Precompiling Systemic Grammar for Parsing

Michael O'Donnell
Department of AI,
University of Edinburgh,
80 South Bridge, Edinburgh.
EH1 1HN, UK.
email: mick@darmstadt.gmd.de
Keywords: Parsing, Systemic Formalism

January 19, 1996

Abstract

Parsing with a large Systemic grammar produces severe complexity problems. Automatic recompilation of the Systemic grammar into a form more suitable for parsing can reduce the level of parsing complexity. This paper describes the form of one such parsing-grammar, and its use in parsing.

1 Introduction

A Systemic grammar (Halliday 1985; Hudson 1971; Bateman 1989; Martin 1992), when used for parsing, suffers severely from complexity problems, due for instance to the degree of disjunction present in the grammar, and also to the multiple layers of function-structure allowed, e.g., a nominal group may simultaneously fill the *Subject*, *Theme*, *Agent*, and *Actor* role in the grammatical structure (see Matthiessen *et al.* 1991, O'Donnell 1993, for more details). Systemic grammars also do not include a context-free backbone, usually included in feature-based formalisms to reduce parsing complexity. Parsing with Systemic grammars is thus that much more difficult.

One might ask the question: if Systemic grammars seem so unsuited to parsing, why bother? One answer to this question is that, over the last decade, several wide-coverage Systemic grammars for generation have been developed, including the Nigel grammar from the Penman Text Generation System. (Mann 1983; Mann & Matthiessen 1985; Matthiessen 1985). Since such resources exist, it is desirable to use them for analysis, as well as generation.

To avoid complexity problems, prior parsers for Systemic grammars have included some kind of limitation, either resorting to a simplified formalism, or augmenting the Systemic analysis by initial segmentation of the text using another grammar formalism.

My goal was to parse using the a large Systemic grammar, without any of these limitations. I have developed a parser which handles a large subset of the Nigel grammar, using one third of its 1500 types[1] (or 'features' in Systemics). This is still a large grammar in terms of current parsing technology. The level of success of these efforts shows that the complexity problems involved in Systemic parsing are not insurmountable.

Most of the complexity has been avoided by recompiling the grammar into a form more suited for parsing. This recompilation is automatic, in the sense that no human intervention is required. This is necessary because the parser is only one module of a Linguist's Workbench.[2] Any change to the grammar needs to be automatically available for both parsing and generation.

Section 2 will provide a brief outline of the Systemic formalism. Section 3 provides more detail on the reasons why Systemic grammars offer complexity in parsing, and describes how prior approaches handled these complexity issues. Section 4 describes the forms in the compiled grammar, and section 5 how they are derived. Section 6 outlines its use in parsing. Section 7 then outlines the advantages of this approach for Systemic parsing.

2 Systemic Grammar

2.1 System Networks

Systemic grammar uses an inheritance network to organise grammatical features. A Systemic inheritance network is called a *system network*, and

[1] 'Type' is meant in the sense of Typed- Feature Structures, cf. Carpenter 1992; Emele & Zajac 1990.

[2] Workbench for Analysis and Generation (WAG): a package of tools for working with Systemic grammars, including syntactic and semantic analysis, sentence generation, knowledge representation, and various tools for resource maintenance (lexical acquisition, grapher, hypertext grammar interface). See O'Donnell 1994a, 1994b, 1995.

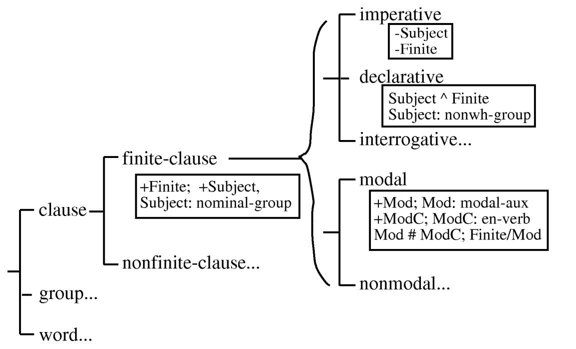

Figure 1: A Fragment of a System Network

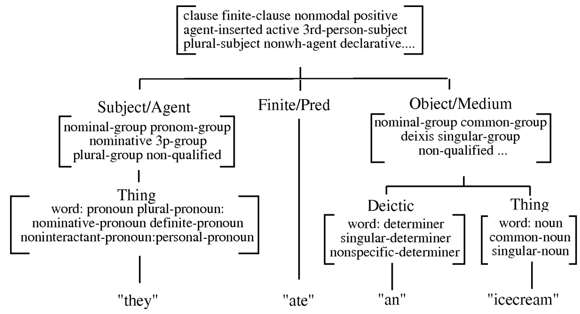

Figure 2: A Systemic Structure

is used to organise the co-occurrence potential of grammatical features, showing which features are mutually compatible, and which are incompatible. It consists of a set of *systems*, each of which is a set of mutually exclusive features. There is also a *covering* relation between the features of a system, meaning that if the entry condition (the logical condition on a system) of the system is satisfied, then one of the features in the cover must be selected. Figure 1 shows a few systems from a small grammar for English. It includes 4 systems, representing various grammatical distinctions.

Each feature inherits the properties of features to its left in the network. Note that the system network may be logically complex, since entry conditions may consist of conjunctions and disjunctions of features. Systemics thus allows multiple inheritance, both in terms of conjunctive and disjunctive inheritance.

2.2 Realisation Statements

Systemic features may have associated realisation statements – the structural consequence of the feature. The boxes under each feature in Figure 1 shows the realisations of the feature. The realisation operators used (from Penman, see Matthiessen & Mann 1985) are as follows:

- **Insert** e.g., *Finite=[]*: the function must be present in the structure.
- **Conflate** e.g., *Modal/Finite*: the two functions are filled by the same grammatical unit (equivalent to *path equality* in unification-style formalisms).
- **Order** e.g., *Subject ^ Finite*: the functions must appear in the surface structure in the indicated order. In this example, the *Subject* is sequenced directly before the *Finite*. Any number of elements can be sequenced in a single rule, and optional elements can be indicated.
- **Partition** e.g., *Thing ... End*: the functions must appear in a particular order, but not necessarily immediately adjacent (linear precedence).
- **Preselect** e.g., *Subject: nominal-group*: the function must be filled by the designated type of element.
- **Lexify** e.g., *Deictic = "the"*: the function must be filled by the specified lexeme.
- **Presume** e.g., -*Subject*: the specified function, while present in the structure for ordering purposes, is for other purposes not present in the structure. Used for phenomena such as grammatical ellipsis.

2.3 Systemic Structures

A system network and associated realisation statements describe a grammatical potential – the range of grammatical structures posited by the theory. An instance from this potential is a *systemic structure*. A typical Systemic structure appears in Figure 2. Each element of the structure is described both in terms of functions (one or more – the syntactic roles this element is filling), and a set of features.

3 The Problem of Parsing with Systemic Grammar

3.1 The Problems

A Systemic grammar does not use a context-free backbone – there is no phrase-structure component. As a consequence, a Systemic parser does not deal with a single category at each node, but rather has to deal with a complex description – a complex feature description (the *selection-expression* for the unit), and a complex functional description (the *function-bundle* of the unit). These complexities are discussed below:

1. **Complexity of the Type Hierarchy**: The type hierarchy of the Nigel grammar is one of the largest in use in NLP systems, with 1500 grammatical types represented. These types are organised in terms of disjunctions (systems). The Nigel grammar, containing around 700 systems, is thus highly disjunctive. System networks also use both conjunctive and disjunctive inheritance, and cross-classification (called *simultaneous systems* in Systemics), all of which further compounds the Type complexity.

2. **Complexity of the Function Structure**: A Systemic structure relates units to their constituents in terms of functions. One constituent may be related to its parent through several functions. The combining of realisation rules during parsing is complex, since we need to consider the possible combinations of various functions (called *function-bundles*). For instance, the preselections affecting a grammatical unit may not pass through a single function, but may be channelled through a number of conflated functions (e.g., *Agent* and *Subject*). Ordering of elements also needs to take conflations into account.

3.2 Prior Solutions to these Problems

There have been seven prior approaches to Systemic parsing. Most of these 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). The 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 to produces a skeletal function structure for the sentence.

There are two parsers for Fawcett's Systemic formalism (Fawcett 1980; Fawcett *et al.* 1993): O'Donoghue (1991a, 1991b) and Weerasinghe & Fawcett (1993). This formalism offers less parsing complexity than the Hallidayan formalism: system networks are not used for parsing – the parse-grammar assigns a single syntactic class per unit.

Both of these parsers use a parsing-grammar, rather than the Systemic grammar directly. However, neither is automatically compiled from the

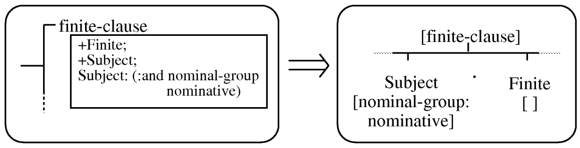

Figure 3: Re-Representing a Feature & Realisation as a Partial-Structure

Systemic grammar: O'Donoghue's is derived by analysing a corpus of randomly generated sentences, while Weerasinghe & Fawcett's is prepared by hand.

4 The Form of the Parsing Grammar

The Systemic formalism has a top-down orientation: it mainly presents what constituents each type of unit can have. 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 functions a given unit can fill* (the function potential of the unit), rather than *what constituents it can have.*

This section describes the automatic recompilation of the usual top-down oriented Systemic grammar into of a bottom-up oriented grammar. A grammar tailored for bottom- up parsing allows more efficient parsing.

4.1 Partial-Structures

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 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 systemic structure (as in figure 2), 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.

The whole grammar could be re-represented in this manner, representing a shift from a paradigmatically-organised grammar (emphasising features) to a syntagmatically- organised grammar

```
(def-lexical-item
  :name they-pron
  :spelling "they"
  :sample-sentence  "They shoot horses, don't they?"
  :grammatical-features
     (grammatical-unit word noun-word pronoun
      personal-pronoun plural-noun
      noninteractant-pronoun definite-pronoun)
  :semantic-features
     (ideational-unit general-domain thing
      countable-thing nonsingular-thing)
  :spelling-exceptions
     ((accusative-pronoun "them")
      (genitive-pronoun "their")))
```

Figure 4: Lexical Item Definition for "they"

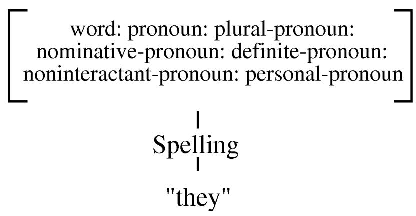

Figure 5: A Partial-Structures Derived from a Lexical-Item Definition

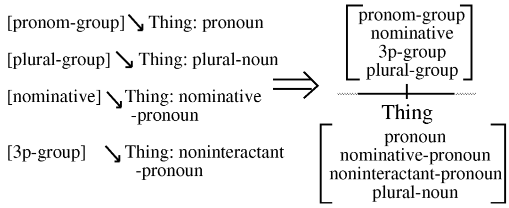

Figure 6: A Linking Partial-structure

(emphasising structure).[3]

The unification of two partial-structures produces another partial-structure, which contains the sum of the information from both partial-structures.

Note, however, that the partial-structures of the parsing grammar are not just re-expressions of a single feature and its realisations. They are more complex. They will be introduced below.

4.2 The Partial-Structures of the Parsing Grammar

For parsing, the Systemic grammar and lexicon are re-represented as sets of partial-structures. Three basic partial-structures used: *lexical* partial-structures from the lexicon, and *linking* and *ordering* partial-structures from combinations of realisation statements.

4.2.1 Lexical Partial-Structures

The first type of partial-structure is derived from lexical items. Lexical items, before precompilation, appear as in figure 4. Each lexical item can be re- represented as a set of partial-structures, one per inflectional form of the lexeme.[4] Figure 5 shows a partial-structure derived from the lexical-item of figure 4. It contains the spelling for the inflection class, and 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).

4.2.2 Linking Partial-Structures

The second type of partial-structures concern the possible fillers of each function of a unit. The realisation rules which specify the filler of each function (preselection and lexify rules) are extracted out of the grammar, and multiplied out, producing a set of partial-structures, which represents the variety of fillers the function can have. See section 5 for details of the expansion process. Figure 6 shows one such partial-structure, which was derived by combining the preselections which involve the *Thing* function 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.

Sometimes a constituent is linked to its parent through a number of functions, rather than a single function. Figure 7 shows another linking partial-structure, this time representing the combination of preselections from two conflated functions. The realisation rule conflating these two functions is also incorporated into the partial-structure.[5]

4.2.3 Ordering Partial-Structures

The third type of partial structures represents the sequencing of functions. The realisation rules

[3] The re-representation of grammatical potential in a form closer to grammatical structures is similar to the approach taken by Martin Kay in Functional Unification Grammar (Kay 1979, 1985). He was also trying to parse with Systemic Grammar at the time.

[4] Lexical partial-structures are not pre-compiled, but rather are produced during the lexical-analysis phase of parsing.

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

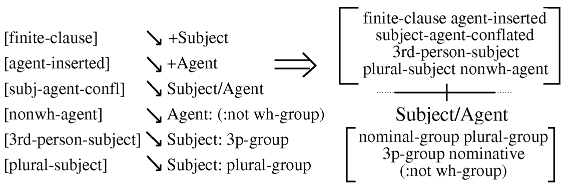

Figure 7: A Linking Partial-Structure involving Two Functions

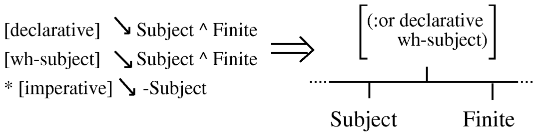

Figure 8: An Ordering Partial-Structure

which determine functional ordering are extracted from the grammar (order, partition, insert and presume), and used to produce a set of ordering partial-structures. Each ordering partial-structure represents the adjacency between two functions (sometimes function-bundles), and the condition under which that adjacency is allowed. two realisation statements which allow the order (realisations of features declarative and *wh-subject*), and restricting the possibility of the *Subject* being presumed (the '*' is read as restriction of the realisation rule).

Insertion realisations may need to be incorporated also, 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. For a similar reason, presumption realisations are also involved. Conflation statements may also be involved, since functions are not always explicitly ordered – see figure 9.

Two important kinds of ordering partial-structures involve the pseudo-functions *Front* and *End*. The first kind shows which functions can start a unit, and under what conditions. Figure 10

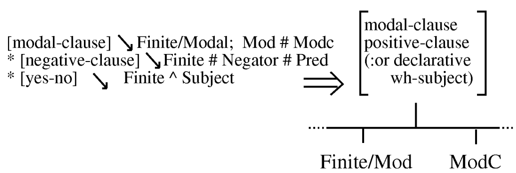

Figure 9: An Ordering Partial-Structure Using Conflation

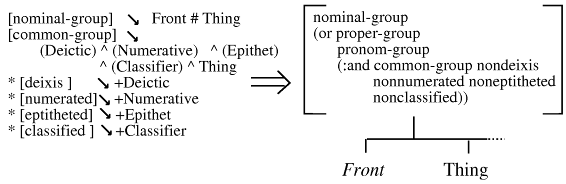

Figure 10: A Partial-Structure Ordering Thing at the Front of a Group

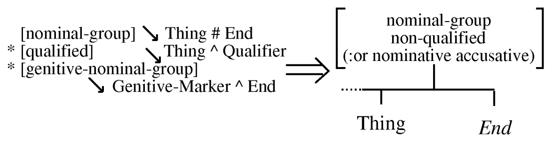

Figure 11: A Partial-Structure Ordering Thing at the End of a Group

shows a partial-structure which includes the conditions under which the *Thing* function can occur as the first element of a nominal-group.

The second kind shows the condition under which a unit can end a structure. For instance, figure 11 shows the partial-structure which allows the *Thing* function to be the final element in a nominal-group.

4.3 Summary of the Compiled Grammar

The lexico-grammatical resources are re-represented as three sets of partial-structures, lexical partial-structures, linking partial-structures and ordering partial-structures. It is these partial-structures which are used for parsing, not the uncompiled resources. The parsing-grammar represents a chunking of the realisation rules of the grammar into larger groupings of information. This results in an overall lower level of complexity in the grammar actually used for parsing. O'Donnell 1993 provides more detail on this issue.

This parsing-grammar is logically equivalent to the Systemic grammar it is derived from - no information is lost or added in the compilation process.

5 Compiling Out the Parsing Grammar

This section outlines how two of the partial-structures – linking partial-structures and ordering partial-structures – are compiled out of the

Figure 12: A partial Systemic network

```
clause:          Subject: nominative
                 Actor: human
                 Finite: finiteverb
                 Pred: lexical-verb
declarative:     Subject^Finite
yes-no:          Finite^Subject
transitive:      Object: accusative
                 Actee = [ ]
                 Pred...Object
active:          Subject/Actor
                 Object/Actee
                 Finite/Pred
passive:         Subject/Actee
                 Object/Actor
                 Pass: be-aux
                 AgentM = "by"
                 Finite/Pass
                 Pred: en-verb
                 Pass^Pred
                 AgentM^Object
intransitive:    Subject/Actor
                 Finite/Pred
single-subj:     Subject: singular
plural-subj:     Subject: plural
```

Figure 13: Realisation Rules

generation grammar.

I will show the re-compilation in terms of a staged series of re-expressions of the canonical (Systemic) form of the grammatical resources, ending in the partial-structure representation.

5.1 The Canonical Form

Figure 12 shows a system network for a simple grammar of English. It includes 11 systems, representing various grammatical distinctions, for instance, between clause and word, between transitive and intransitive clauses, or between nominative and accusative pronouns.

Types of the system network are associated with structural realisations – the structural consequence of the type. Figure 13 shows the realisations of the types in Figure 12.

This grammar deals mainly with some systems involving the Subject and Object, what types of units fill these roles, and how these roles conflate with two other roles: Actor and Actee. The grammar assumes that both roles are filled by pronouns, which are either [nominative] or [accusative], [singular] or [plural], and [human] (e.g., "I", "you", "he") or [nonhuman] (e.g., "it", "that"). Only [human] pronouns can fill the Actor role of a clause.

5.2 Logical Expression of the Grammar

For the purposes of the expansion of this grammar, we re-express it in a logical formalism. Figure 14 shows Logical Form I of this grammar, containing both the logicical organisation of the system network, and also (separately) the realisational consequences of each feature. Note that :xor indicates exclusive disjunction.

Since these two components are used distinctly, we will ignore the type-logic components in future expressions.

5.3 Compiling Linking Partial Structures

A linking partial-structure (LPS) represents a constituency relationship between a parent item and a constituent. It links the parent, represented by a feature-bundle, to the child,also represented by a feature-bundle, via a function (or function-bundle) – the constituency relation. In parsing, an LPS is used to assign a function to a completed grammatical unit.

We now need to extract out a sub-grammar of LPSs for use in parsing. The following outlines the steps of this process.

5.3.1 Extracting the relevant description

For the function-assignment process, we do not need all of the role logic description. We can select out only those rules involving preselection, lexify, and conflation. See Logical Form II in Figure 15.

```
(:and
 ;1. Type Logic Component
 (:xor (:and clause
             (:xor declarative yes-no)
             (:xor (:and transitive  (:xor active passive))
                   intransitive)
             (:xor single-subject plural-subject))
       (:and word
             (:xor (:and pronoun (:xor nominative accusative)
                                 (:xor singular plural)
                                 (:xor human nonhuman))
                   (:and verb ... ))))

 ;2. Role Logic Component
  (:and  (:implies clause (:and Subject: nominative
                                Actor: human
                                Finite: finite-verb
                                Pred: lexical-verb))
         (:implies declarative Subject^Finite)
         (:implies yes-no  Finite^Subject)
         (:implies transitive  (:and Object: accusative
                                     Actee: [ ]
                                     Pred...Object))
         (:implies active (:and Subject/Actor
                                Object/Actee
                                Finite/Pred))
         (:implies passive (:and Subject/Actee
                                 Object/Actor
                                 Pass: be-aux
                                 AgentM= "by"
                                 Pred: en-verb
                                 Finite/Pass
                                 Pass^Pred)
                                 AgentM^Object))
         (:implies intransitive  (:and Subject/Actor
                                       Fin/Pred))
         (:implies single-subject Subject: singular)
         (:implies plural-subject Subject: plural)))
```

Figure 14: Logical Form I of the Grammar

```
(:and (:implies clause
          (:and Subject: nominative
                Actor: human
                Finite: finite-verb
                Pred: lexical-verb))
      (:implies transitive
          Object: accusative)
      (:implies active
          (:and Subject/Actor
                Object/Actee
                Finite/Pred))
      (:implies passive
          (:and Subject/Actee
                Object/Actor
                Pass: be-aux
                AgentM= "by"
                Pred: en-verb
                Finite/Pass))
      (:implies intransitive
         (:and Subject/Actor
               Fin/Pred))
      (:implies single-subject
           Subject: singular)
      (:implies plural-subject
           Subject: plural)))
```

Figure 15: Logical Form II: The Function Assignment Sub-Description

5.3.2 Implications Out

We next put this description into a form more suitable for expansion to Disjoint-Normal Form (DNF). Note that implication can be re-expressed using disjunction, conjunction and negation:

```
(:implies a b)  is-equivalent-to
       (:xor (:and a b) (:not a))
```

Using this rule, we can re-express the logical form II as Logical Form III, as shown in Figure 16.

5.3.3 Expansion to DNF

Simple algorithms exist to expand Logical Form III into DNF. A small part of the result appears in Logical Form IV of the grammar, shown in Figure 17.

The order of worst-case complexity of the expansion to DNF is easily calculated – it is simply two to the power of the number of disjunctions, which is equal to the number of types which have realisation rules of type conflation, insertion, or preselection.

By opting to expand only subsets of the whole grammar, we have reduced the complexity of the description, since the size of n for this sub-description is smaller than for the whole description. However, for a real-sized grammar such as NIGEL, the size of n is still large.

```
(:and (:xor (:and clause
                  Subject: nominative
                  Actor: human
                  Finite: finite-verb
                  Pred: lexical-verb)
            (:not clause))
      (:xor (:and transitive
                  Object: accusative)
            (:not transitive))
      (:xor (:and active
                  Subject/Actor
                  Object/Actee
                  Finite/Pred))
            (:not active))
      (:xor (:and passive
                  Subject/Actee
                  Object/Actor
                  Pass: be-aux
                  AgentM: "by"
                  Pred: en-verb
                  Finite/Pass)
            (:not passive))
      (:xor (:and intransitive
                  Subject/Actor
                  Fin/Pred)
            (:not intransitive))
      (:xor (:and single-subject
                  Subject: singular)
            (:not single-subject))
      (:xor (:and plural-subject
                  Subject: plural)))
            (:not plural-subject)))
```

Figure 16: Logical Form Form III: After Implications Out

5.3.4 Re-expression in terms of Function Bundles

From the DNF-form of this description, we can extract out partial-descriptions for each function bundle. We now re-express this logical form in terms of the type constraints on each function-bundle, including both the constraint on the type of unit the function-bundle can be part of (the 'parent-constraint'), and the constraint on the filler of the function-bundle (the 'filler-constraint'). We show this as a set of triplets, of the form:

```
( <parent-types>
  <function-bundle>
  <child-types> )
```

```
1. ( (:and clause transitive
           active single-subject)
     Subject/Actor
     (:and nominative human singular)))

2. ( (:and clause transitive
           active single-subject)
```

```
(:xor
  (:and clause transitive active
        single-subject
        Subject/Actor: (:and nominative
                             human singular)
        Object/Actee: accusative
        Finite/Pred: (:and verb finite-verb
                           lexical-verb))
  (:and clause transitive
        active plural-subject
        Subject/Actor: (:and nominative
                             human plural)
        Object/Actee: accusative
        Finite/Pred: (:and verb finite-verb
                           lexical-verb))
  etc...
```

Figure 17: Logical Form Form IV: The Function Assignment Sub-Description in DNF

```
     Object/Actee
     accusative)))

3. ( (:and clause transitive
           active plural-subject)
     Subject/Actor
     (:and nominative human plural)

4. ( (:and clause transitive
           active plural-subject)
     Object/Actee
     accusative)

5.  ( (:and clause transitive
            active singular-subject
      Finite/Pred
      (:and verb finite-verb lexical-verb))
etc....
```

This representation can now be used to assign function-bundles A unit can take on a function-bundle if it can unify with the filler-constraint on the function-bundle.

For the instance we started with, "he", with types: `(:and pronoun nominative human singular)`, only one triplet would unify. We could thus posit structure for our unit:

```
[clause:transitive:active:single-subject]
    ..__________|__________...
                |
         Subject/Actor
                |
 [ pronoun:nominative:human:singular ]
                |
              "he"
```

Note that we have also gained information about the types of the parent-unit of which the unit is a constituent.

5.3.5 Reducing the number of Rules

Note that there is another simplification we can make to the triplet list. We can take all triplets with identical function bundle and child-type specification, and join them. The parent-types slot is replaced with the disjunction of the two parent-type slots. Thus, elements 2 and 4 above become a single item. This process reduces the number of rules to apply:

```
2,4. ( (:and clause transitive active)
        Object/Actee
        accusative)))
```

5.4 Compiling Order Partial Structures

Another process we use in parsing involves the prediction of what function-bundles can come next in a partially completed structure. With a systemic grammar, this process requires:

- Ordering and Partition rules: to see which function can come next.
- Conflation rules: to see which functions can conflate with the function predicted to come next.
- The type logic: to show which of these ordering, partition and conflation rules are systemically compatible.

The processing of this sub-description, and any others, is exactly the same as for function-assignment.

1. Extract from the role logic description the relevant realisation rules;
2. Replace implications with disjunction and negation;
3. Expand out the grammar;
4. Index the rules in a form useful for the processing.

6 Parsing with the Parsing Grammar

It remains to be shown how these partial-structures are put together during parsing. The general strategy for parsing in WAG is bottom-up, breadth-first, left-to-right parsing, using a chart mechanism. I will briefly demonstrate the joining together of partial-structures which occurs during parsing.

6.1 Lexification

A sentence is analysed one word at a time, from left to right. Lexical analysis involves the production of a set of candidate lexical partial-structures for the word. For instance, the lexification of a word "they" would result in a single candidate partial-structure, as was shown in figure 5.

Each of the candidates then needs to be incorporated into the parse-chart. Incorporation is performed through three steps described below: *function assignment*, *structural placement*, and *completion*.

6.2 Function Assignment

The first step to incorporate of a lexical partial-structure an analysis involves discovering what constituency functions the unit can fill. To do this, we attempt to unify the lexical partial-structure with each of the linking partial-structures in the parsing grammar, to see which are compatible.[6] Figure 18 shows the lexical partial-structure from above unifying with one of the linking partial-structures, producing a group-level partial-analysis of the pronoun.

6.3 Structural Placement

This function-assigned partial-structure has resulted from the analysis of a single word in isolation. We now need to relate it to any existing structure which resulted from analysis of words to the left of the current word (extending incomplete-edges in the chart).

Assuming that "they" was at the beginning of a sentence, there is no structure so far. We do need however to hypothesise this as the first element in a unit. For this we look for an ordering partial-structure which allows a *Front ^ Thing* ordering, and attempt to unify it with the analysis of "they" from the prior step. Figure 19 shows the result of this unification, an analysis spanning the first word of the sentence. The same operation is used to place successive constituents of each unit.

6.4 Completing a Unit

After each structural placement, we need to test if the extended analysis can be considered a completed analysis. To do this, we look for an ordering partial-structure which allows *End* to be the next element. If it unifies with the partial-structure, then the result is a *completed-structure* (a fully specified analysis of a grammatical unit). Figure 20 shows such a unification: the partial-structure produced in the last stage unifies with the linking partial-structure for *Thing ^ End*, resulting in a complete analysis of a nominal-group.

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").

6.5 Recursion of these Steps

When a 'completed-structure' is recognised, we then need to repeat the function- assignment, structural placement and completion steps for this structure. When no more function-assignments can take place, the processor advances to the next word. When the last word is handled, the parser returns any completed structure which spans the sentence as a whole.

7 Summary & Conclusions

This research has resulted in system which parses using a large Hallidayan-formalism Systemic grammar, without pre-parsing with a non-Systemic grammar, or simplifying the formalism. The WAG parser is the first parser to fit these conditions.

The major factor which makes this possible is the re-representation of the Systemic grammar in a form more suitable for bottom-up parsing. 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 parsing-grammar is automatically compiled (no modification by hand is required), and thus can be derived from the grammar used for generation.

application.

approach. This means that the grammar can be modified - or without re-programming the system.

In regards to efficiency, the WAG parser is able to handle grammars of a reasonable size, while still producing results in a reasonable time. For instance, using a version of the Nigel grammar with 500 clause- and group- (phrase) rank features, the precompilation takes approximately 2 minutes using Sun Common Lisp on a Sun Sparc II. A sentence such as "A user-password is a character string consisting of a maximum of eight alphanumeric characters." is then analysed in 15 second. With a smaller Systemic grammar developed

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

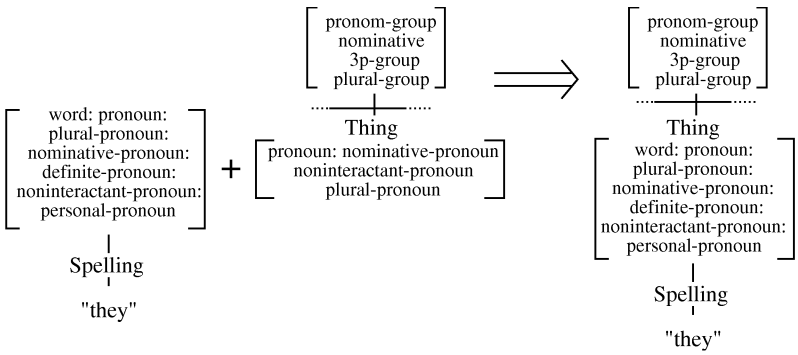

Figure 18: The Function-Assignment Operation

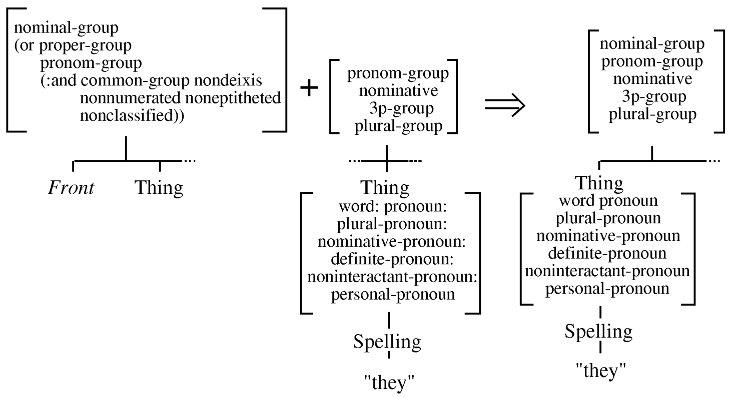

Figure 19: Placing a Partial-Structure at the Beginning of a Unit

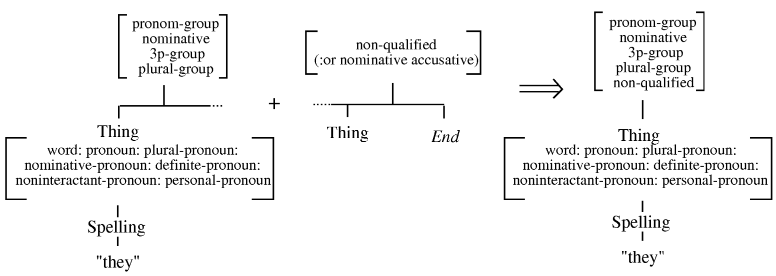

Figure 20: ‘Completing’ a Partial-Structure

by the author, the same sentence is parsed in under two seconds. These times compare favourably to other Systemic parsers, using grammars of similar coverage and complexity.

Acknowledgements

The parser discussed in this paper was partially developed in the Electronic Discourse Analyser project, funded by Fujitsu (Japan). Thanks also to Cecile Paris abd Brigitte Grote whose insightful comments have improved this paper.

Bibliography

Bateman, John 1989 "Grammar, Systemic ", in Stuart Shapiro (ed.), Encyclopedia of Artificial Intelligence, Second Edition, New York: John Wiley & Sons, pp. 583 – 592.

Bateman, John – Martin Emele – Stefan Momma, (1992) "The nondirectional representation of Systemic Functional Grammars and Semantics as Typed Feature Structures" in *Proceedings of COLING-92*, Volume III, Nantes, France, 916-920.

Benson, James – William Greaves (eds.) 1985 *Systemic Perspectives on Discourse: Selected Theoretical Papers from the 9th International Systemic Workshop*, Norwood, N.J.: Ablex.

Carpenter, Bob 1992 *The Logic of Typed Feature Structures*, Cambridge University Press, Cambridge, England.

Cummings, Michael – Al Regina (1985) "A PROLOG parser-generator for Systemic analysis of Old English Nominal Groups", in Benson and Greaves, 1985.

Emele, Martin – Rémi Zajac 1990 "Typed Unification Grammars", in *Proceedings of the 13th International Conference on Computational Linguistics, COLING-90*, Helsinki, August 1990.

Fawcett, Robin P. 1980. *Cognitive Linguistics and Social Interaction: Towards an Integrated Model of a Systemic Functional Grammar and the other Components of a Communicating Mind*, Heidelberg: Julius Groos Verlag, and Exeter: University of Exeter.

Fawcett, Robin P. – Gordon Tucker – Lin Yuen 1993 "How a Systemic-Functional Grammar Works ", in Helmut Horacek & Michael Zock (eds.) *New Concepts in Natural Language: Planning Realisation and Systems*, Pinter: London.

Halliday, M. A. K. (1985) *Introduction to Functional Grammar*, London: Edward Arnold.

Hudson, R.A. (1971) *English Complex Sentences*, North-Holland.

Kasper, Robert (1986) "Systemic Grammar and Functional Unification Grammar" In Benson, J. and Greaves, W., *Selected Papers from the 12th International Systemics Workshop*, Norwood, N.J: Ablex.

Kasper, Robert (1988a) "An Experimental Parser for Systemic Grammars", *Proceedings of the 12th Int. Conf. on Computational Linguistics*, Budapest: Association for Computational Linguistics.

Kasper, Robert (1988b) "Parsing with Systemic Grammar", Technical Document, USC/Information Sciences Institute.

Kasper, Robert (1989) "Unification and Classification: An Experiment in Information-Based Parsing", in *Proceedings of the International Workshop on Parsing Technologies*, pages 1-7, CMU, Pittsburgh.

Kasper, Robert (1990) "Performing Integrated Syntactic and Semantic Parsing Using Classification", paper presented at *Darpa Workshop on Speech and NL Processing*, Pittsburgh, June 1990.

Kay, Martin (1979) "Functional Grammar" in *Proceedings of the Fourth Annual Meeting of the Berkeley Linguistics Society*.

Kay, Martin (1985) "Parsing In Functional Unification Grammar" in Dowty D., L. Karttunen, and A. Zwicky, (Eds): *Natural Language Parsing*, Cambridge University Press, Cambridge, England.

Mann, William C. 1983 "An Overview of the Penman Text Generation System ", USC/ISI Technical Report RR-84-127.

Mann, W. C. – C. I. M. Matthiessen (1985) "Demonstration of the Nigel Text Generation Computer Program", In Benson and Greaves, 1985.

Martin, James (1992) *English Text: System and Structure*, Amsterdam: Benjamins.

Matthiessen, Christian 1985 "The systemic framework in text generation: Nigel ", in Benson and Greaves, 1985.

Matthiessen, Christian – W. C. Mann (1985) "NIGEL: a Systemic Grammar for Text Generation" in Benson and Greaves, 1985

Matthiessen, Christian – Michael O'Donnell – Licheng Zeng 1991 "Discourse Analysis and the Need for Functionally Complex Grammars in Parsing", in *Proceedings of the Second Japan-Australia Joint Symposium on Natural Language Processing*, October 2-5, 1991, Kyushu Institute of Technology, Iizuka City, Japan.

McCord, Michael (1977) Procedural Systemic Grammars in *Int. J. Man-Machine Studies*, 9, 255-286, London: Academic Press.

O'Donnell, Michael 1993 "Reducing Complexity in a Systemic Parser ", in *Proceedings of the Third International Workshop on Parsing Technologies*, Tilburg, the Netherlands, August 10-13, 1993.

O'Donnell, Michael 1994a *Sentence Analysis and Generation - A Systemic Perspective*, Ph.D. Thesis, Department of Sydney, University of Sydney, Australia.

O'Donnell, Michael 1994b *Workbench for Analysis and Generation: User Manual*, WAG System Documentation.

O'Donnell, Michael 1995 "Sentence Generation Using the Systemic WorkBench", in Proceedings of the Fifth European Workshop on Natural Language Generation, 20-22 May, Leiden, The Netherlands, pp 235-238.

O'Donoghue, Tim F. (1991a) "The Vertical Strip Parser: A lazy approach to parsing" Research Report 91.15, School of Computer Studies, University of Leeds, Leeds, UK.

O'Donoghue, Tim F. (1991b) "A Semantic Interpreter for Systemic Grammars" in *Proceedings of the ACL Workshop on Reversible Grammars*, University of California at Berkeley, June 1991.

Weerasinghe, A. Ruvan & Robin Fawcett 1993 "Probabilistic Incremental Parsing in Systemic Functional Grammar", in *Proceedings of the Third International Workshop on Parsing Technologies*, Tilburg, the Netherlands, August 10-13, 1993.

Winograd, Terry (1972) *Understanding Natural Language*, New York: Academic Press.