by some bushes. It was slightly windy and it was Saturday. Holly enters the park. She is a yellow girl and she is very stupid. The park keeper laughs. He is a man and he has a yellow nose.
Emma goes up to Holly. She wants a pear and Holly has a pear. Emma asks her about the pear but Holly doesn't give her the pear. Emma was angry. She punches Holly. Holly shouts at her. Holly and Emma stand by the river. They walk by the river. They stand by the river. They eat curry. Emma steals a pear from Holly and then she leaves the park.
Holly leaves the park.

Emma goes to the shops. It hailed and it was Tuesday. The shopkeeper buys the counter. Holly goes to the shops.
She goes over to Emma. Holly and Emma sit by the door. Holly was angry. Holly and Emma stand by the counter. Holly punches Emma. She steals the door from Emma. She slaps Emma. Emma punches her. Holly steals from her. Holly and Emma sit by the door. Emma argues with Holly. Holly and Emma buy some keyboards. Holly leaves the shops. Emma hates her.
She leaves the shops.

Emma is in the bar; she is standing by the door. It was Tuesday. Holly goes to the bar. The bar tender eats.
He stands by the door. He walks by the bar. Holly goes up to Emma. She hates Emma. Emma steals the bar from her. Holly and Emma walk by the bar. They read some newspapers. Emma leaves the bar. The bar tender stands by the bar. Holly leaves the bar.

It snowed and it was Wednesday. Holly goes to the shops. Emma enters the shops.
She sits by the counter. She goes over to Holly. Holly wants some bananas and Emma has a banana. Holly asks her about the banana but Emma doesn't give her the banana. Emma and Holly buy the keyboards. Emma argues with Holly. Emma and Holly eat an orange. Emma punches Holly. Emma and Holly stand by the door. Holly leaves the shops. Emma hates her. The shopkeeper sits on a chair.
He stands by the counter. Emma leaves the shops.

Emma hates Holly and Holly hates her. Emma kills her and then she goes to the pub.

The End

A.7.2 Test Output 2

It snowed and it was Friday. Adam is in the park; he is sitting on a bench. Emily is in the park; she is standing by a tree. She is a girl and she has purple eyes. She has a green nose. She laughs.

The park keeper walks by the river. Emily walks by the river. Adam goes over to her. Emily was glad. Emily and Adam stand by a tree. Adam plays tennis with Emily. Emily and Adam drink. They sit by a path. They walk by some trees. They stand by a tree. Emily leaves the park. Adam loves her. He was thrilled.

It was Tuesday. Millie is in the bar; she is sitting by the bar. It hailed and Millie orders some lemonade. Mohammed is in the bar; he is sitting on a chair.
He goes up to Millie. Millie steals from him. Mohammed wants a bicycle and Millie has some bicycles. Mohammed shouts at her. He asks Millie about the bicycle but Millie doesn't give him the bicycle. Mohammed and Millie stand by the door. Mohammed steals a bicycle from Millie and then he leaves the bar.
Millie leaves the bar.

Mohammed enters the cafe. It hailed and it was Wednesday. Emily is in the cafe; she is walking by the door. Millie goes to the cafe. Adam is in the cafe; he is sitting by the door. The waiter orders a crisp.
Adam goes up to Millie. Adam and Millie order the crisp. Millie wants some sandwiches and Adam has some sandwiches. Millie asks him about the sandwich and Adam gives her the sandwich. Millie was thrilled. Adam and Millie eat. They walk by a table. They sit on some chairs. Millie leaves the cafe. Adam loves her. He goes up to Mohammed. Mohammed shouts at him. Adam and Mohammed stand by a table. Mohammed steals the door from Adam. He kills Adam. He goes up to Emily. Emily was thrilled. She talks with Mohammed. Mohammed was sad. Emily plays tennis with him. Mohammed argues with her. Emily was glad. Mohammed and Emily walk by the door. They stand by the door. They order the crisp. Mohammed leaves the cafe. Emily likes him.
The waiter sits by the door.

It was slightly cloudy and it was Friday. Emily is in the cafe; she is walking by the door. The waiter laughs. Millie goes to the cafe.
Emily stands by the door. She goes over to Millie. Millie talks with her. Emily laughs with her. Emily and Millie sit by a table. Emily was thrilled. Emily and Millie walk by a table. Emily wants a book and Millie has some books. Emily talks with her. She asks Millie for the book and Millie gives the book to her. She laughs at Emily. Emily and Millie read. They order some crisps. Millie leaves the cafe. Emily is friends with her. The waiter walks by the door. He was glad.

He stands by the door. Emily sits by the door. She leaves the cafe.

Emily is friends with Millie. They go to the cafe together and Emily plays draughts with Millie.
Mohammed killed Adam. The Police are searching for him. Mohammed runs away to America.

The End

A.7.3 Test Output 3

It was Saturday. It snowed and Samuel enters the shops. He is a big boy and he has a tiny nose. He buys the counter. He has tiny hair. Lucy enters the shops. She is a girl and she is cunning. The shopkeeper buys some tabla.
Lucy goes over to Samuel. Lucy and Samuel buy some cameras. They stand by the counter. They drink some wine. Samuel leaves the shops.
Lucy sits on a chair.

It was cloudy and it was Wednesday. Lucy enters the cafe. The waiter reads some newspapers. Samuel is in the cafe; he is sitting by the door.
Lucy goes over to him. Lucy and Samuel order some oranges. Lucy steals the oranges from Samuel. She was sad. Samuel wants some pears and Lucy has a pear. Samuel was sad. Lucy and Samuel eat. They stand by a table. They sit on some chairs. Lucy steals the oranges from Samuel. Samuel asks her for the pear but Lucy doesn't give him the pear. Samuel steals the pear from her and then he leaves the cafe.
Lucy leaves the cafe.

It was Wednesday. Samuel enters the bar. It rained and Lucy is in the bar; she is standing by the bar. The bar tender orders some coke.
Lucy goes up to Samuel. She hates Samuel. Lucy and Samuel order the coke. Lucy was quite angry. Lucy and Samuel walk by the door. Lucy wants some bicycles and Samuel has a bicycle. Lucy asks him about the bicycle but Samuel doesn't give the bicycle to her. Lucy and Samuel drink the coke. They sit on some chairs. Samuel was very angry. Lucy leaves the bar. The bar tender stands by the door.

He sits on a stool. Samuel leaves the bar.

Samuel hates Lucy and Lucy hates him. Samuel steals from her and then he goes to the cafe.

The End

A.7.4 Test Output 4

It was Friday. Joseph is in the bar; he is standing by the door. It was rather cloudy and Joseph is a dead boy and he is rather cunning. He is quite devious. Harry enters the bar. Joseph laughs.
Harry goes up to him. Joseph wants a chocolate and Harry has some chocolates. Joseph argues with him. He asks Harry about the chocolate but Harry doesn't give the chocolate to him. Harry and Joseph stand by the bar. Joseph was quite angry. Harry steals from him. He kills Joseph. The Police enter the bar they are looking for Harry. Harry swiftly leaves the bar, avoiding the Police.
The bar tender sits on a chair.

It was Friday. Ryan is in the park; he is standing by a path. It was rather cloudy and Ryan is a dead boy and he is rather cunning. He is quite devious. Megan enters the park. Ryan laughs.
Megan goes up to him. Ryan wants some chocolates and Megan has a chocolate. Ryan shouts at her. He asks Megan about the chocolate but Megan doesn't give the chocolate to him. Megan and Ryan sit on some benches. They stand by some trees. They walk by a path. Megan was quite upset. Ryan steals some chocolates from her and then he leaves the park.
Megan leaves the park.

It was Sunday. It was slightly cloudy and Megan is in the cafe; she is sitting by a table. The waiter reads. Connor is in the cafe; he is standing by the door.
He goes over to Megan. He sits with Megan. Megan and Connor stand by the door. They read a letter. Megan steals the letter from Connor. Connor leaves the cafe. Megan hates him. She walks by a table. She was quite angry. She sits by the door.
The waiter walks by a table. Megan leaves the cafe.

It was Sunday. It was slightly cloudy and Megan is in the cafe; she is standing by a table. The waiter orders an apple. Connor is in the cafe; he is walking by the door.
Megan goes over to him. She hates Connor. She walks with Connor. Connor punches her. Megan steals the apple from him. Connor shouts at her. He punches Megan. Megan argues with him. Connor wants some pasta and Megan has some pasta. She steals some chairs from Connor. Connor asks her about the pasta but Megan doesn't give the pasta to him. Connor kills her. He was very angry.
He sits on a chair. He leaves the cafe.

Harry goes to the cafe. Ryan is in the cafe; he is sitting on a chair. It was Friday. The Police enter the cafe they are looking for Harry. Ryan goes over to Harry. Harry wants a bag and Ryan has a bag. Ryan and Harry read some magazines. Harry asks Ryan for the bag but Ryan doesn't give him the bag. He leaves the cafe. Harry hates him. The Police arrest him. The waiter walks by the door.

Connor goes to the shops. Ryan is in the shops; he is walking by a shelf. Isabelle goes to the shops.
She goes over to Connor. Isabelle and Connor eat a crisp. Isabelle wants some pasta and Connor has some pasta. Isabelle and Connor drink. They buy the counter. Isabelle kills Connor. Ryan leaves the shops. Isabelle walks by some shelves. She was very angry. The shopkeeper stands by the door. He walks by some shelves. Isabelle leaves the shops.

Harry killed Joseph. The Police arrested him and Harry is going to prison.
Connor killed Megan but then Isabelle killed him.
She killed Connor. The Police are searching for her. Isabelle hides in America.
Ryan was happy because he got some chocolates. He goes to the cafe to celebrate.

The End

A.7.5 Test Output 5

Luke is in the bar; he is sitting by the bar. Charlotte is in the bar; she is walking by the bar.
She goes over to Luke. Luke talks about a chair with her. He was happy. He wants some bells and Charlotte has some bells. Luke asks her for the bell but Charlotte doesn't give him the bell. Luke and Charlotte order some coke. They stand by the bar. Luke leaves the bar. Charlotte likes him. She was glad.
She sits by the door. She leaves the bar.

It snowed and it was Thursday. Connor goes to the park. Lucy goes to the park. She goes up to Connor. Connor and Lucy stand by a bush. Connor wants some string and Lucy has some string. Connor and Lucy eat. They drink. They sit on some benches. They stand by a path. Lucy talks about a bush with Connor. Connor and Lucy walk by some bushes. Connor asks Lucy about the string but Lucy doesn't give the string to him. Connor and Lucy sit by some bushes. Connor steals some string from Lucy and then he leaves the park. The park keeper stands by the river.
He walks by a bush. Lucy leaves the park.

It snowed and Lucy goes to the shops. It was Friday. Luke goes to the shops. The shopkeeper buys a crisp. He walks by the counter. Luke sits by the counter. He was quite angry. Lucy goes up to him. Luke wants a bell but Lucy doesn't have a bell. Lucy and Luke stand by the counter. Lucy laughs with Luke. She talks about some oranges with Luke. Lucy and Luke read. They drink some squash. Luke talks about some bats with Lucy. Lucy leaves the shops. Luke loves her.
He was very thrilled. The shopkeeper stands by a shelf.

It was rather rainy and it was Wednesday. Luke goes to the bar. Connor goes to the bar. The bar tender reads.
Connor stands by the bar. Luke goes over to him. Luke and Connor drink. Connor was rather glad. He leaves the bar. Luke is friends with him. He walks by the door. He stands by the door.
He sits on a chair. He leaves the bar.

It was Tuesday. Charlotte is in the bar; she is walking by the door. It rained and

the bar tender reads a letter. Luke goes to the bar.
Charlotte goes over to him. She likes Luke. Luke steals from her. Charlotte wants some lemonade and Luke has some lemonade. Charlotte asks him for the lemonade but Luke doesn't give the lemonade to her. Charlotte was angry. Luke plays lines of action with her. Charlotte and Luke stand by the door. Luke talks to Charlotte. Charlotte steals some lemonade from him and then she leaves the bar.
The bar tender stands by the door. Luke leaves the bar.

It was Tuesday. Charlotte is in the shops; she is walking by the door. It rained and the shopkeeper reads a letter. Luke goes to the shops.
Charlotte goes over to him. She hates Luke. Luke steals from her. He wants some bells and Charlotte has a bell. Luke asks her for the bell but Charlotte doesn't give the bell to him. Luke shouts at her. He steals a bell from Charlotte and then he leaves the shops. Charlotte was upset.
The shopkeeper walks by the counter. Charlotte leaves the shops.

Luke loves Lucy and Lucy loves him. They go on a trip together.
Charlotte was glad because she got some lemonade. She goes to the bar to celebrate.
Connor was glad because he got some string. He goes to the zoo to celebrate.

The End