Chapter 10

The WAG Analyser

1. The Stages of Sentence Analysis

The WAG system includes a sentence analyser, which reads in paragraphs of text, and produces a micro-semantic analysis of each sentence. The WAG parser was initially developed as part of the Electronic Discourse Analyser (EDA) project, funded by Fujitsu (Japan). Analysis in WAG can be broken up into four stages:

1. **Graphological Analysis**: the text is broken up into sentences, graphological-words, punctuation, etc.;
2. **Lexical Analysis**: Using the lexicon, each graphological word and punctuation mark is assigned a set of candidate lexical items, and associated inflectional class;
3. **Grammatical Analysis**: a lexico-grammatical representation (sometimes several) is produced from the string of lexical-items;
4. **Micro-Semantic Analysis**: a micro-semantic representation is derived from each lexico-grammatical representation.

Figure 10.1 shows a) how these stages relate to each other; b) the resources each process requires; and c) the flow of information between stages (inputs and outputs). The rest of this chapter will describe the stages in more detail. The resources involved in the processing (graphological, lexico-grammatical and micro-semantic resources, the lexicon, and semantico-grammatical mapping constraints) are described in Part A of this thesis.

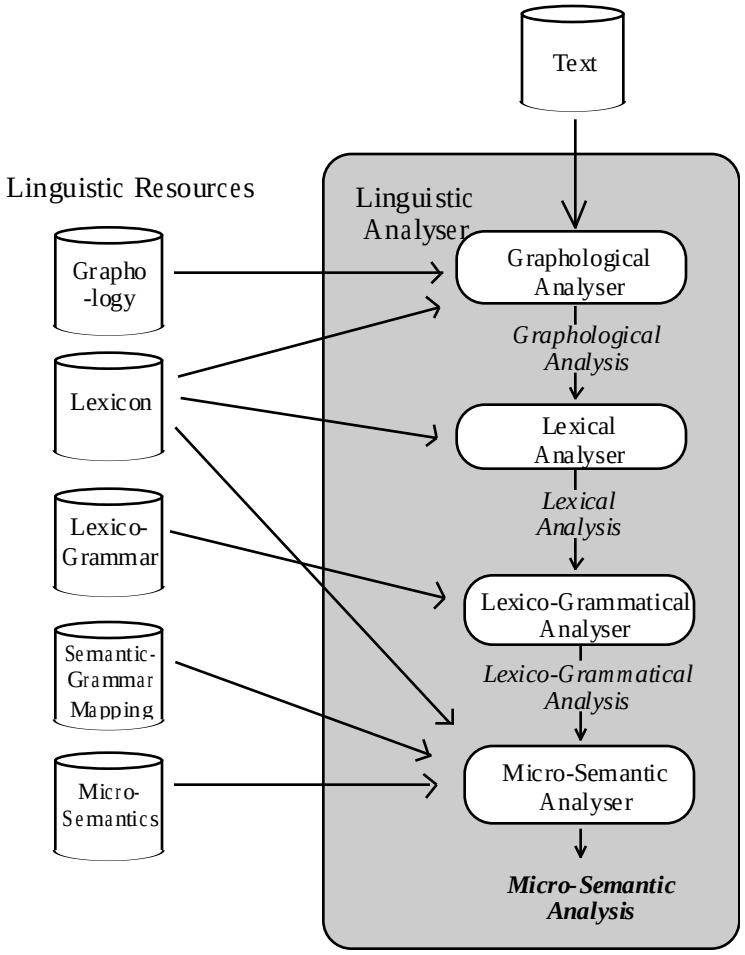

Figure 10.1: An Overview of the Analysis Process

2. Graphological Analysis

The first stage of sentence analysis is graphological analysis. This process involves:

- Segmentation of the text into sentences, which are in turn segmented into graphological-words and punctuations;
- De-capitalisation of sentence-initial forms, unless entered in the lexicon as a proper noun;
- Recognition of multi-word terms, e.g., *New Zealand* would be returned as a single spelling.

Graphological and lexical analysis are to some degree inter-leaved, since the graphological interpretation of a string of characters sometimes depends upon the lexical status of the string. For instance, in the de-capitalisation step above, the graphological analyser needs to know if the word is available as a proper noun or not. The recognition of multi-word items also requires some inter-leaving.

The graphological analysis of the following sentence...

With a TSS session, this character string is entered in accordance with the prompts provided when the LOGON command is entered.

...would result in the following list:

```
("with" "a" "TSS session" "," "this" "character string" "is" "entered"
"in accordance with" "the" "prompts" "provided" "when" "the" "LOGON
command" "is" "entered" ".")
```

In some cases, sequences of words are recognised as a single compound word, e.g., "TSS session". The lexicon stores some technical phrases as if they were single words. This will be discussed below.

3. Lexical Analysis

The output of graphological processing is made available to the lexical-analyser. The lexical analyser uses the lexicon to produce alternative lexical interpretations of each graphological-word. These lexical analyses are then made available to the parser.

3.1 Morphological Analysis

To discover the lexical item corresponding to a particular spelling, two approaches are possible:

Morphological Analysis: The morphology rules are applied to the spelling, to recover the root form, and the inflection class.

Generate and Test: For each item in the lexicon, the spelling of each inflectional variant is generated. Each inflection whose spelling matches the target spelling is returned as a lexical candidate.

Key	Value	
"can"	can-noun	singular-noun
	can-aux	nil
	can-verb	(:and present non-third-person-singular)
"canned"	can-verb	past

Table 10.1: A Sample of the Spelling Table

For large lexicons, morphological analysis is the most efficient solution. Unfortunately, morphological analysis is a complex task, which I have not yet implemented. The generate-and-test option, by itself, is not very attractive. However, it becomes workable if we perform the generation as a pre-compilation task: as the lexicon is loaded in, the possible spellings of each lexical item are generated, and stored away in the *spelling table*. Table 10.1 shows a small section of the spelling table.

Finding the lexical-items for a particular graphological-word is thus a simple table lookup. The *key* to the table entry is the word's spelling. The *value* component is a list of entries, the first item being the lexeme identifier, the second being the inflection for which this spelling is appropriate. The generation of the spelling table amounts to a re-indexing of the lexicon into a form more amenable to parsing.

If a particular graphological word is not found in the spelling table, then a program is called to acquire the lexical-item information: the user is prompted to provide the semantic and grammatical features of the item, as well as the spellings of inflectional forms.

3.2 Compound Words in Parsing

WAG allows multi-word (compound) items to be entered in the lexicon as a single lexical item. These compound items are then treated the same as non-compound words.

Compound items may be of several kinds:

- **Compound Proper Nouns**, e.g., "New Zealand".
- **Compound Technical Terms**: A string of words which recurrently occurs in a register may be treated as a single lexical item, e.g., "Direct Access Storage Device". These are usually treated as a common noun, and can thus be modified

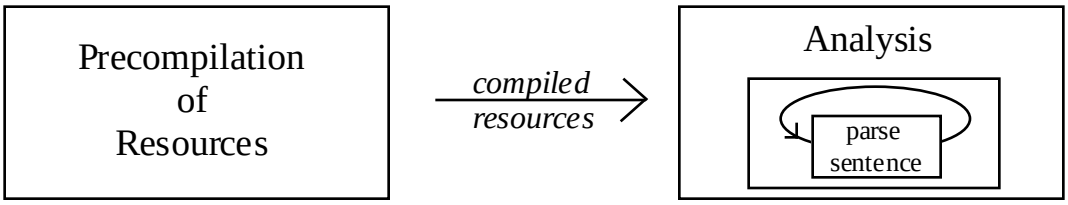

Figure 10.2: Precompilation before Processing

in the same manner as other common nouns, e.g., “a small Direct Access Storage Device from Spain”.

- **Phrasal Verbs**: Compound verbs such as “put up”, e.g., “He put up a sign”. WAG doesn’t yet allow for the separated realisation of the elements of a phrasal verb, e.g., “He put a sign up”.
- **Phrasal Lexicon**: Longer phrases which are to some degree fixed in structure, and lose their meaning when decomposed, can be entered in the lexicon, e.g., “kick the bucket”, meaning, “to die”. This example would be treated as an intransitive verb, and the various inflectional variants acquired, e.g., ing-verb: “kicking the bucket”.

4. Preparation of the Parsing Grammar

The Systemic formalism has a top-down orientation: it mainly presents what types of roles each unit can have, and what types of units can fill these roles. This is ideal for generation, where top-down processing is preferred. However, the orientation of these resources is not well suited to bottom-up processing, which is the most efficient strategy for parsing with large grammars.

Bottom-up parsing requires an upwards orientation of the grammar. It is more concerned with knowing what roles a given unit can fill (the *function potential* of the unit), rather than what constituents it can have.

This section describes the automatic extraction of a parsing-oriented grammar from the usual top-down oriented Systemic grammar. The Systemic resources are *re-compiled* into a parsing-grammar. By using a grammar tailored for bottom-up parsing, the efficiency of the parser is increased dramatically, increasing the size of the grammar that the parser can use.

This compilation is often called *precompilation*, since it represents a compiling of the resources *before* parsing begins (see figure 10.2). Precompilation can be seen as performing some of the parsing work before even looking at the input sentence. Because precompilation shifts work from the parser itself, parsing times are substantially reduced.

4.1 Register Restriction of the Grammar

The WAG parser was intended to parse using the Nigel grammar, a comprehensive grammar of English comprising around 750 systems, and 1500 features. Since parsing with this grammar was found to be very slow, the grammar was reduced in size. The grammar was reduced using *register restriction* -- elimination of parts of the grammar which were unlikely to be utilised in the target texts. For example, in a domain of computer manuals, the interrogative structures are not likely to be used[1]. By eliminating these from the grammar, the overall complexity is reduced, and thus forms which *do*

[1] Some of the restricted forms may actually appear in any one text, although quite rarely. There is a trade-off between speed for the majority of sentences, and ability to parse all sentences in a text.

appear in the grammar can be parsed faster. The method of deriving the register-restricted grammar is as follows:

1. A section of the target text (computer manuals) is parsed by hand[2], thus building up a register-profile of our target texts.
2. An automatic procedure extracts out all the grammatical features which occur in these sentences.
3. This information is used to discover which features *do not* occur in the sample.
4. These features and their realisations are then eliminated from the grammar.

The register-restriction produces a grammar which covered all the sentences in the sample, and a reasonable number of those which are not in the sample, depending on the size of the sample (a larger sample would produce a more accurate register profile). The size of the grammar was reduced by approximately 50% using this method.

Register restriction is only necessary when dealing with Systemic grammars which are too large for the parser -- when using a smaller grammar, no register restriction needs to be applied.

4.2 Partial-Structures

The next step involves transforming the lexico-grammatical resources into the parsing-grammar. Before discussing this precompilation process, I will first introduce the notion of *partial-structures*, the basic component of the parsing-grammar. The left-hand side of figure 10.3 shows a systemic feature and its associated realisation statements. The right-hand side of the figure shows the same information, except re-represented as a fragment of a structure tree, or what I will call a *partial-structure* (the '.' between the Subject and the Finite element indicates that they are unordered with respect to each other; the dotted lines at each end of the partial-structure indicate that other elements can precede or follow these elements).

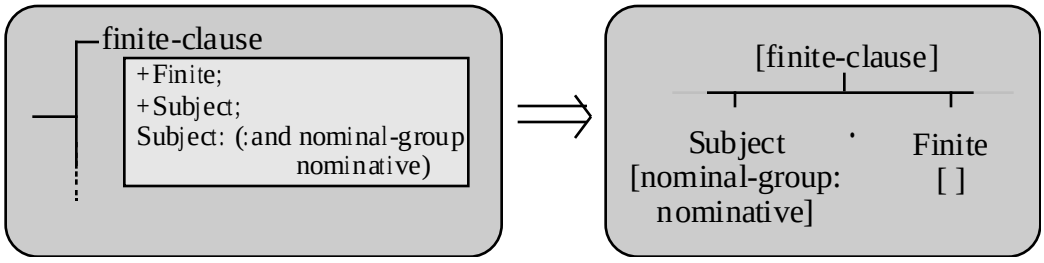

Figure 10.3: Re-Representing a Feature & Realisation as a Partial-Structure

The whole grammar can be re-represented in this manner, representing a shift from a paradigmatically-organised grammar (emphasising features) to a syntagmatically-organised grammar (emphasising structure). However, we will see below that the partial-structures used in the parser are not the same as those shown so far.

This syntagmatic re-representation of the grammar is similar to the approach taken by Kay (1979, 1985). In the Seventies, Kay had been experimenting with parsing with various formalisms, including Systemics. He found that by re-representing grammars in a form closer to the product of analysis (a structural representation), processing was facilitated. Processing becomes simply a matter of unifying these structural fragments. He developed the Functional Unification Grammar (FUG) formalism for this purpose.

---

[2]The hand-parsing is really computer-assisted, -- a tool was developed to traverse the system network for each sentence (and each constituent of the sentence) asking the human which features are appropriate for the target string. This process guaranteed that the human-analysis conformed to the computational grammar. The coding of the text was undertaken by Arlene Harvey and Chris Nesbitt, as part of the EDA project.

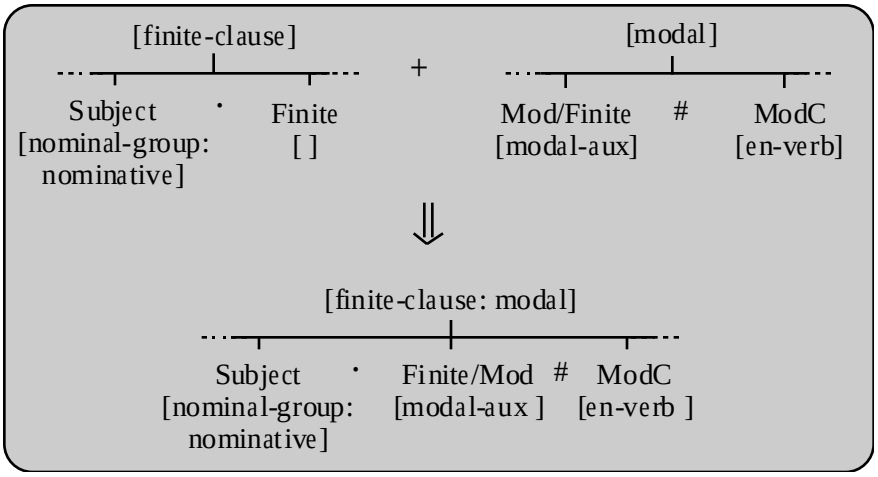

Figure 10.4: Unification of Two Partial-Structures
Produces a New Partial-Structure

Kasper (1988a) utilised the similarities between FUG and Systemic Grammars to parse with the Nigel grammar. He translated the grammar into FUG, and used an existing FUG unifier in his parser. See Kasper (1986) for discussion of the re-representation of Systemic grammar in FUG.[3]

The unification of two partial-structures produces another partial-structure, which contains the sum of the information from both partial-structures, as shown in figure 10.4 (the '#' sign between the Modal role and its complement indicates that the two units are partitioned in relation to each other -- other units may intercede).

Unifications such as these continue until a structure covering the whole sentence is produced. Unification of partial-structures is the basic operation in WAG's parsing process.

4.3 The Partial-Structures of the Parsing Grammar

In this section, I describe the re-expression of the lexico-grammatical resources in a form more useful for bottom-up analysis. Basically, the lexico-grammar and the lexicon are re-represented as a set of partial-structures. The parser uses these partial-structures for analysis -- not the resources as normally used. I will firstly show what types of partial-structures are used, and then show how they are used. For more details regarding the actual production of these partial-structures from the Systemic resources, refer O'Donnell (1993).

4.3.1 'Lexical' Partial-Structures

Three types of partial-structures are used, the first is derived from lexical items. One partial-structure is produced for each inflectional variant. Lexical partial-structures are not pre-compiled, but rather produced during the lexical-analysis phase. Figure 10.5 shows a partial-structure derived from the lexical-item named *they-pron.* A lexical partial-structure contains the features from the *:grammatical-features* field of the lexical item, to which is added the inflectional feature (in this example, the *nominative-pronoun* feature).

[3]Mellish (1988) also compares the Systemic formalism to FUG. However, he limits his discussion to translating systems, not mentioning the translation of realisations.

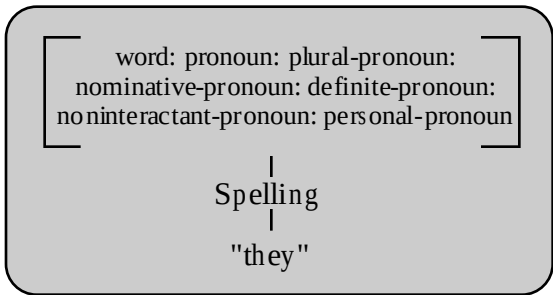

Figure 10.5: A Partial-Structures Derived from a Lexical-Item Definition.

4.3.2 'Linking' Partial-Structures

A second type of partial-structure used in the WAG parser is derived from the preselection rules of the grammar. Figure 10.6 shows one such partial-structure, which was derived by combining the preselections which involve the Thing role of a nominal group, and eliminating those combinations which are incompatible. The partial-structure shown is only one out of several valid combinations. Each of these is called a *linking partial-structure*, because it represents the constraints on the linking between a parent unit and one of its constituents. It is the partial-structures, rather than the preselection rules from which they are composed, that are used by the parser.

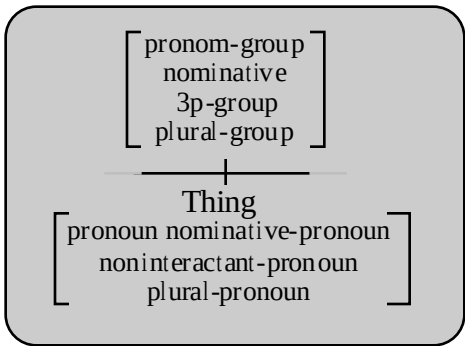

Figure 10.6: A Linking Partial-structure

Sometimes a constituent is linked to its parent through a number of roles, rather than a single role. Figure 10.7 shows another linking partial-structure, this time representing the combination of preselections from two conflated roles. The realisation rule which conflates these two roles is also incorporated into the partial-structure.[4]

[4]The examples in this chapter draw upon the WAG grammar, which provides only Ergative and Mood structure at clause level. The Nigel grammar would provide role-bundles involving up to five roles.

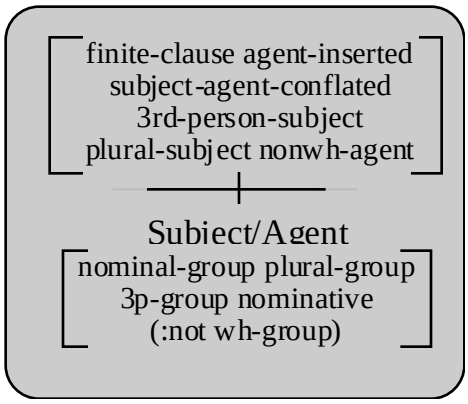

Figure 10.7: A Linking Partial-Structure involving Two Roles.

4.3.3 'Ordering' Partial-Structures

That part of the lexico-grammatical resources which concerns the ordering of constituents is extracted out of the grammar, and re-represented as a set of *ordering partial-structures*. Each of these partial-structures represents the adjacency between two roles, and the condition under which that adjacency is allowed. One of these is produced for each sequencing of two roles allowed by the grammar. Figure 10.8 shows a typical example. It is derived by combining the two realisation statements which allow the order (realisations of features *declarative* and *wh-subject*), and restricting the possibility of the Subject being presumed.

Insertion realisations are sometimes involved, since order and partition realisations may contain optional elements. The insertion statements are used to find the condition under which an optional element is actually present or absent in the structure.

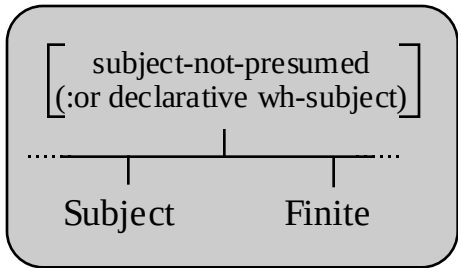

Figure 10.8: An Ordering Partial-Structure

Conflation statements are also sometimes involved, since roles are not always explicitly ordered, but sometimes ordered through a conflated role. For instance, assume we wanted to find what can follow the *Finite* element. One realisation shows that *Finite* can conflate with the *Mod* role, and another shows that *ModC* can be partitioned after *Mod*. Figure 10.9 shows the ordering partial-structure which results from the combination of these realisations.

Since *Mod* and *ModC* are partitioned rather than ordered, we need to restrict the possibility of another element interceding between them: the Subject in a polar-question ("Will she come?"), and a Negator in a negative-clause ("She will not come"). The two disjunctions in the condition restrict these possibilities.

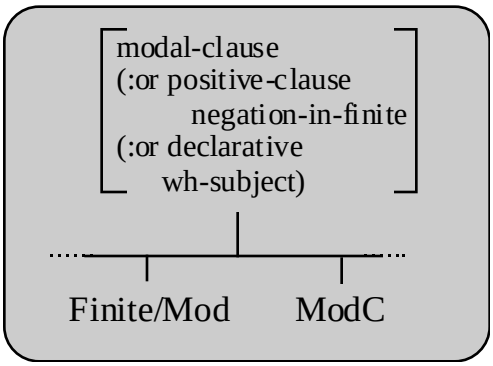

Figure 10.9: An Ordering Partial-Structure Using Conflation

Two important kinds of ordering partial-structures involve the pseudo-roles *Front* and *End*. The first kind show which roles can start a unit, and under what conditions. Figure 10.10 shows a partial-structure which includes the conditions under which the *Thing* role can occur as the first element of a nominal-group.

The second kind shows the condition under a unit can end a structure. For instance, figure 10.11 shows the partial-structure which allows the Thing role to be the final element in a nominal-group.

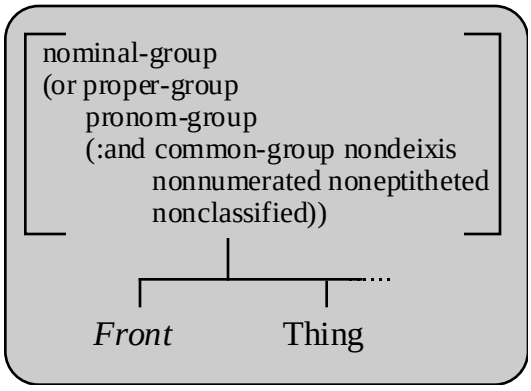

Figure 10.10: A Partial-Structure Ordering Thing at the Front of a Group

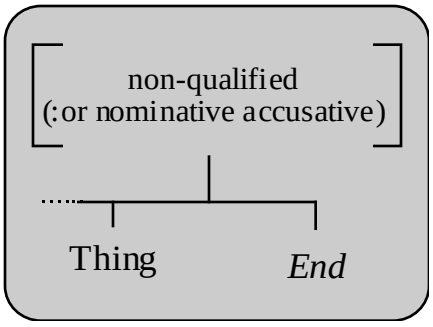

Figure 10.11: A Partial-Structure Ordering Thing at the End of a Group

4.3.4 Summary of the Precompilation Stage

The lexico-grammatical resources are re-represented as sets of partial-structures, which are then used as the parsing-grammar. Figure 10.12 shows the relationships between the Systemic resources and the resources as used for parsing.

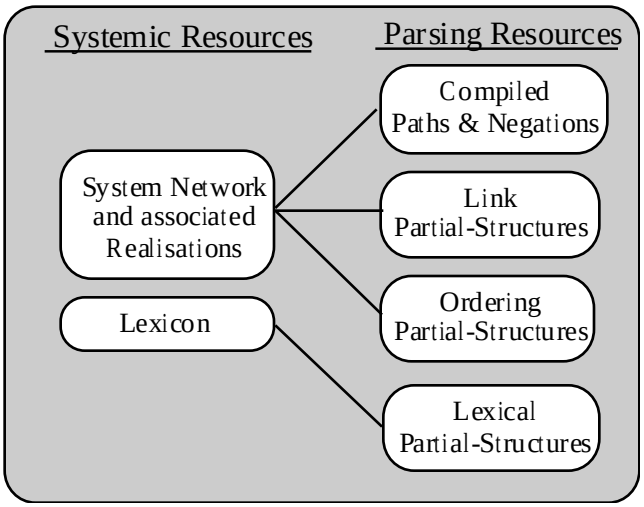

Figure 10.12: Compiling Out the Parsing Grammar

In addition to the partial-structures, the parser needs information about feature combinability. Rather than using the system networks themselves, the *system path* and *system negation* tables (see chapter 8, section 2.5) are used as a resource to control the unification of feature-specifications. For instance, when two partial-structures are unified, the feature-specifications of corresponding units are unified. If the unification fails, then the unification of the partial-structures fails.

5. Lexico-grammatical Analysis

The previous section introduced the three basic partial-structures used in the parser. Each of these is derived from the resources: lexical partial-structures from the lexicon, and linking and ordering partial-structures from combinations of realisation statements.

It remains to be shown how these partial-structures are put together during parsing. This section describes the parsing algorithm used in WAG - the process which transforms a graphological representation into a lexico-grammatical representation.

The general strategy for parsing in WAG is as follows:

- **Bottom-Up:** the WAG parser uses a bottom-up ("data-driven") control strategy, since this strategy makes best use of the available information. A top-down approach needs to make many structural hypotheses before reaching the input string, even using a depth-first strategy. For parsing with a large Systemic grammar, such as Nigel, the bottom-up strategy is most efficient.
- **Breadth-First**: For the same reason, breadth-first parsing is preferred, seeing all the data before building too much structure.
- **Left-to-Right Analysis**: A left-to-right strategy is used: the parse tree is constructed by successively incorporating words from the front of the input string into the parse structure (based on the left-to-right ordering of the source-representation).
- **No Top-Down Filtering**: top-down filtering was initially used in the parser, but was dropped, since it actually slowed the parser down. As the coverage of a grammar increases, the range of items which can fill any particular role increases, and thus the savings of top-down restriction diminishes. For the WAG parser, the time spent in performing the top-down filtering was greater than the time saved in parsing.

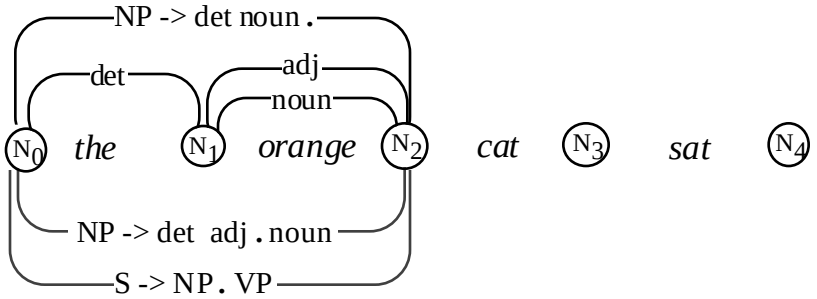

Figure 10.13: A PSG Chart after the Recognition of Two Words

- **Non-Deterministic**: The parser does not attempt to resolve each decision as reached, but rather pursues all alternatives.
- **Chart Parser**: A chart table is used to store successful analyses (passive edges), and analyses under construction (active edges).
- **Exhaustive Search**: The parser produces all possible parses from the input string.

5.1 The Parse-Chart

The parse-chart is a data-structure which keeps track of analyses under way. It stores both:

- completed analyses of sequences of words, e.g., a completed nominal-group. This is called the *passive chart*, and is basically a well-formed substring table (see chapter 9);
- analyses which are still waiting for further elements to complete them, e.g., a nominal-group for which only a Deictic element has been recognised. This is called the *active chart*.

The analyses are called *edges*, because the parse chart is conceptualised as a graph. The point between each word is a node, and the edges connect these nodes together. An edge represents a partial or complete analysis of the words contained between its start-node and its end-node. An edge representing a completed analysis is called a *complete edge*, and an edge representing an analysis in progress is called an *incomplete edge*.

I will use a phrase-structure grammar chart to demonstrate the chart. Figure 10.13 shows the result of parsing the first two words of a sentence. The lines represent edges. Those above the sentence represent complete edges. Thus the first word provides one analysis, the second, two (*orange* is lexically ambiguous). There is also a completed edge which spans these two words, an *NP* analysis.

Incomplete edges are shown beneath the sentence. These represent analyses still in progress. This chart has two incomplete edge, one for an *NP* which has so far recognised the *det* and *adj* elements, and is still waiting for the *noun*. The other is for an *S* unit, and so far an *NP* has been recognised. The '.' in the rules shows how many of the rule's elements have been recognised so far. In a completed edge, the dot is at the end of the rule.

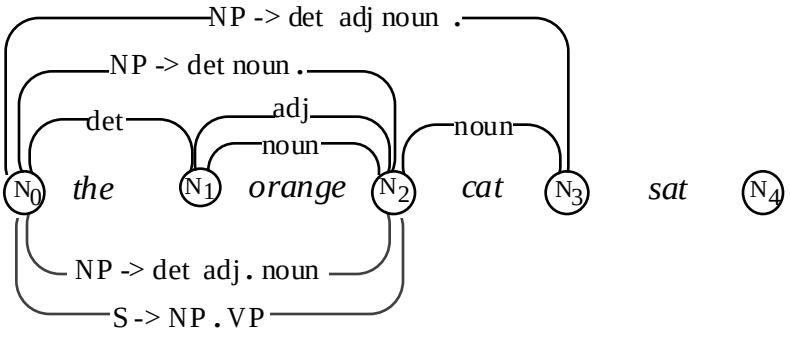

Figure 10.14: Extending an Incomplete Edge

The parse-chart carries all analyses of the sentence forward simultaneously. As each new word is introduced, it is incorporated into each incomplete edge which is waiting for that category of item. For instance, to incorporate *cat* into the parse-chart, there is one incomplete edge waiting for a noun, so this edge can be extended, producing a new complete edge, as shown in figure 10.14.

The incomplete-edges in the example all included at least one recognised element. Part of the chart-parsing strategy involves introducing, at each node, a set of *starting edges.* Starting edges are incomplete-edges with no element recognised yet. We introduce one such edge for each rule in the phrase-structure grammar. For example, one of these edges would be labelled 'NP -> . det noun', with the dot before the first element. These starting edges represent the potential analyses which can start at the node. As we introduce each new lexical-item, we see if it can extend any of the starting-edges (as well as the other incomplete edges).

A Systemic chart-parser operates in principle much like the PSG chart-parser described above. There are however some difference:

a) the categories which label the edges are more complex. Each edge represents a partial-structure, rather than a PSG rule;

b) an incomplete edge doesn't include a list of constituents still to recognise: the edge only shows the elements which have been recognised so far. To discover what can extend the edge, ordering partial-structures are used to predict what can follow the last-recognised element.

c) While a PSG chart requires one starting-edge for each rule in the grammar, a Systemic chart-parser requires only a single starting edge per node. It is simply a partial-structure for the front of a grammatical unit, with minimal feature information -- see figure 10.15. This partial-structure, in conjunction with the various ordering partial-structures involving *Front*, allows us to predict what roles can be first element of an edge.[5]

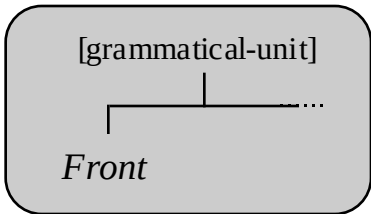

Figure 10.15: A Systemic Starting-Edge

[5]This is not to be confused with top-down filtering: what we do here is just predict the functional roles which can follow *at this level of structure*, while top-down filtering involves prediction of the range of grammatical elements which can follow at this level or any level below (the first constituents).

5.2 Advancing Through the Parse String

Figure 10.16 provides a flowchart of the basic parsing algorithm. The process selects the next graphological-word from the front of the input string, from which all possible lexical analyses are derived. One lexical partial-structure is produced for each analysis. These lexical partial-structures are then incorporated into the parse-chart, one at a time. For instance, assuming we are parsing “They will come”, the lexical analysis of the first word produces a single partial-structure, shown in figure 10.17 (repeated from figure 5 above). Lexical analysis and Grammatical analysis are thus interleaved - the processor lexically analyses a word, then incorporates this analysis into the parse-tree.

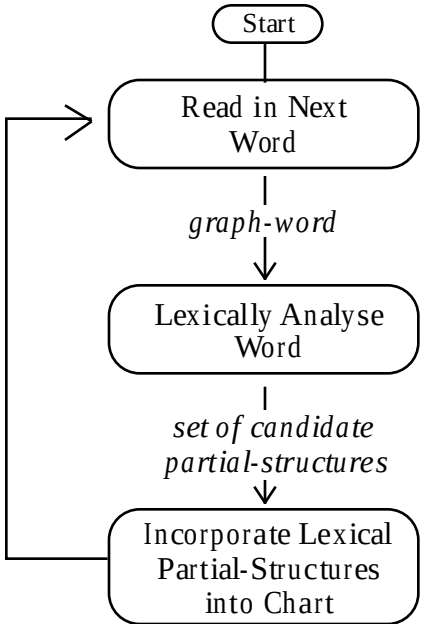

Figure 10.16: Advancing Through the Parse String

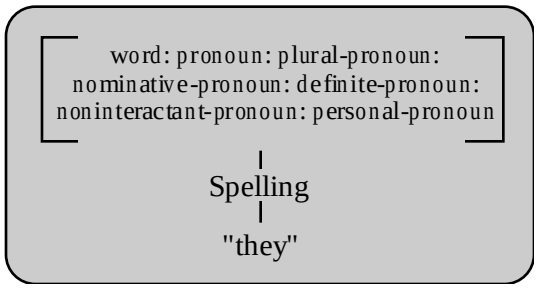

Figure 10.17: A Partial-Structures Derived from a Lexical-Item Definition.

5.3 Incorporating Partial-Structures into the Chart

As each lexical partial-structure is recognised, it needs to be incorporated into the parse-chart. I will now describe this incorporation in more detail. Figure 10.18 shows the basic algorithm. The stages of this process are described below.

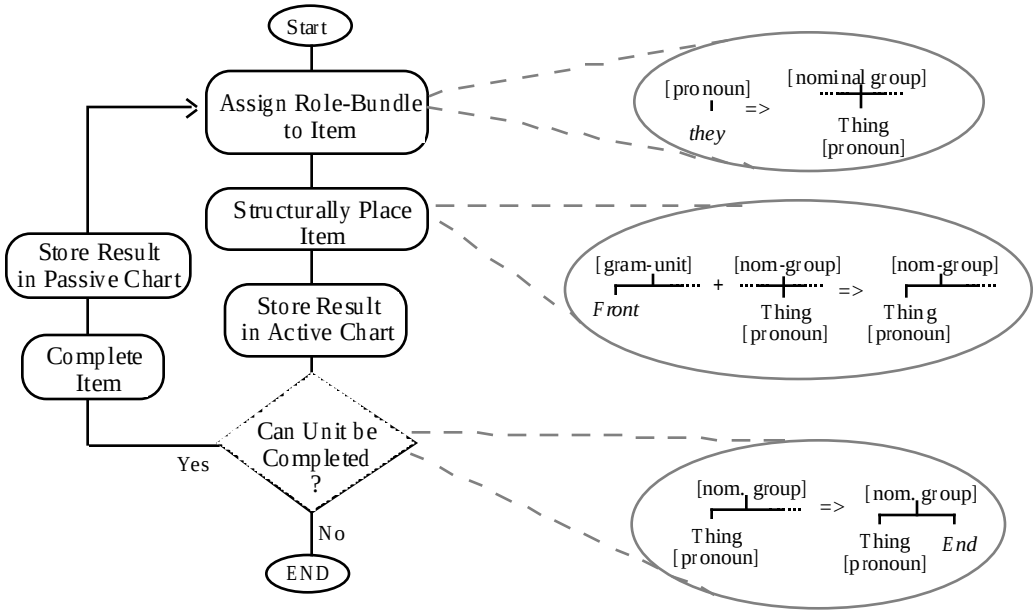

Figure 10.18: Incorporating Lexical Items Into the Parse Chart

5.3.1 Role Assignment

The first step in the incorporation of a lexical partial-structure into the parse-chart involves discovering what constituency roles the unit can fill. To do this, we attempt to unify each of the linking partial-structures with the lexical partial-structure, to see which are compatible.[6] Figure 10.19 shows the lexical partial-structure from above unifying with one of the linking partial-structures, producing a group-level partial-analysis of the pronoun.

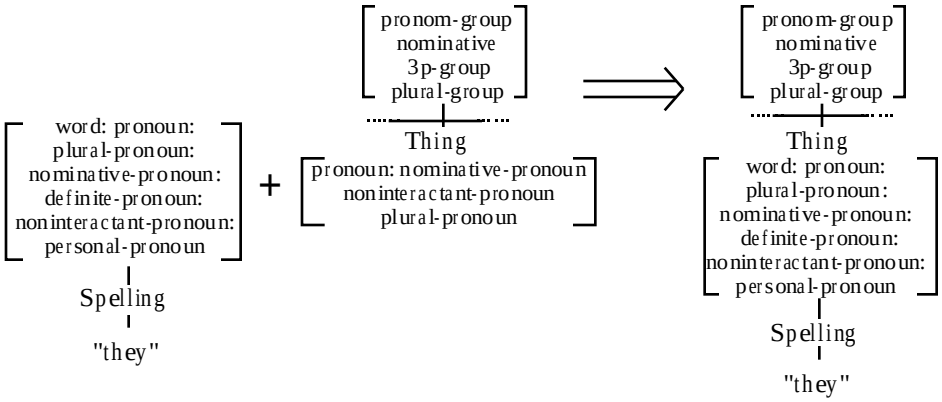

Figure 10.19: The Role-Assignment Operation

[6] In practice, various methods are used to limit the number of linking partial-structures actually matched against the lexical partial-structure.

5.3.2 Structural Placement

The role-assigned partial-structure has resulted from the analysis of a single word in isolation. We now need to incorporate it into the parse-chart, using it to extend one or more of the incomplete-edges in the chart.

At this point we have only one incomplete-edge on the chart -- the starting-edge. To unify the partial-structure with this starting-edge, we need to first fetch the ordering partial-structure which constrains the *Front ^ Thing* ordering. Figure 10.20 shows the unification of these three partial-structures, resulting in an incomplete-edge for the first word of the sentence.

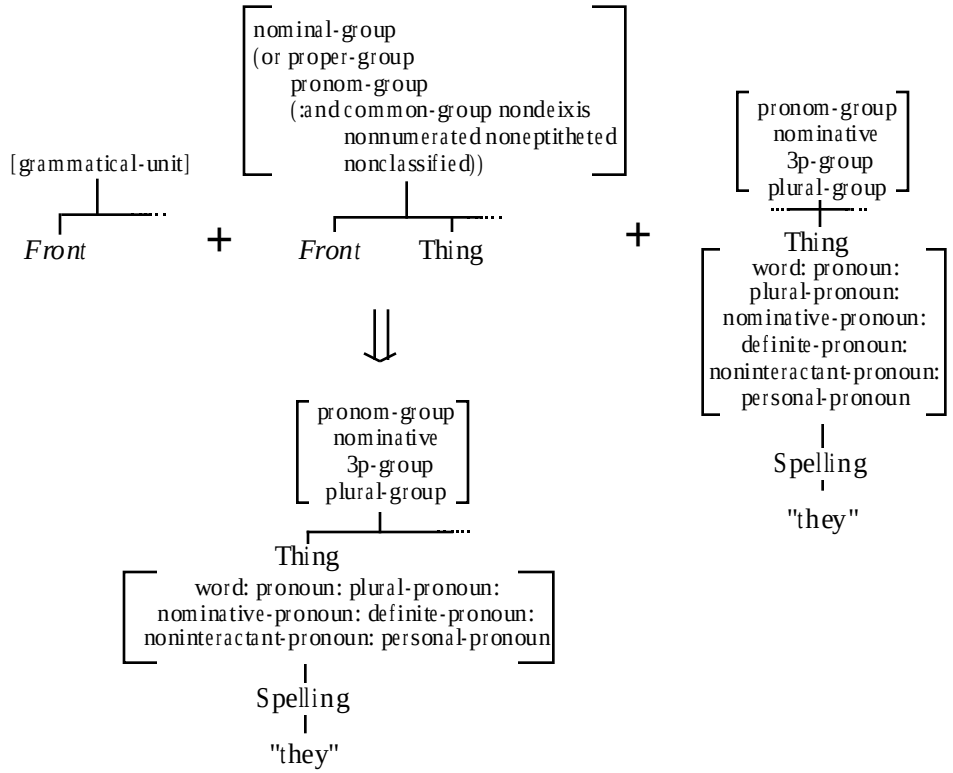

Figure 10.20: Placing a Partial-Structure at the Beginning of a Unit

5.3.3 Completing A Unit

After an incomplete edge has been extended, we need to test if the extended edge can be transformed into a completed edge. To do this, we look for the ordering partial-structure which allows *End* to be the next element, and unify it with the edge. Figure 10.21 shows this process. The partial-structure on the left represents the incomplete-edge produced in the last stage. The linking partial-structure which allows *Thing^End* is retrieved from the parsing resources, and unified with the incomplete-edge. The result is a completed edge, an analysis of a nominal-group.

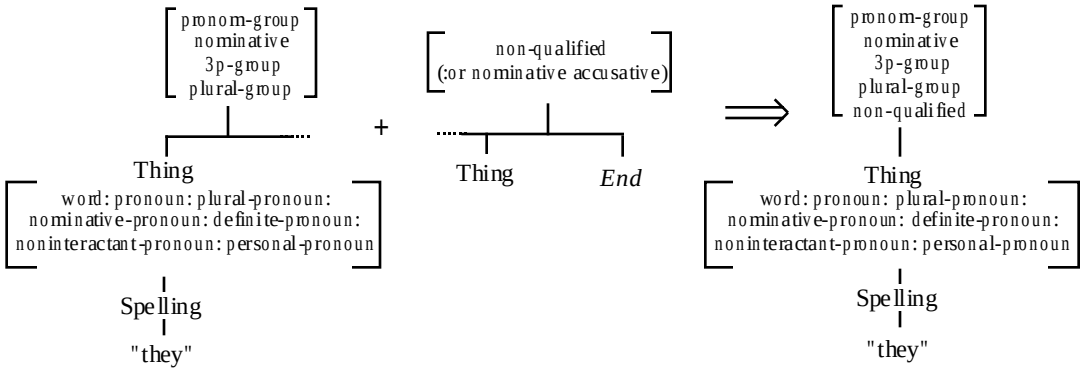

Figure 10.21: 'Completing' a Partial-Structure.

The completion of the nominal-group is only one of the structural possibilities. The incomplete edge also remains in the chart, perhaps to be extended by later occurring items, e.g., waiting for a Qualifier (e.g., "they who died").

5.3.4 Recursion of These Steps

This last step resulted in a completed nominal-group. We must then repeat the role-assignment, structural placement and completion steps for this edge. Figure 10.22 shows one possible role-assignment for the completed edge from figure 10.21: the nominal-group as *Subject/Agent.* Another candidate would be *Subject/Medium.*

5.3.5 Summary

I have described parsing as the recursive application of three steps -- role-assignment, structural-placement and completion. As each lexical-item is incorporated into the parse-chart, the parser then moves on to incorporate the next lexical-item, until all items are incorporated.

5.4 Lexico-Grammatical Output

After the last lexical-item is incorporated into the parse-chart, we can extract the successful analyses from the chart. Successful analyses are completed-edges which span the entire sentence. Some sentences will produce only one successful analysis, while others will produce a large number of alternative analyses. Multiple analyses exist

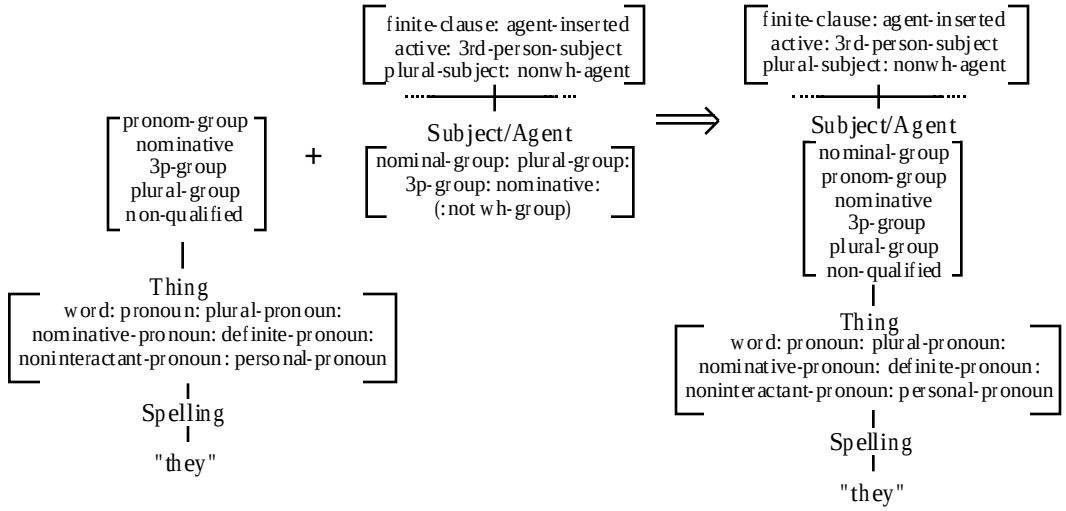

Figure 10.22: Role-Assignment -- Second Round

because of grammatical ambiguity -- the resources allow alternative interpretations of the input string. This may be due to multiple senses of lexical items (e.g., a dog's *bark* vs. a tree's *bark*), or perhaps due to different ways of attaching units together.

The main ambiguity in my experience is due to the *pp-attachment* problem -- a prepositional phrase at the end of a sentence may attach to the clause (as a Circumstance), or to the Complement (as a Qualifier). For instance, in "I saw the girl with a telescope", it is not clear whether "with a telescope" is attaches to *saw* (what I saw with), or to *the girl*. (what the girl had). Given two or three prepositional-phrases at the end of a clause, the number of possible structural interpretations can become quite large.

Figure 10.23 below shows one of the parse-trees produced from parsing "A user-password is a character string consisting of a maximum of eight alpha-numeric characters.". Several analyses were produced, corresponding to different attachments of the non-finite clause -- "consisting of..." -- and also of the prepositional-phrase -- "of eight...". The diagram does not show the grammatical features, which are also recovered.

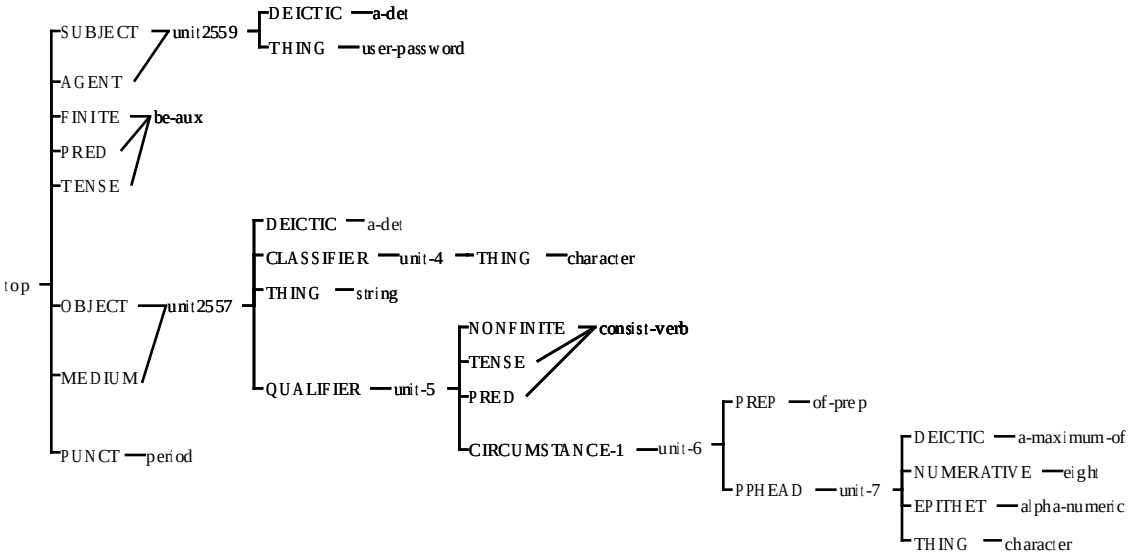

Figure 10.23: The Parse-Tree for "A user-password is a character string consisting of a maximum of eight alpha-numeric characters" (computer-drawn).

5.5 Comparison With Other Systemic Parsers

There have been seven prior approaches to Systemic parsing: Winograd (1972), McCord (1977), Cummings & Regina (1985), Kasper (1988a, 1988b, 1989), O'Donoghue (1991a, 1991b), Bateman *et al*. (1992), and Weerasinghe & Fawcett (1993).

Most of these parsers used grammars too small to produce the complexity problems faced by larger grammars, e.g., Winograd (1972), McCord (1977), Cummings & Regina (1985), and Bateman *et al*. (1992). The first three of these parsers also used reduced forms of the Systemic formalism.

Two of the parsers rely on the input sentence being pre-parsed using a grammar from another formalism, e.g., Kasper's parser initially parses the sentence using a phrase-structure grammar (PSG). This PSG forms a *context-free backbone* to the Systemic grammar. A set of constraints are then applied which builds up the Systemic representation corresponding to the PSG analysis. Bateman *et al*. (1992) also depends on a pre-analysis, using a HPSG parser, which produces a skeletal function structure for the

sentence.[7] A unification system (Typed Feature Structures (TFS) - see Emele & Zajac 1990) then uses the Systemic resources to produce a full lexico-grammatical and ideational analysis of the clause.[8]

O'Donoghue (1991a, 1991b) and Weerasinghe & Fawcett (1993) both use Fawcett's Systemic formalism (Fawcett *et al.* 1992), which offers less parsing complexity than the Hallidayan formalism (used by WAG). Fawcett divides his analyses into semantic representations (the feature selections for each unit) and the syntactic representation (a structure involving roles and syntactic classes, but no features). Parsing concerns the construction of the syntactic representation, and both the parsers for this formalism perform the *syntactic* analysis in isolation from the system networks. Since each unit in the syntactic representation involves only a single class, rather than a collection of features, the parsing task is simplified significantly.

Weerasinghe & Fawcett (1993) further simplifies the formalism by assuming that each unit fills a single functional role (e.g., no conflation), with one exception, the *O* element (the Finite in Hallidayan terms), which seems to be handled specially. They thus avoid the second major cause of complexity in Systemic parsing -- functional layering.

O'Donoghue (1991a) comes closest to the WAG parser, in that he attempts to automatically re-compile the Systemic grammar into a form more oriented to parsing. However, his parsing resources are not compiled from the grammar itself. Instead, he uses the Genesys generation system (Fawcett & Tucker 1990) to generate a large number of random sentences.[9] A parsing-grammar is produced by analysing this corpus. He calls the grammar *Vertical-Strip Grammar* (VSG), and he describes it as follows:

> "In a VSG, rather than considering the horizontal slices through the tree -- the phrases -- we consider vertical slices. The vertical slices are called strips, hence the name VSG. A strip is a vertical path of nodes in the tree starting at the root and ending at a lexical node." (p4).

One problem with this approach is that his compiled grammar is not *equivalent* to the Systemic grammar from which it is derived, but will only parse a (large) subset of the sentences which that grammar allows.

His approach also suffers from a high level of complexity. To parse, each lexeme is assigned the set of strips for which it can fill the leaf node. Working left-to-right through the input string, the parser attempts to unify the potential strips for each lexeme. Assume we have 10 possible strips per lexeme (a small estimate), and have 8 words in a sentence (again small). The possible combinations of these strips (worst case) is 10 x10 x 10 x 10 x 10 x10 x 10 x 10, or 10^8, which is quite large. Constraints (e.g., ordering constraints), and heuristics can reduce the actual number of strip combinations, but this method would be unusable for a grammar of reasonable coverage.

5.6 Summary

My research has resulted in a parsing system which parses using a large systemic grammar, without pre-parsing with a non-Systemic grammar. The parser uses the Hallidayan formalism, which is the offers more parsing complexity than the Fawcett formalism. The WAG parser is the first parser to fit these conditions.

---

[7] In Bateman *et al.* (1992), the starting structure is produced by hand rather than by the HPSG parser. The HPSG pre-parser was added later (Bateman, personal communication).

[8] Only those resources needed to parse *Kim ate every cookie* were included. These resources included lexico-grammatical and ideational resources; and declarativised chooser-inquiries.

[9] A cut-down version of the grammar is used, including around 450 systems; and 500 realisation rules.

The major factor which makes this possible is the re-representation of the Systemic grammar in a form more suitable for bottom-up parsing. To summarise the preparation of the parsing-grammar, the following steps were followed:

- This grammar is reduced in size, by applying register-restrictions, leaving a less complex grammar, but a grammar which still handles the bulk of the phenomena in the target texts. This is not a necessary phase -- when using WAG's Systemic grammar, which is less complex, register-restriction does not need to be applied.
- The grammar is then re-compiled to produce the parsing-grammar -- the partial-structures used in parsing, as described in section 4 of this chapter. For the Nigel grammar, this process takes approximately 2 minutes using Sun Common Lisp on a Sun Sparc II.
- The parsing-grammar is then used to parse sentences.

The type of re-representation is important. The re-representation I have developed allows efficient parsing, because it provides the answer to two questions which a bottom-up Systemic parser asks:

- What element can come next;
- What is the function-potential of a given unit;

The WAG parsing-grammar is automatically compiled. If this was not the case, then the parsing-grammar would need to be adjusted whenever the generation grammar is adjusted -- the usability of the system is lessened. Also, the compiled grammar is *equivalent* to the source-resources: capable of recognising any structure which the grammar and lexicon allow, and rejecting any analysis which the resources do not allow. This is not however true for O'Donoghue's VSG resource.

Other important features of the WAG parsing-grammar are:

- No cover-grammar is required for parsing;
- The formalism of the grammar (before re-compilation) is not simplified to make the parsing task easier: the theory should not be sacrificed for the sake of the application.
- The grammar is represented declaratively, not procedurally as in Winograd's approach. This means that the grammar can be modified - or completely replaced - without re-programming the system.

The WAG parser is able to handle grammars of a reasonable size, while still producing a result in a reasonable time. For instance, using a version of the Nigel grammar with 500 clause and group-rank features, the parser analysed the sentence "A user-password is a character string consisting of a maximum of eight alpha-numeric characters." in 15 seconds (using Sun Common Lisp on a Sun Sparc II). With the WAG grammar, which is less complex, the same sentence is parsed in under two seconds. These times compare favourably to other Systemic parsers, using grammars of similar coverage and complexity. This time is however slow compared to context-free parsers, but the Systemic formalism offers a lot more power than context-free grammars.

6. Micro-Semantic Analysis

I have so far described how a sentence can be analysed lexico-grammatically. The next stage of processing produces a micro-semantic analysis from the lexico-grammatical analysis (see figure 10.24). Lexico-grammatical analysis may produce a number of analyses, but I will assume for the following discussion that only one of these analyses is selected for semantic analysis.

The micro-semantic analyser draws on three resource-modules:

- **The Lexicon**: providing the ideational features for each lexical-item in the parse-tree;
- **Selection Constraints on Grammatical Features**: providing the ideational, interactional and textual meaning encoded in each grammatical feature;
- **Micro-Semantic Resources**: used to test whether semantic constraints are mutually compatible.

6.1 Information Flow

Chapter 7 introduced three information-flow architectures: *staged*, *interleaved* and *integrated*. These architectures can be characterised in relation to micro-semantic analysis as follows:

Staged Analysis: lexico-grammatical analysis is completed before micro-semantic analysis is started.

Interleaved Analysis: the lexico-grammatical analyser and the micro-semantic analyser alternate construction, passing control between each other as required.

Integrated Analysis: as with interleaved analysis, lexico-grammatical analysis and micro-semantic analysis are intermixed. However, a single processor is used, which does not distinguish between the two representations.

In WAG, I have followed a *staged* approach, performing semantic analysis only after lexico-grammatical analysis is complete. My main reason for choosing this approach is that lexico-grammatical analysis is relatively cheap compared to the computation performed in applying semantic constraints (semantic constraints contain a high degree of disjunction and negation). By completing the grammatical analysis before applying semantic constraints, alternative readings at the phrase or word level may be eliminated on purely syntactic grounds. For instance, in "the orange cat sat on the mat", the interpretation of “the orange" as an NP is eliminated syntactically: semantic analysis of this unit does not need to be considered at all. Staged architectures are also easiest to implement, since they assume an autonomy of levels.

There are arguments, however, for an interleaved or integrated approach. As each lexico-grammatical unit is completed (word, phrase or clause), a semantic analysis of that unit can be performed. Grammatical ambiguity can be resolved as it is discovered. For instance, the grammar and lexicon allows two readings of "the orange seller" - a person who sells $\text{oranges}_{\text{noun}}$, or a seller who happens to be coloured $\text{orange}_{\text{adj}}$. Using the

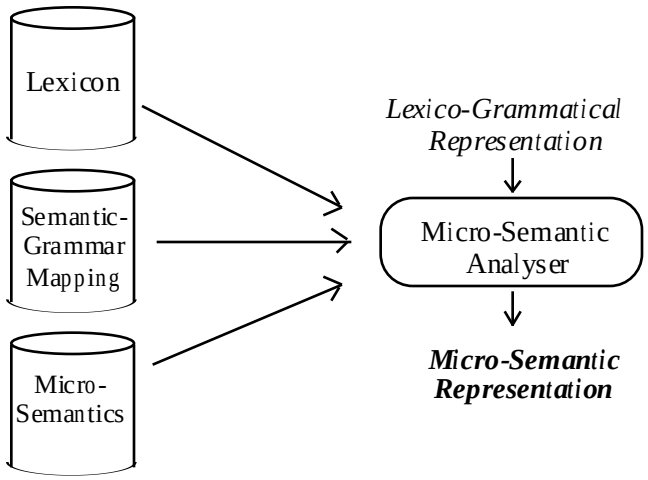

Figure 10.24: The Micro-Semantic Analysis Stage

ideational resources, it could be discovered that people come in shades of red, yellow, pink, black, white and brown, but never orange. The semantics could thus reject the adjectival analysis without seeing the rest of the sentence.[10]

6.2 Control Strategy

In micro-semantic analysis, the lexico-grammatical representation is the source, and the micro-semantic representation is the target. Since the WAG resource model associates the interstratal constraints with grammatical features (see chapter 6), it is easiest to use a source-driven strategy for semantic analysis: the lexico-grammatical representation drives the analysis. Each unit of the parse-tree is taken in turn, and the selection-conditions of its features are asserted. If the unit is at word-rank, then the analyser recovers the word's ideational features from the lexicon, and assigns them to the word's Referent. This process is described in more detail below.

A target-driven control-strategy could also have been used -- traversing the system networks of the micro-semantics, and choosing features as appropriate for the given lexico-grammatical input. Traversal needs to be repeated for each unit of the semantic structure, e.g., we might first traverse the speech-act network to construct the speech-act, then traverse the ideational network for each element of the proposition. Such an approach mirrors the way in which most Systemic generators operate (e.g., WAG, Penman -- see chapter 11), except that it is the semantic networks which are traversed rather than the lexico-grammatical network. I have not followed this approach, since the inter-stratal mapping constraints are indexed in terms of lexico-grammatical features, and it would be difficult to re-index these resources in terms of semantic features.

6.3 The Micro-Semantic Analysis Process

The lexico-grammatical analysis of "Is Mary coming tomorrow?" would produce the representation shown in figure 10.25. I will describe how the micro-semantic analysis is derived from this analysis.

Is	*Mary*	*coming*	*tomorrow*
[clause:modal:indicative:interrogative:yes-no:temporally-located]			
Finite/ Prog	Subject	Pred/ ProgC	Circumstance
	[nominal-group:proper-group]		[adverbial-group]
	Head		Head
[be-aux]	[proper-noun]	[lexverb: ing-verb]	[adverb]

Figure 10.25: A Lexico-Grammatical Representation

Each unit of the parse-tree is mapped in turn, using the information in that unit to build up the semantic representation. A bottom-up strategy is used, mapping first word-rank units, then their parent-units.

[10] Although note that contexts can be found for almost any combination of quality and thing. Semantic filtering should thus be seen as a means of eliminating unlikely readings, but should not be depended on if one desires to accurately recognise unlikely readings when they are intended..

6.3.1 Mapping of Word-Rank Units

The first step in mapping involves asserting the semantics for each lexical-item: the ideational features of each lexical-item are retrieved from the lexicon, and assigned to the unit's Referent.[11] The lexicon entry for the word "Mary" is shown below:

```
(def-lexical-item
  :name mary1
  :spelling "Mary"
  :grammatical-features (word noun proper-noun not-determiner-required)
  :semantic-features    (thing conscious 3d-object human female))
```

To map the grammatical unit which is expressed by "Mary", we first provide it with a Referent role. The filler of this role is then provided with the features taken from the *:semantic-features* slot of the lexical-item: *(:and thing conscious 3d-object human female).* This process is repeated for each word-rank unit.

6.3.2 Mapping Grammatical Units

After the word-rank items are mapped, the units higher in the parse-tree are mapped. For each of these units, the unit's features are used to drive the mapping. The selection-constraint associated with each of the unit's features are asserted. The constraint for the feature *nominal-group* is shown below:

```
(def-constraint  nominal-group
  (:and (:type Referent (:or thing process ideational-relation))
        (:same Referent Thing.Referent)))
```

In figure 10.25, this constraint would be applied to the *Subject*, since it has the feature *nominal-group* in its selection-expression. The constraint tells us two things: firstly, that the *Referent* of the *Subject* will be either a *thing, process,* or *ideational-relation*. The second constraint tells us that the *Referent* of the *Subject* is the same as the *Referent* of its *Head* role (passing the *Referent* role between the parent and its *Head*). Since the *Referent* of the *Head* is already known from the lexical mapping, the Subject's Referent is equated with that unit.

Some constraints, like that for the *nominal-group*, reflect ideational meaning, but others supply information about the interactional and textual meaning, as shown below:

```
(def-constraint  wh-agent
  (:and (:type *Speech-Act* elicit)
        (:same  Agent.Referent *Speech-Act*.Required)))

(def-constraint  possessive-deixis
  (:and (:relevant Referent.Owner)
        (:same Referent.Owner Deictic.Referent)))
```

6.3.3 Micro-Semantic Representation

The information from the selection-constraints and the lexicon weaves together, building up a micro-semantic representation for the sentence. This representation contains ideational, interactional and textual information. Figure 10.26 shows a computer-generated graph of the speech-act structure resulting from the micro-semantic analysis of

---

[11] Some closed-class lexical items do not have associated semantic features, since their semantics is already associated with the grammatical features which determine the lexical item, e.g., determiners, intensifiers, auxiliary verbs, etc. Some closed-class items are not totally determined by the syntax, for instance, the gender of pronouns, in which case their semantics does need to be recovered from the lexicon.

the sentence “John hit Mary”. The semantic features of each unit are also recovered, but not shown in the example. Textual information is also not shown.

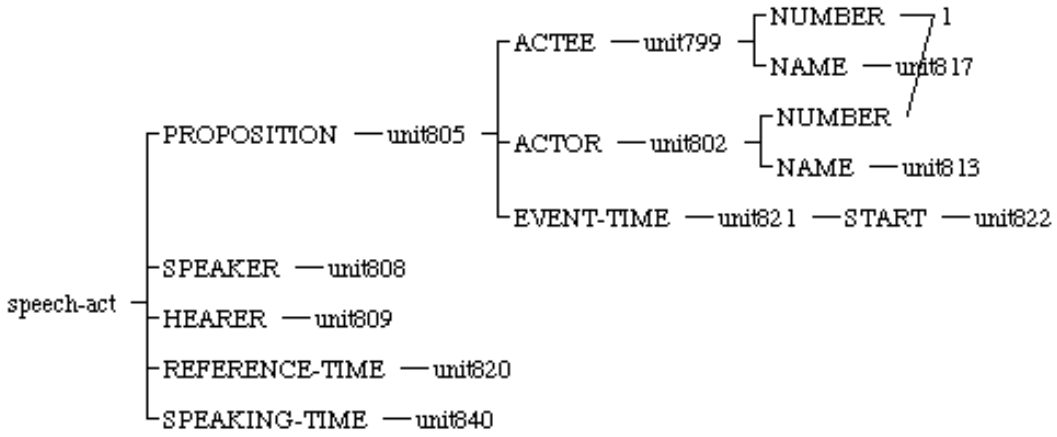

Figure 10.26 Micro-Semantic Analysis of “John shot Mary.”

6.3.4 Evaluation

The WAG system is successful in that it produces a micro-semantic analysis of an input sentence. However, this success needs to be moderated, because the process is not greatly robust. A simple sentence was used in figure 10.26, because the level of complexity involved in the semantics-grammar mapping is rather high. A large number of the mapping constraints contain disjunction, and the assertion of these results in combinatorial explosion. I am continuing research into this process, hoping to reduce the level of complexity.

7. Summary & Conclusions

This chapter has outlined the four stages of processing in the WAG analyser:

- Graphological Analysis;
- Lexical Analysis;
- Grammatical Analysis (Parsing);
- Micro-Semantic Analysis.

I have provided details of each of these stages, showing how each stage builds on the results of the stage before.

This is the first Systemic parser which parses with the full Hallidayan formalism without use of a non-systemic formalism to segment the sentence. There have been various systems which parse using a simpler Systemic formalism (Winograd 1972; McCord 1977; Cummings & Regina 1985; Weerasinghe & Fawcett 1993), and others which rely on a non-Systemic formalism to segment the input string (Kasper (1988a, 1988b, 1989): Phrase Structure Grammar; O'Donoghue (1991a, 1991b): his `Vertical Strip Grammar'; and Bateman *et al*. (1992): Head-driven Phrase Structure Grammar).

The important characteristics of the WAG parser were described in section 5.4 above. Probably the most important contribution of this chapter is the discussion of compiling out a parsing-grammar, and the composition of this resource. This re-compilation stage is necessary to produce a parser capable of handling large Systemic grammars, without a cover grammar.

Future work will attempt to increase this parsing speed. Four directions are being followed:

a) Finding alternative re-representations of the Systemic resources which facilitate the parsing process;

b) Research into the automatic extraction of a ‘context-free’ backbone (or a grammar closer to context-free) from the Systemic resources;

c) Moving more processing to the pre-compilation stage;

d) Reducing the complexity of the description without reducing its coverage.