A Computational Framework for Lexical Description

Graeme D. Ritchie
Department of Artificial Intelligence
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN

Stephen G. Pulman
Computer Laboratory
University of Cambridge
Corn Exchange Street
Cambridge CB2 3QG

Alan W. Black
Department of Artificial Intelligence
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN

Graham J. Russell
Computer Laboratory
University of Cambridge
Corn Exchange Street
Cambridge CB2 3QG

To achieve generality, natural language parsers require dictionaries which handle lexical information in a linguistically motivated but computationally viable manner. Various rule formalisms are presented which process orthographic effects, word structure, and lexical redundancy in a manner which allows the statement of linguistic generalisations with a clear computational interpretation. A compact description of a medium-sized subset of the English lexicon can be stated using these formalisms. The proposed mechanisms have been implemented and tested, but require to be refined further if they are to be regarded as an interesting linguistic theory.

1. Methodology

As can be judged from the review in Ingria(1986), there are a wide variety of techniques and sub-systems used for handling lexical information within natural language processing systems. In many systems, particularly experimental ones, the lexicon module is fairly small and rudimentary, as the vocabulary is limited and the research is not primarily concerned with lexical issues. On the other hand, theoretical linguists have often discussed regularities that occur within the lexicon, primarily in the areas of **morphology** (word structure) and **lexical redundancy** (generalisations across lexical entries). We have designed a related set of rule-formalisms and structures which embody a linguistically-motivated theory of lexical structure, and have implemented these techniques in software which can serve as a general lexical module within a natural language parsing system. This is of theoretical interest as it presents a computer-tested set of mechanisms which fulfil, in an integrated way, some of the roles that linguists have posited for morphological and lexical rules. From a practical point of view, it defines a software module which is largely rule-driven and so can be tailored to different vocabularies, and perhaps even to various languages. Although it has been designed with syntactic parsing as the main intended application, most of the linguistic mechanisms and descriptions are independent of their use within a parser.

It is important to bear in mind the distinction between a linguistic *mechanism* and a linguistic *description* which uses that mechanism. We have developed not only a related set of formalisms, all with a clear computational interpretation, we have devised a description of a large subset of English morpho-syntactic phenomena using these formalisms. Although the adequacy of the mechanism and of the description are mutually interdependent, it is important to maintain this distinction when appraising the work reported here, particularly when considering its possible extension to other vocabularies or other languages. Another important issue when considering a computationally feasible system is the question of how to interpret a rule-notation procedurally. Linguistic formalisms tend to be discussed as declarative statements of regularities within the language, and it is not always clear what is the appropriate interpretation when the rules have to be used for processing data. For example, the Feature Co-occurrence Restrictions of Gazdar et al.(1985) define arbitrary logical constraints to which feature-sets (categories) must conform. A computational implementation has (at least) two ways in which these statements could be interpreted - as recipes for filling in extra features, or as filters for rejecting ill-formed categories (cf. Stanley(1967)). It is not at once apparent whether a linguist writing FCR statements would accept both of these as equally “natural” interpretations. Whatever algorithmic interpretation is chosen for a rule notation, it should be computationally tractable and fairly obvious to the reader. This has led us, particularly in the area of lexical redundancy rules, to opt for notations which have a very obvious and explicitly defined procedural interpretation.

A further methodological question which arises when giving a computational interpretation to declarative statements of lexical regularities is that of whether a rule notation is best regarded as a notational device which allows the linguist to write more succinct entries, but which is not used directly in the computation of the association between a character string and a lexical entry. In terms of the implementation, this is the question of whether a rule-system is an aid to the entry of data by the linguist (and can be used for some form of pre-processing) or is a mechanism which is used in a more general or efficient look-up procedure.

In designing linguistic rule-formalisms, there is traditionally a trade-off between the power of the mechanism and the substance of the linguistic claims or theories embodied in the notation. We have generally opted for fairly powerful techniques, in the interests of achieving generality and flexibility, for two reasons. Firstly, we were not sure initially what facilities would be needed for an adequate description of lexical phenomena in English, and so had to allow scope for experimentation. It would be possible, in the light of regularities within our description, to devise a more restricted set of rule formalisms if this was desired. Secondly, we wished to design and implement a set of tools which could be used by computational linguists of a variety of theoretical persuasions and with varying needs, and hence we felt it would be too restrictive to tailor the rule systems to the minimum that our description of English demanded.

We shall start by giving an informal description of the overall system, then we shall outline some of the rule systems in more detail, and finally our description of English word-structure will be summarised.

2. Overview

The system can be thought of as a **lexicon** and various sets of **rules.** The lexicon contains entries for separate **morphemes,** each entry containing four fields — a **citation form** (which is a canonical spelling for the morpheme), a **phonological form**, a **syntactic category** (an unordered set of features as in current unification-style grammars such as Gazdar et al.(1985), Kay(1985)), and a **semantic structure**, about which we shall say nothing here. (In the implementation, a fifth field is included for miscellaneous purposes, but it has no linguistic significance. However, all examples shown in this paper are taken from our implemented description and hence will contain this fifth field, with a value of NIL).

There are two classes of rules. **Lexical rules** (of which there are three types) express relationships between entries, or between fields within entries, and have a procedural interpretation which maps a set of basic entries into a possibly larger set of entries with more specified categories. **Morphological rules** (of which there are two types) express relationships between characters and morphemes within a word, and have a procedural interpretation which allows a string of characters to be decomposed into a structural analysis of a word. There is also a very simple mechanism for defining **default values** for syntactic features, which does not properly fall into either of these classes of rules.

The lexical rules are of the following types:

Completion Rules. These state implications between parts of lexical entries — typically, that the presence of certain syntactic feature-values (or combinations of them) make the presence of other particular values obligatory. Their procedural interpretation is that the predicted information should be added to the entries which match the antecedent of the implication.

Multiplication Rules. These state implications concerning the existence of lexical entries. Their procedural interpretation is to construct new lexical entries that are predictable in form from existing ones.

Consistency Checks. These state implications between combinations of information within entries (again, typically syntactic features). Their procedural interpretation is to reject any lexical entries which are internally inconsistent.

Notice that generalisations which might, in a wholly declarative framework, be stated using a single type of rule (e.g. the Feature Co-occurrence Restrictions of Gazdar et al.(1985)) are here dealt with by two separate rule-types, differing in their procedural interpretation. It was found to be convenient for practical use to make the distinction between asking the system to force entries to have a particular form by adding information and stipulating that erroneously specified entries be rejected (however, see also the discussion on Lexical Rules in Section 8 below).

The morphological rules are of two sorts:

Spelling Rules. These state relationships between **surface forms** of words (i.e. ordinary orthography including inflections) and **lexical forms** (i.e. canonical forms in lexical entries, typically with stems and affixes stored separately). The rules are based on the formalism in Koskenniemi(1985), which is in turn a high-level notation related to Koskenniemi(1983a,b). Their procedural interpretation is that they can be used to segment character strings into individual morphemes, taking account of orthographic effects which may occur at morpheme boundaries (or elsewhere).

Word Grammar Rules. These rules describe the possible internal structures of words, using a feature-grammar notation like that of Gazdar et al.(1985), with certain feature-passing conventions to supplement the use of variables and unification. They have the obvious procedural interpretation, in that a fairly conventional context-free parser can use these rules to analyse a sequence of morphemes into a structural tree.

The remaining notational device is the Feature Default definition, which allows the statement of a single default value for a feature. That is, these are not as sophisticated as the Feature Specification Defaults of Gazdar et al.(1985), since they do not allow logical combinations of features and values (but see further comments in the section about Lexical Rules in Section 8 below).

In addition to the above rule mechanisms, the implemented version contains various other notational conveniences to support the definitions of rules or to reduce the work that the lexicon-writer must carry out. These include the ability to define **aliases** for clusters of syntactic features, and to define names for **feature-classes.**

As observed above, the need to use the lexicon within a natural language parsing system demands that there be a clearly defined computational process underlying the definitions of the rule-types. This also applies to the integrated functioning of the various mechanisms. The operation of the dictionary system can be viewed as having two stages— **compilation** and **use.** The former phase is a pre-processing in which lexical entries supplied in a linguistically appropriate form are manipulated by the lexical rules to produce a modified lexical entry set, containing additional (predictable) information, and possibly more entries. Of course, if the linguist has chosen to state no lexical rules whatsoever, this process is fairly simple and the set of entries is unaltered (in the implementation, the compilation process also inserts the entries in a tree-like index (cf. Thorne, Bratley and Dewar(1968)), and hence even without lexical rules there is a need for compilation so that subsequent access works correctly). The phase of dictionary use is essentially the process of looking up arbitrary character strings in the compiled dictionary, with the morphological rules being used to produce a list of all possible analyses of the string into words. Feature Defaults are inserted at appropriate junctures during the look-up phase.

3. Spelling Rules

These rules (called "morphographemic rules") are concerned with undoing spelling or phonological changes to recover the form of a word which corresponds to some morpheme entry in the lexicon. For example, *moved* can be viewed as *move+ed*, but with the deletion of the extra *e*; *provability* can be viewed as *prove+able+ity*, with adjustments occurring at both the internal boundary points.

The formalism used within this system is based on the work of Koskenniemi (1983a, 1983b, Karttunen 1983). In earlier versions of this formalism, the linguist had to specify the spelling rules in a low-level notation similar to that for finite state automata, but Koskenniemi (1985) outlined a more perspicuous high level notation, and we have adopted a variant of that, with a compilation technique inspired by the work of Bear(1986).

The first point to understand about the rule formalism is that the rules describe relationships between the **surface form**, that is the actual word as it appears in a sentence, and the **lexical form**, as it appears in the citation forms of the lexical entries. For example, *moved* is the surface form while *move* and *+ed* are the lexical forms. What is required is a rule that allows the deletion of an *e* from the lexical form. Note that the rule should refer to the context where the *e* can be deleted and not just allow arbitrary deletions of *e*s in the lexical form as then the surface form *reed* would match *red* in the lexicon.

The format for the Spelling Rules includes initial declarations and definitions of the associated entities (character sets, etc.) needed to support the actual rule-definitions, as follows. The **surface alphabet** is the set of acceptable symbols in a string being looked up, the **lexical alphabet** is the set of acceptable symbols within citation forms in lexical entries, and named subsets of these alphabets can be declared.

The spelling rules are specified as a pair (lexical symbol : surface symbol), and the context in which that pair is acceptable. A lexical symbol can be one of three types: a lexical character from the declared lexical alphabet; a lexical set, declared over a range of lexical characters; or the symbol 0 (zero) which represents the null symbol. Similarly there are three possibilities for the surface symbol.

Before a more detailed description of the formalism is given a simple example may help to explain the notation. The following example describes the phenomenon of adding an *e* when pluralising some nouns (also making some verbs into their third person singular form). e.g *boys* as *boy+s* while *boxes* as *box+s*. This phenomena is known as "epenthesis":

```
Epenthesis

    +:e   <=>   { < { s:s  c:c  } h:h  >  s:s  x:x  z:z  } — s:s
```

The left and right contexts are basically regular expressions, with angle brackets indicating sequences of items, curly braces indicating disjunctive choices, and ordinary parentheses enclosing optional items. This rule assumes that the morpheme *+s* (see below for comments on the + character) is in the lexicon and represents the plural morpheme. (Let us exclude for the time being its use as the third person singular morpheme). Roughly speaking, the epenthesis rule states that *e* can be added at a morpheme boundary when and only when the boundary has *sh*, *ch*, *s*, *x*, or *z* or on the left side and *s* on the right. The "—" can be thought of as marking the position of the symbol pair *+:e*.

Within our formalism there are no built-in conventions concerning morpheme boundaries. However, it is often necessary to state a rule which stipulates the presence of a morpheme boundary in the context. One way to do this is to add a marker (some special character) to the lexical form of the morphemes involved. Rules would then be able to refer indirectly to morpheme boundaries by means of this special character in the context statement. This means we have morphemes of the lexical form *+ed*, *move*, *+ing*, *+ation*, etc.

Another example in our English description is the "E-Deletion" rule:

```
E-Deletion

   e:0   <=>   =:C2 --- < +:0 V:= >
         or    < C:C V:V > --- < +:0 e:e >
         or    { g:g c:c } --- < +:0 { e:e i:i } >
         or    l:0 --- +:0
         or    c:c --- < +:0 a:0 t:t b:b >
```

where V, C and C2 represent particular subsets of the alphabets, and the = sign matches any symbol (roughly speaking). Although alternatives can be specified within a left or right context using the disjunctive construct, we also need the ability to allow alternatives for full contexts. If separate rules were given for each alternative left and right context there would be the undesirable effect of each one blocking the other, since rules are treated as conjoined; that is, *all* rules must match for a sequence of symbol pairs to be acceptable. Hence, to achieve a disjunctive choice for contexts there is the "or" connective as used in "E-Deletion" above. (This is not fully general as a rule pair can only have one operator type). Each context in the above rule is for particular cases: the first allows words like *moved* as *move+ed*; the second allows *argued* as *argue+ed*; the third allows *encouraging* as *encourage+ing* but also copes with *courageous* as *courage+ous*; the fourth context deals with e-deletion in words like *readability* as *read+able+ity*; and the last context allows e-deletion in *reduction* as *reduce+ation*.

The three possible rule operators are: <=, => or <=>, which represent forms of implication, in the following manner.

Context Restriction:

```
a:b   =>   LC --- RC
```

This means the lexical character *a* can match the surface character *b* only when it is in the context of LC and RC, and hence *a:b* cannot appear in any other context.

Surface Coercion:

```
a:b   <=   LC --- RC
```

This means that in the context LC and RC a lexical *a* can only be matched with a surface *b* and nothing else; for example *a:c* is disallowed in this context.

Combined Rule:

```
a:b   <=>   LC --- RC
```

This is equivalent to the combination of the context restriction and surface coercion rules. It means *a* matches *b* only in the context LC and RC, and *a:b* is the only pair possible in that context.

An addition to the formalism, which is formally not needed, is the introduction of a "where" clause. This saves the user writing separate rules for similar phenomena. A good example can be seen in the rule for consonant doubling (gemination):

```
Gemination

   +:X   <=>   < C:C  V:V  =:X  >  ---  V:V
         where X in { b  d  f  g  l  m  n  p  r  s  t }
```

The rule is effectively duplicated with the variable *X* bound to each member of the set in turn. If a "where" clause were not used and *X* declared as a set ranging over { b d f g l m n p r s t }, the value found for *X* in the rule pair *+:X* would not necessary be the same value for *X* in the left context. There would be no point in giving sets this interpretation as we do not want the *V:V* in the left context necessarily to be the same *V:V* in the right.

The interpretation of pairs containing sets depends on the notion of **feasible pairs**. A pair consisting of a lexical symbol and a surface symbol is a feasible pair if either it is a **concrete pair** (see below) or consists of two identical symbols from the intersection of the lexical and surface alphabets. Concrete pairs are those pairs appearing in the rules (assuming any "where" clauses are expanded into explicit enumeration) which are made up of characters in the alphabets or null symbol only (i.e. containing no sets). Pairs containing sets, such as *V:V* where the lexical set V is { a e i o u y } and the surface set V is { a e i o u y } are interpreted as all feasible pairs that match. If *y:i* is a feasible pair then it will match *V:V*. Rules will typically be written only for pairs *a:b* where *a* and *b* are different characters. It is built into the formalism that unless otherwise restricted, all feasible pairs are accepted in any context.

In addition to the definition above for feasible pairs there is the facility to declare explicitly that certain pairs are feasible. This may be useful where some pair in a rule contains a set and the user wishes it to stand for some concrete pair that does not actually exist in any of the currently specified rules. For example the pair *+:=* may be used, where = can be thought of as a set containing the whole surface alphabet. The user may intend this pair to stand for, among others, *+:l*, although *+:l* does not actually appear in any of the rules. In this case, *+:l* should be declared as a **default pair**.

Any number of spelling rules can be specified (our English description has 15 — see appendix 2 for an annotated list). These rules are applied in parallel to the matching of the surface form and the lexical forms. For a match to succeed, all rules must find it acceptable. All members of the set of feasible pairs not on the left-hand side of some rule (i.e. *a:a, b:b, c:c*, etc.) are accepted in any context.

There are some problems with this form of rule. When a rule pair *a:b* from some rule A with the operator <=> or => also appears within a context of some other rule B, the user must take care to ensure that the context where *a:b* appears within rule B is catered for in rule A. An example will help to illustrate this point. Consider the following two rules:

```
E-Deletion
      e:0   <=>   =:C2 --- < +:0 V:= >
        or    < C:C V:V > --- < +:0 e:e >
        or    { g:g c:c } --- < +:0 { e:e i:i } >
        or    l:0 --- +:0

A-deletion
      a:0   <=>   < c:c e:0 +:0 > --- t:t
```

The *e:0* in the left context of the “A-deletion” rule is in a context that is not catered for within the “E-Deletion” rule. This means that “A-deletion” will always fail. What is required is the addition of another context to the “E-Deletion” rule:

```
        or    c:c --- < +:0 a:0 t:t >   ;; A-deletion
```

This rule-clashing is a significant factor that must be taken into consideration when specifying spelling rules (see Black et al.(1987) for further discussion). We have not yet investigated formal criteria for detecting clashes within a rule-set, and it may in principle be undecidable (or at least highly intractable)

Another decision the linguist has to make is when to treat a given alternation as morphographemic, and when to treat it by writing distinct morpheme entries. For example, it seems ridiculous to go as far as writing the following rule:

```
o:e   <=> g:w --- < +:0 e:n d:t >
```

which will match *went* to *go+ed*. This rule is in fact insufficient as it introduces the pairs *w:g*, *e:n* and *d:t* into the feasible pairs set and thus allows *wear* to match *gear* etc. If this rule were to be included then three more would be needed to cope with these three extra pairs. But rules that match surface forms to such different lexical forms are not recommended. It seems wise to have *went* as a morpheme entry with the necessary past tense marking. *Went* is a clear example but some others are not so clear. Should *written* match *write+en*? The question of when a change is to be taken as a different morpheme or just as a spelling change is a question of the overall adequacy and elegance of the description — there are no firm guidelines.

4. Word Grammar

The morphological rules concerned with word-structure can be viewed as a “Word Grammar”, characterising derivational and inflectional morphology in abstraction from the details of the actual character strings involved. These rules describe what constitutes an allowable sequence of morphemes, stating which concatenations are valid, and the syntactic class of the overall word formed by several morphemes. For example, *happy+ness* is a valid noun, but *arrive+ness* is not a valid word.

The word grammar is based on the concept of features and values. Any constituent (morpheme, word, word-part, etc.) can be represented by a set of features and values, called a **category.** Our model of English morphology is based heavily on the GPSG model of syntactic features (cf. Gazdar et al. (1985), chap. 2) (although it could be used simply as a very general feature-grammar by anyone who did not wish to adopt the more esoteric aspects of GPSG theory). For example the category of a plural noun can be represented as:

```
((N +) (V -) (PLU +) (BAR 0))
```

All features used in the word grammar (and lexical entries) must be declared to the analyser system. There are two types of features, **atomic-valued** and **category-valued**. Atomic-valued features must be declared with an enumerated set of atomic values. Category-valued features can take any valid category as their value. These are declared using the keyword *category*, e.g.

```
Feature  N    {+,-}
Feature  BAR  {-1,0,1,2}
Feature  AGR  category
```

Although our sample English description uses particular feature names, there is no need for the linguist to copy such conventions. There is only one restriction on the features declared. If a feature of the name STEM is declared, it must be a category-valued feature. This feature is used by the WSister Convention (see below) and should not be used in any other way.

The word grammar is a feature unification grammar with rules of the form:

```
mother  ->  daughter1, daughter2, ..., daughterN
```

where *mother, daughter1, daughter2,* etc. are categories made up of features. Rules may have one or more daughters. In addition to simple categories the grammar may also contain variables and aliases (see below).

Aliases are a short-hand for writing categories (and parts of categories). They allow an atomic name to be associated with a category, and hence then be used to represent that category in a rule. For example the aliases *Noun* and *Verb* might be declared as:

```
Alias   Noun = ((BAR 0) (N +) (V -))
Alias   Verb = ((BAR 0) (N -) (V +))
```

There are two types of variables allowed within the categories in the grammar; "rule-category variables" and "feature value variables". Rule-category variables range over specific categories, and are a short-hand for writing similar grammar rules. They are declared with a range of possible values that must be stated as a list of aliases. Rule-category variables can be used to capture generalisations in rules. For example, in French both nouns and adjectives can take a plural morpheme *s* (which can be represented by the category ((PLU +))). This phenomenon could be described using the following alias statements and rules:

```
Alias    Adj  =  ((BAR 0) (N +) (V +))
Alias    Noun =  ((BAR 0) (N +) (V -))

(AdjPlural
        ( Adj (PLU +) ) -> ( Adj (PLU -) ), ( (PLU +) ) )

(NounPlural
       ( Noun (PLU +) ) -> ( Noun (PLU -) ), ( (PLU +) ) )
```

Alternatively, the two rules can be written as one by declaring a category variable:

```
Alias    Adj  =  ((BAR 0) (N +) (V +))
Alias    Noun =  ((BAR 0) (N +) (V -))

Variable   C  =  {Adj,Noun}

(Plural
       ( C (PLU +) ) -> ( C (PLU -) ), ( (PLU +) ) )
```

Rule-category variables are "compiled out" during grammar compilation, and are thus actually used to collapse a number of rules.

Feature value variables, on the other hand, can best be thought of as "holes" that are filled in during parsing (although theoretically they have equivalent semantics to rule-category variables, if we overlook the distinction between abbreviations for finite sets and for infinite sets). There are two types of feature value variables — **atomic-valued** and **category-valued** (category-valued variables are *not* the same as rule-category variables). The distinction is analogous to that between the atomic-valued features and category-valued features described above. Atomic-valued variables are declared with an enumerated set of values, while category-valued variables are declared with the keyword *category*:

```
Variable   ALPHA = {+,-}
Variable   ?AGR  = category
```

Feature value variables are not compiled out at grammar compile time but are instantiated during parsing. The ranges of feature value variables can be used to restrict the scope of rules. They can also be used to "copy" values of features up (and down) the parse tree. For example, a compound noun can be said to inherit its plural feature marking from the rightmost daughter. Using feature value variables we can write a rule that ensures that the compound noun will have the same PLU marking as its rightmost daughter:

```
Variable    ?X   =   {+,-}
Alias       N    =   ((BAR 0) (N +) (V -))

(NounCompound
      ( N (PLU ?X) ) ->
          ( N (PLU -) ),   ;; ensure basic noun
          ( N (PLU ?X) )
)
```

Note that although atomic-valued variables can be thought of as a short-hand for a number of rules, one for each value in the range of the variable, category-valued variables cannot. This is because there is potentially an infinite number of categories that could be the value of a category-valued feature.

There are no typographical conventions built-in for specifying variables; the rule-writer, however, may wish to adopt some convention such as starting all variables with underscore or question mark. This does make rules easier to read but is in no way mandatory.

In addition to the use of variables for "passing" features around during parsing there are some built-in feature-passing conventions (see below for more details).

Before a description of what constitutes a valid analysis can be given two definitions are required.

Extension A feature-value (either atomic or a category) is an extension of any variable of an appropriate type. An atomic feature-value is an extension of itself. Category A is an extension of category B iff for any feature f in category B, there is a value of f in A which is an extension of the value of f in category B.

Unification

The unification of two categories is the smallest category that is an extension of both of them if such category exists. It is possible that no such category exists, and in that case unification is undefined.

Intuitively, extension and unification can be though of as the set relation superset and the set operation union, respectively, with the extra refinement of allowing at most one entry for each feature within a category. The creation of the unification of two (or more) categories is referred to as "unifying" the categories.

The morphological analyser uses the rules in the Word Grammar to find all possible structures for a given word. Each structure is a tree in which each node is

EITHER:

the keyword ENTRY and a lexical entry

OR: a local tree of the form *N* -> *c1 c2 ... cn*, where *N* is a category and *ci* is a constituent. This tree must match the following constraints

a. there must be a rule in the word grammar of the form *A* -> *d1 d2 ... dn*, where category *N* is an extension of *A* and *ci* is an extension of *di* for each *i* from *1* to *n*.

b. the local tree must be valid with respect to the feature-passing conventions.

The analyser returns all constituents that span the given string and have a category that is an extension of the category which the linguist has defined to be the distinguished category. That is, in the same way that a traditional context-free grammar has a single distinguished symbol which is used to define complete derivations, our morphological model has a distinguished *category.*

5. Feature-Passing Conventions

Feature-passing conventions can be thought of as a way of extracting various patterns which occur in the word-grammar rules and stating them separately. The effect of this is to diminish the amount of explicit information that needs to be stated in the word-grammar rules, reducing both the size of the word-grammar (the number of rules) and the complexity of the individual rules. These regularities can be expressed as **feature-passing conventions** which can be thought of as rules for passing information up the analysis tree (from terminal morphemes to the final word), or down the analysis tree (from word to constituent morphemes). The way of stating these conventions is based on the mechanisms employed by Generalised Phrase Structure Grammar at the level of the sentence (Gazdar et al. (1985)), but the morphological generalisations embodied in them are essentially those of Selkirk(1982).

There are three conventions built into the system at present. Notice that the definitions of the feature-passing conventions themselves are not under the control of the lexicon-writer, although the features that are affected by the conventions may be modified. The conventions act on certain specific features or feature-classes, so the linguist can make use of these conventions by defining certain features to lie within these named classes. The system will then automatically apply the conventions to these features.

All three feature conventions act on what is called within GPSG terminology a “local tree” — a set of one mother node and its immediate daughters. The conventions were originally designed for binary branching rules (introducing exactly two daughters), but they apply to all rules. They are written in terms of “mother”, “left daughter” (i.e the leftmost daughter in a local tree) and “right daughter” (the rightmost daughter). In unary rules, those with just one category on the right-hand side, the left and right daughters are the same category.

The Word-Head Convention

The WHead feature-values in the mother should be the same as the WHead feature-values of the right daughter.

In the word parser, this is achieved, roughly speaking, by unifying the WHead features of the right daughter and those of the mother when the daughter is attached.

From a linguistic point of view, the WHead features are typically those that will be relevant to sentence-level syntax, and hence those that will be of particular use to a sentence-parser which uses the dictionary. This convention is a straightforward analogue of the simplest case of the Head Feature Convention in (Gazdar et al. (1985)). Its effect is to enforce identity of the relevant feature values between mother and the head daughter. Note that in the current system there is no formal definition of "head" to which the lexicon-writer has access (despite the name given to this convention), since the right daughter always acts in this head-like fashion within our treatment of English morphology. Other analyses may deviate from this pattern, of course; different views of "head" may be implemented using variables and unification.

Assuming the set of WHead features includes N, V, PLU, and VFORM, the Word-Head Convention would allow the following trees:

```
((N +) (V -) (PLU +))
        ()
        ((BAR -1) (N +) (V -) (PLU +))
```

and

```
((N -) (V +) (VFORM ING))
        ((N -) (V +))
        ((BAR -1) (N -) (V +) (VFORM ING))
```

but not (after all unification has occurred) trees of the form:

```
((N +) (V +) (PLU +))
        ()
        ((BAR -1) (N +) (V -) (PLU +))
```

and

```
((N -) (V +))
        ()
        ((BAR -1) (N -) (V +) (VFORM EN))
```

since one of the trees has a clash in the V value for mother and right daughter, and the other lacks a VFORM marking on the mother to match that on the right daughter.

The Word-Daughter Convention

(a) If any WDaughter features exist on the right daughter then the WDaughter features on the mother should be the same as the WDaughter features on the right daughter.

(b) If no WDaughter features exist on the right daughter then the WDaughter features on the mother should be the same as the WDaughter features on the left daughter.

Again, this is ensured by carrying out unification of the appropriate feature markings during parsing. This convention is designed to capture the fact that the subcategorization class of a word (in English) is not affected by inflectional affixation, although it may be affected by derivation.

Assuming the feature SUBCAT to be the only WDaughter feature, this convention allows trees such as:

```
((SUBCAT NP))
        ((V +) (N -))
        ((SUBCAT NP))

((SUBCAT NP))
        ((SUBCAT NP))
        ((VFORM ING))
```

but not

```
((SUBCAT NP))
        ((V +) (N -))
        ((SUBCAT VP))

((SUBCAT NP))
        ((SUBCAT VP))
        ((VFORM ING))
```

In the first example the right daughter is specified for a SUBCAT value, and the mother has the same specification; in the second example, the right daughter has no specification for SUBCAT and so the second clause of the WDaughter convention applies. The third example is illegal because the values of SUBCAT on the right daughter and mother differ, and the fourth is illegal because, under clause (b) of the convention, the left daughter and mother WDaughter features must be identical when there are no WDaughter features in the right daughter.

The Word-Sister Convention

When one daughter (either left or right) has the feature STEM, the category of the other daughter must be an extension (superset) of the category value of STEM.

This third convention enables affixes to be subcategorized for the type of stem to which they attach. Notice that this convention is not defined in terms of any feature-classes, but is defined using just one "built-in" feature (STEM). Hence, the way that the lexicon-writer makes use of this convention is not by declaring the extent of feature-classes (as for the other two conventions), but by adding STEM specifications to the features in morphemes in the lexicon, thereby indicating the combination possibilities for each affix. The following examples follow the convention

```
()
        ((N -) (V +))
        ((STEM ((N -) (V +))))

()
        ((V +) (N -) (INFL +))
        ((STEM ((N -) (V +) (INFL +))))
```

6. Feature Defaults

Feature Defaults are similar in concept to the Feature Specification Defaults of Gazdar et al. (1985). They are statements which define values for particular features in circumstances where no value has been entered by other mechanisms (i.e. the original morpheme entries, the action of the lexical rules, or the feature-passing conventions). That is, they state what the value of a feature should be if there is no information to indicate any other value for it. The defaults are applied to all new constituents (words or parts of words) built during morphological analysis. (In terms of the active chart implementation of the parsing mechanism, the default checking is done whenever a complete (inactive) edge is entered into the chart).

At present, only very simple defaults are available, compared to the various kinds of defaults proposed (for sentence-level grammar) by Gazdar et al. (1985). All the linguist can do is define the default value for a given feature (either a category-valued feature or an atomic-valued one). For example, the statement

```
Defaults BAR 0, AGR Inf
```

declares default values for two features (BAR and AGR), where "Inf" could be an alias for some category. However, Completion Rules have arbitrary descriptive power, and can be used to achieve complex insertion of default feature values, providing that the default-insertion can be performed adequately on individual morphemes (not categories formed by combining morphemes), since Completion Rules have their effect prior to morphological analysis. (See discussion in Section 8 below)

7. Lexical Rules

All three types of rule (Completion Rules, Multiplication Rules, and Consistency Checks) have the same basic form:

```
< pre-condition > < operator > < action >
```

Although the < operator > and < action > are different in each type of rule, the syntax of the <

pre-condition > is the same. Pre-conditions are specified as conjunctions of (possibly negated) patterns, describing lexical entries. Variables are denoted by atoms starting with an underscore e.g. *_fred*, *_fix* etc. and are bound during matching so that they can be used later in a match or in a rule action. There is a special variable consisting of only an underscore ("_"), which never gets bound but can be used to match anything (cf. Prolog). All other variables have a consistent interpretation throughout a rule. Matching is done from left to right (which is significant in the matching of syntactic fields). The entry being matched does not have to have the features in the same order as the pattern.

The following examples of syntactic category matching illustrate some of the above points:

```
((FIX _fix) ~(BAR _) _rest)
```

with tilde indicating negation, matches

```
((FIX SUF) (N +) (V -))
```

with *_fix* bound to SUF and *_rest* bound to ((N +) (V -))

```
((FIX _fix) ~(BAR _) _rest)
```

does not match

```
((FIX SUF) (BAR -1) (N +) (V -))
```

since ~(BAR _) fails to match any feature value for BAR

```
((N -) (V +) _rest)
```

matches

```
((V +) (PLU -) (N -) (INFL +))
```

with *_rest* bound to ((PLU -) (INFL +))

```
((N +) (V -) _rest)
```

matches

```
((V -) (N +))
```

with *_rest* bound to an empty list of features. The pattern *((N +) _junk (V -))* does not match any syntactic category because the variable *_junk* will match all remaining features in the category being checked.

When negation is used no bindings that are made within a negative pattern are passed on through the match (again cf. Prolog), although bindings can be passed into negations.

The above examples concern only the syntactic field, but pre-conditions match against entire entries. For example:

```
~(be _ _ _ _) and (_ _ ((N -) (V +) _rest) _ _)
```

would match all entries that do not have the citation form *be* and are marked with the features (N -) and (V +), and

```
(_ _ ((N +) (V -) ~(PLU _) _) _ _)
```

would match any entry with the features (N +) and (V -) but not the feature PLU (with any value).

Completion Rules

Completion Rules are designed to be used to add values to the entries that are specified by the linguist, and are applied in order to the entries (after aliases have been expanded). Accordingly, the order of the Completion Rules is significant. A Completion Rule is of the form

```
< pre-condition > => < entry skeleton >
```

If a pre-condition matches an entry the entry is *replaced* with the newly constructed one described by the entry skeleton. A entry skeleton is of the same general form as a lexical entry, but various parts of it may contain the ampersand symbol (&), to mean "the same as in the original entry", or variables which have appeared in the pre-condition (and hence would have been bound in the matching process).

For example the rules:

```
(_ _ ((FIX _fix) ~(BAR _) _rest) _ _) =>
        (& & ((FIX _fix) (BAR -1) _rest) & &)

(_ _ (~(BAR _) _rest) _ _) =>
        (& & ((BAR 0) _rest) & &)

(_ _ ((STEM (~(INFL _) _stem) _rest) _ _) =>
        (& & ((STEM ((INFL +) _stem)) _rest) & &)
```

have the action of adding (BAR -1) to entries containing the feature FIX, adding (BAR 0) to all entries that do not have a BAR marking and lastly adding (INFL +) to all values of STEM that do not already have a marking for INFL. Note that the ordering of the first two rules is significant. If the first two rules were in the reverse order, the FIX rule would not apply to any entries, as all entries would by that time have had (BAR 0) added.

Multiplication Rules

Multiplication Rules construct additional entries (as opposed to the replacement of entries performed by Completion Rules). These are typically used to generate similar entries with slightly varying feature markings; for example, in English these rules can be used to generate the first person, second person and plural of verbs from the base form (as an alternative to designing the morphological rules to handle the verb paradigm). The syntax of these rules is very similar to that of the completion rules:

```
< pre-condition > =>> ( < list of entry skeletons > )
```

The syntax of the entry skeletons is exactly the same as above. The ordering of the rules is not significant as newly created entries are *not* re-tested against the Multiplication Rules. This is to avoid possible infinite application of rules.

A Multiplication Rule to generate the first and second person singular and plural of a base verb could be:

```
(_ _ ((V +) (N -) (BAR 0) (VFORM BSE) (INFL +) _rest) _ _) =>>
    (
      (& & ((V +) (N -) (BAR 0) (PN PER1) (INFL -) _rest) & &)
      (& & ((V +) (N -) (BAR 0) (PN PER2) (INFL -) _rest) & &)
      (& & ((V +) (N -) (BAR 0) (PN PLUR) (INFL -) _rest) & &)
    )
```

Note that the entry being tested is not replaced but remains in the lexicon (assuming the Consistency Checks are passed—see below). So, given the entry

```
(like lAIk ((BAR 0) (V +) (VFORM BSE) (N -)
            (INFL +) (SUBCAT VP2a)) LIKE NIL)
```

four entries would exist after the application of the Multiplication Rule, having the form:

```
(like lAIk ((BAR 0) (V +) (VFORM BSE)
            (N -) (INFL +) (SUBCAT VP2a)) LIKE NIL)
(like lAIk ((V +) (N -) (PN PER1)
            (BAR 0) (INFL -) (SUBCAT VP2a)) LIKE NIL)
(like lAIk ((V +) (N -) (PN PER2)
            (BAR 0) (INFL -) (SUBCAT VP2a)) LIKE NIL)
(like lAIk ((V +) (N -) (PN PLUR)
            (BAR 0) (INFL -) (SUBCAT VP2a)) LIKE NIL)
```

Consistency Checks

Consistency Checks are applied to every entry (including newly created ones) after the above two sets of rules have applied. Any entry that does not pass these tests is not included in the lexicon. The only formal requirement on lexical entries is that entries are quintuples and that the syntactic field is a set of feature pairs with values as declared. These Consistency Checks allow the linguist to check linguistic dependencies within entries; for example, there is no built-in prohibition of a category that contains a V marking but no N marking, but the linguist may wish to specify that such a category is invalid. Consistency Checks are statements of the form:

< pre-condition > **demands** < post-condition >

The < post-condition > has the same syntax as the pre-conditions. The interpretation is:

If an entry matches the pre-condition it must also match the post-condition.

For instance, if all entries that are marked for V must also be marked for N and vice versa then this condition can be written as the following two Consistency Checks:

```
(_ _ ((V _) _) _ _) demands (_ _ ((N _) _) _ _)
(_ _ ((N _) _) _ _) demands (_ _ ((V _) _) _ _)
```

The rules are applied in the following order: Completion Rules (in order of specification), Multiplication Rules (order is not significant), and finally the Consistency Checks are applied to each entry created by the previous rule applications.

8. Description of English

The mechanisms outlined in the preceding sections could be used to construct almost any description of English lexical facts. Here we sketch one such description, which we have developed using the mechanisms described here. It is worth observing in passing that the features used in the description can be broadly grouped into the following (overlapping) classes:

Purely sentential. These features are included as part of the grammatical description of sentence structure, and do not have any particular import within morphological rules. For example, the feature SUBCAT is used to indicate the subcategorisation of verbs, etc. but does not affect spelling or word-structure. Features which do not affect morphology may still be manipulated by the morphological rules, since the feature-passing conventions will cause various features to be passed from morphemes to words. Thus whole words will inherit features from their component morphemes, without the rules mentioning the features explicitly.

Sentential with morphological effects. The features V and N (for classifying nouns, verbs, adjectives and prepositions, in the GPSG style), although obviously motivated by the syntactic form of sentences, also affect various affixing processes.

Purely morphological. Certain features have been postulated in our description solely to distinguish classes of morpheme that have different behaviour morphologically. For example, the feature FIX (with possible values PRE and SUF) indicates an affix, and the feature INFL (possible values + and -) indicates whether a word or morpheme is capable of further inflection.

Notice that this is not a formal distinction, and does not correspond to any sub-divisions in our mechanisms - it is merely an observation about our description of English that certain features are not motivated by morphological considerations. In a sense, they could still be said to "affect" the morphological processing, since if a feature is mentioned in the STEM value of a morpheme, it will restrict possible morpheme combinations. The sentential features (i.e. the first two classes above) have been devised in collaboration with the writers of a medium-sized grammar of English (Briscoe et al. 1986), but we shall not discuss here the justifications for the decisions regarding sentential grammar. Appendix 1 outlines the usage of the more morphologically or lexically significant features.

The Word Grammar describing inflectional and derivational morphology is not large; the complete set is given below. Each rule is preceded by a mnemonic name, and VAL is a variable ranging over + and -.

```
( PREFIXING
   ((BAR 0)) ->
          ((FIX PRE)), ((BAR 0)) )

( SUFFIXING
   ((BAR 0) (N +)) ->
          ((BAR 0)), ((N +) (FIX SUF)) )

( V-SUFFIXING
   ((N -) (V +) (AUX VAL) (BAR 0)) ->
          ((AUX VAL) (BAR 0)), ((FIX SUF) (N -) (V +)) )

( NON-V-SUFFIXING
   ((N -) (V +) (AUX VAL) (BAR 0)) ->
          ((N +) (BAR 0)), ((N -) (V +) (FIX SUF) (AUX VAL)) )
```

Since the feature-marking (BAR 0) indicates a item which constitutes a whole word, the PREFIXING rule can be summarised as “Any word can be made up of a prefix followed by another valid word”. (Properties of the prefix and stem determine the full features of the word by the Feature-Passing Conventions—see below).

The SUFFIXING rule can be phrased “Any noun or adjective can be made up of a noun or adjective stem followed by a suffix”. Notice that this rule does not stipulate that the stem must be of the same major category (noun or adjective) as the overall word, and hence it covers derivational morphology (where the category is altered by affixation) as well as noun inflections. The restriction to nouns or adjectives (i.e. entities marked as (N +)) is necessary as verbs require the slightly more detailed rules V-SUFFIXING and NON-V-SUFFIXING (and prepositions — ((N -) (V -)) — do not take affixes at all).

The V-SUFFIXING rule states that “a verb can be made up of a verbal stem of the same auxiliary marking followed by a verbal suffix”. This is to cover general verb inflection, for both auxiliaries (AUX +) and main verbs (AUX -).

The NON-V-SUFFIXING rule is to cover those cases of derivational morphology where a noun or adjective (N +) stem becomes a verb through suffixation - “any noun or adjective which forms a whole word can form a whole word verb by the addition of a verbal suffix".

In all these cases, the rules may seem to be rather sketchy and lacking in feature specifications. For example, the PREFIXING rule does not stipulate much about the relationship between stem and composite word, and therefore seems to omit the generalisation that prefixation does not alter the grammatical features of the word (in particular, the major category is the same). However, these highly economical grammar rules are made possible by the assumption that the various feature-passing conventions (and feature defaults) will ensure that features are correct. Hence it is crucial that the word grammar be assessed in conjunction with the feature-passing conventions defined in Section 5 above, and the following definitions of feature classes:

```
WHead Features:
        N V INFL PAST AFORM VFORM BARE-ADJ ADV AGR PLU POSS FIN
WDaughter Features:
        SUBCAT
```

The features in the WHead list will be forced to have the same values on the right-daughter and the mother; hence these feature-values when specified on a suffix will percolate on to the main word, and will also remain on the main word when a prefix is added. Similarly, the WDaughter feature will be inherited from the appropriate part of the word. There are also two feature-defaults:

```
        BAR 0
        LAT +
```

These ensure that any morpheme, word, or part of a word which does not have a value for BAR will be marked as a potential whole word, and that any item not marked for being "Latinate" will be assumed to be so.

There are two main types of lexical rules - Completion Rules and Multiplication Rules. The third type (Consistency Checks) are desirable for disciplined lexicon-writing, but they do not insert any features or entries, and will not be discussed here.

Although it is not obvious from the outline of the formalism given here, Completion Rules can be used in several ways to control the content of lexical entries. This is not to say that the notation has several interpretations, but rather that the lexicon writer can choose to employ the facilities of these rules in different styles, in much the same way that a programmer can use a programming language in different manners. We will term the three main usages *overwriting, obligatory insertion,* and *optional defaulting.* The overwriting use is fairly straightforward - in this, a rule is used to alter the values of one or more specified features. For example, a rule like the following would change all adjectives to nouns:

```
(_ _ ((V +) (N +) _rest) _ _)  =>
        (& & ((V -) (N +) _rest) & &)
```

This is achieved by mentioning the relevant features explicitly in the pre-condition (pattern), and supplying a new form (right-hand side) which has an explicitly stated revised form of them; all other features are carried over unchanged by the variable *_rest*. Leaving aside the rather absurd content of the above example, we have made no use of this effect in our description of English, as there seemed no point in putting in entries which were to be systematically altered by rule (notice that this is different from having the entries filtered out by some rule such as a Consistency Check).

The notion of obligatory insertion is more subtle. This involves writing rules which will insert a feature value if it is not there already, but will result in the lexical entry being *discarded* if that feature is already specified in it. For example, consider a rule which adds (INFL -) to all entries marked with an AFORM value:

```
(_ _ ((AFORM _af) _rest) _ _) =>
        (& & ((INFL -) (AFORM _af) _rest) & &)
```

The Completion Rule mechanism is not defined to check for the presence of features it is attempting to add, so this rule will attempt to add (INFL -) even if the entry already has a value for INFL (whether + or -). If there is no previous marking for INFL, this insertion will be successful, and the rule will have effectively added a default marking. If there is some previous marking, the insertion will fail, since the mechanism is not defined to overwrite entries, and the lexical entry will be discarded.

Optional defaulting is a slightly more circumspect way of inserting default values. Consider the following rule, which also inserts (INFL -) as a default marking on entries specified for AFORM.

```
(_ _ ((AFORM _af) ~(INFL _) _rest) _ _) =>
   (& & ((INFL -) (AFORM _af) _rest) & &)
```

In this version, the pre-condition has an explicit check for the absence of an INFL marking (since the tilde sign indicates negation of the immediately following condition). This rule will apply only to entries which are unspecified for INFL, and will have no effect (i.e. leave in the lexicon unaltered) entries for which this pre-condition is not true.

All the Completion Rules in our description of English are written to act in a "default" manner; in fact, to use the terminology introduced immediately above, they are written in the "optional default" style, in that each rule checks for the absence of a feature-value before inserting the value. The full set of Completion Rules is given in Appendix 3.

There are two Multiplication Rules. The first is to account for the fact that any noun or adjective which is subcategorised for a complement (e.g. *critic* takes a prepositional phrase with *of*) can also occur with no complement:

```
(_ _ ((N +) (SUBCAT _) _rest) _ _) and ~(_ _ ((SUBCAT NULL) _) _ _)
  =>>
(
        (& & ((N +) (SUBCAT NULL) _rest) & &)
)
```

The second clause of the pre-condition is necessary simply because the pattern-matching mechanism does not permit the use of negated feature values (e.g. (SUBCAT ~NULL)) within a pattern. The second Multiplication Rule expands the present tense verb paradigm for all verbs apart from *be*, by adding three further lexical entries per verb:

```
 (be _ _ _ _) and
(_ _ (~(VFORM _) ~(FIN _) ~(AGR _) (V +) (N -) (INFL +) _rest) _ _)
   =>>
(
  (& & ((V +) (N -) (FIN +) (INFL -) (PAST -) (AGR PLUR) _rest) & &)
  (& & ((V +) (N -) (FIN +) (INFL -) (PAST -) (AGR PER1) _rest) & &)
  (& & ((V +) (N -) (FIN +) (INFL -) (PAST -) (AGR PER2) _rest) & &)
)
```

It might seem that this regularity would be more naturally handled in the morphological analyser, rather than as lexical redundancy. However, that would necessitate the introduction of a morpheme whose entire surface form was null (with suitable syntactic features). The complications this would introduce into the morphographemic segmentation and word-grammar parsing are regarding as wholly unacceptable computationally, and so the Multiplication Rules have been used to capture this generalisation. (This is a very obvious example of the methodological issue mentioned in our opening section, concerning the need for a viable procedural interpretation of the whole set of mechanisms).

There are 15 spelling rules in our description. Appendix 2 contains an annotated list of them. In addition to the actual rules the spelling rule mechanism requires the definition of the lexical and surface alphabets. The Surface Alphabet contains all normal alphabetic letters, space, hyphen and apostrophe (for simplicity, we shall ignore the issue of upper and lower case here). The Lexical Alphabet contains exactly the same symbols together with the plus sign (+) which we use to mark morpheme boundaries. The null symbol (0) is a part of the formalism, and hence is not regarded as part of either alphabet (but may occur in rules anywhere that a normal alphabet symbol might occur). In addition to standard identity pairs made from the intersection of the lexical and surface alphabets, three default pairs are declared, so that these pairs are valid in any context during matching. a morpheme boundary symbol may be deleted on the surface. hyphens in a citation form (e.g. *data-base*) may be absent from the surface string (e.g. *database*).

": - where a lexical form has a space (e.g. *data base*), the surface strings may optionally contain a hyphen instead (*data-base*).

9. An Example

The above word grammar, and various other parts of the rules and definitions, can be illustrated with a simple example - the analysis of the word *applications*. The Spelling Rule interpreter will segment this (using the "C-Insertion" rule, and the Default Pair definition that pairs morpheme boundaries with null) into three morphemes - *apply*, *+ation*, and *+s*. In the original lexical entries, these morphemes are listed thus, with *+s* having two entries:

```
(apply apply ((V +) (N -) (SUBCAT NP_PPTO)) APPLY NIL)

(+ation +ation
        ((FIX SUF) (V -) (N +) (INFL +)
          (STEM ((V +) (INFL +) (N -))))
        ATION
        NIL)

(+s +s
    ((FIX SUF) (V +) (N -) (FIN +) (PAST -) (AGR SING3)
       (STEM ((V +) (N -) (INFL +))))
    S
    NIL)

(+s +s
    ((FIX SUF) (V -) (N +) (PLU +)
       (STEM ((N +) (V -) (INFL +))))
    S
    NIL)
```

However, various Completion Rules will have acted upon these basic entries at the pre-compilation stage of the lexicon, resulting in the following more detailed entries for the three morphemes we are interested in here (ignoring the other entry for *+s*):

```
(apply apply ((INFL +) (V +) (N -) (BAR 0) (AT +) (LAT +)
       (SUBCAT NP_PPTO) (AUX -)) APPLY NIL)

(+ation +ation
       ((FIX SUF) (V -) (N +) (BAR -1) (INFL +) (PLU -)
        (AT +) (LAT +) (STEM ((V +) (INFL +) (N -))))
        ATION
        NIL)

(+s +s
    ((FIX SUF) (V -) (N +) (BAR -1) (PLU +) (INFL -)
     (AT +) (LAT +) (STEM ((N +) (V -) (INFL +))))
    S
    NIL)
```

Seven rules achieve this - one adds the marking (BAR -1) to entries marked as affixes (i.e. specified for FIX), another adds (BAR 0) to all entries which are specified for V and N but lack a BAR value, the third adds (INFL -) to all morphemes marked with (PLU +), the fourth adds (INFL +)

to all (BAR 0) entries which lack an INFL value, the fifth adds (AUX -) to all verbs (but not verbal affixes), the sixth adds (LAT +) to all entry with a V marking, and the seventh adds (AT +) to all entries with a (LAT +) marking. Notice that the ordering of the Completion Rules in the description is crucial, for example the third of these rules "feeds" the fourth.

The SUFFIXING rule in the Word Grammar combines the first two morphemes into a subtree with the lexical entries for *apply* and *+ation* as daughter nodes. The left-hand side of this rule assigns the following syntactic category to the mother node:

```
((BAR 0) (N +))
```

Further feature markings are then computed, using the Feature Defaults and the Feature-Passing Conventions, giving

```
((BAR 0) (N +) (V -) (INFL +) (PLU -) (LAT +) (SUBCAT NP_PPTO))
```

The markings (V -), (INFL +) and (PLU -) result from the WHead Convention, since they must be equal to the markings on the right daughter (*+ation*). The marking (LAT +) follows directly from the Feature Default, since these are added to all constituents found by the word-parser, not just to individual morphemes. The (SUBCAT NP_PPTO) is a result of the WDaughter Convention, because there is no SUBCAT feature on the right daughter it must be the same as that on the left. The SUFFIXING rule operates again, to combine this word (*application*) with the plural morpheme *+s*, to form a tree whose daughter categories are:

```
Left:   ((BAR 0) (N +) (V -) (INFL +) (LAT +) (PLU -) (SUBCAT NP_PPTO))
Right:  ((FIX SUF) (V -) (N +) (BAR -1) (PLU +) (INFL -)
         (AT +) (LAT +) (STEM ((N +) (V -) (INFL +)))
```

Notice that this combination will accord with the WSister Convention, since the category of the left-daughter is an extension of the value of STEM on the right-daughter - ((N +) (V -) (INFL +)). The category of the mother node includes, from the SUFFIXING rule, the following markings:

```
((BAR 0) (N +))
```

Again, further feature markings are then computed, giving:

```
((BAR 0) (N +) (V -) (INFL -) (LAT +) (PLU +) (SUBCAT NP_PPTO))
```

The markings (V -),(INFL -) and (PLU +) result from the WHead Convention, since V, INFL and PLU are WHead features and so must have the same values as on the right daughter (*+s*). The marking (LAT +) follows directly from the Feature Default, and the SUBCAT feature is a result of the WDaughter Convention. The overall structure for the word can then be viewed as a tree in which each node is annotated with either a syntactic category and a rule-name or the keyword ENTRY and a morphemic lexical entry:

```
((SUBCAT NP_PPTO) (V -) (LAT +) (PLU +) (INFL -) (N +) (BAR 0))
SUFFIXING
   ((SUBCAT NP_PPTO) (LAT +) (PLU -) (V -) (INFL +) (N +) (BAR 0))
      SUFFIXING
         (ENTRY (apply apply ((AUX -) (BAR 0) (V +) (N -) (INFL +) (AT +)
                             (LAT +) (SUBCAT NP_PPTO)) APPLY NIL))
         (ENTRY (+ation +ation ((INFL +) (BAR -1) (N +) (V -) (PLU -)
                         (AT +) (LAT +) (FIX SUF) (STEM ((LAT +) (INFL +)
                         (N -) (V +) (BAR 0)))) ATION NIL))
      (ENTRY (+s +s ((INFL -) (PLU +) (AT +) (LAT +) (V -) (FIX SUF)
               (BAR -1) (STEM ((INFL +) (N +) (V -))) (N +)) S NIL))
```

10. Conclusions

We have presented an integrated set of formalisms for describing various aspects of the lexicon in a computationally tractable manner, which have been used to create a non-trivial description of English lexical phenomena. All these facilities have been implemented (in Franz Lisp on a Sun 2/120) and are being used as part of collaboration between Edinburgh, Cambridge and Lancaster universities to develop a set of software tools for natural language processing, under the Alvey Programme. It should be borne in mind that all the rule-formalisms are highly experimental, and if they are to form a useful linguistic theory (as opposed a practical software package) a great deal of refinement is required. Not only are some of them perhaps too powerful (e.g. Completion Rules have arbitrary computational power), some of them may be too weak descriptively (e.g. it is not clear if the morphological mechanisms are adequate for all languages). The description is also still under development; the rules given here reflect the state of the system in summer 1986.

Appendix 1 - Lexically Significant Syntactic Features

The following are brief explanations of the features which are involved in the Completion Rules in Appendix 3, the Word Grammar in Section 8, or which are particularly pertinent to the Feature Passing Conventions. Each feature name is followed by a list of its allowable values.

AT (- +)

Stems to which the suffixes *+ation* and *+ative* may attach are marked as (AT +), while those taking the corresponding forms *+ion* and *+ive* are (AT -). This specification is referred to in the STEM feature of the suffixes in question, resulting in *action* and *presentation*, but not e.g. *presention*.

LAT (- +)

The feature LAT distinguishes those stems traditionally analysed as latinate from others. Certain affixes may only attach to latinate stems; *+an* is one such, giving *magician*, but not *artan*.

BARE-ADJ (- +)

This is a feature allowing us to refer to two disjoint category sets. Certain suffixes (e.g. *+ly*) may attach either to the base form of regular, inflectable, adjectives (as in *easily*), or to non-inflectable adjectives (as in *dangerously*). They may not, however, attach to inflected forms (*easiestly*); BARE-ADJ distinguishes e.g. *easy* and *dangerous* ((BARE-ADJ +)) from *easiest*, whereas INFL (see below) does not.

FIX (PRE SUF)

All affixes bear a specification for FIX; prefixes have the value PRE, and suffixes have the value SUF. The rules of the word grammar refer to the specifications, so that prefixes always precede their stems and suffixes always follow.

AGR (SING3 SING IT PER1 PLUR PER2 DEF SING1 N1SING N1PLUR THAT_S)

The feature AGR is responsible for enforcing the necessary correspondence between categories in sentence structure. This is most common in the case of subject and verb; *is* is specified as (AGR SING3), and *am* as (AGR SING1).

POSS (+ -)

Distinguishes possessive items from others. *His, whose*, and the possessive *'s* are specified as (POSS +).

INFL (- +)

INFL distinguishes those stems which may bear an additional suffix (e.g. *walk* would be specified with (INFL +)) from those which cannot (e.g. *walking*).

STEM category

The STEM feature controls the attachment of affixes to stems. The value of STEM in an affix category must (by the WSister feature passing convention) be included in the category of any stem that affix attaches to. In this way, *+ing*, for example, can be restricted to the base form of verbs.

BAR (-1 0 1 2)

The sentence grammar employs a three-level system of categories, various phrases being specified as (BAR 1) or (BAR 2), and preterminals as (BAR 0). In our analysis of word-structure, we extend this concept below the level of the complete word; stems are specified as (BAR 0), and affixes as (BAR -1).

V (- +)
N (- +)

The major categories (nouns, verbs, adjectives, and prepositions) are classified by means of the features V and N. Verbs and adjectives, and their phrasal counterparts, are specified as

(V +), while nouns and prepositions are specified as (V -). Nouns and adjectives, and their phrasal counterparts, are specified as (N +); verbs and prepositions are (N -).

QUA (- +)

Determiners (articles like *the, a* and adjectives like *all, three*) are specified as (QUA +). Other adjectives are (QUA -).

ADV (- +)

Adverbs derived from adjectives (*quickly, easily*) are analysed as adjectives bearing the specification (ADV +).

AUX (- +)

Verbs are specified as (AUX +) if they are auxiliary verbs, and as (AUX -) otherwise.

FIN (- +)

Verbs are specified as (FIN +) if they are finite (tensed), and as (FIN -) otherwise.

PAST (- +)

Finite verbs are specified as (PAST +) if they are in the past tense, and as (PAST -) otherwise.

NEG (- +)

NEG distinguishes negative words from others; *aren't* etc. are specified as (NEG +).

PLU (- +)

PLU distinguishes plural nouns from others; *men* and *cats* both bear the specification (PLU +), and *man* and *cat* (PLU -).

DEF (- +)

Determiners are specified for DEF; *the* is (DEF +) and *a* is (DEF -).

AFORM (ER EST NONE)

AFORM encodes information concerning adjective morphology. Comparatives and superlatives are specified as (AFORM ER) and (AFORM EST) respectively, and non-inflectable adjectives are (AFORM NONE).

NFORM (IT THERE NORM)

NFORM encodes the type of a noun phrase. The "dummy subjects" *it* and *there* are specified as (NFORM IT) and (NFORM THERE) respectively, while other NPs are (NFORM NORM). Certain verbs can then be associated with one type of NP by means of their AGR feature.

VFORM (BSE EN ING TO)

VFORM encodes verb morphology. "Bare infinitives" are (VFORM BSE), passives and past participles are (VFORM EN), gerunds and present participles are (VFORM ING), and the infinitive *to* is (VFORM TO).

SUBCAT (NP N1 AP INF PRED PP PPFROM PPOF PPAT PPWITH PPTO
PPON PPIN NP_PPOF NP_PPTO NP_INF ING THAT_S FOR_S
BARE_S BASE_S FIN_S BASE_VP IT PPTO_THAT_S NP_Q
NP_LOC NP_PPFOR PPABOUT NP_PPFOR NP_PPFROM NP_PPWITH
PPOF_PPWITH OBJ_GAP NP_PPIN A1 PPFOR NP_NP NP_THAT_S
LOC NP_BASE_VP NP_PPBY PPAGAINST PPBY NP_ING Q_S
NP_AS_PRED PPTO_THAT_S NP_PPINTO DEF NP_IN NP_Q_S
N1PLUR S NPOF PLUR NPTO N1SING NP_PPFROM PPOVER SING3
NP_PRED NP_OFF NP_ON NP_UP ANY OBJ SUBJ_- NULL)

SUBCAT encodes the subcategorization class of a word. *Elapse*, like other purely intransitive verbs is (SUBCAT NULL), and *devour*, like other transitives, is (SUBCAT NP). Many minor category items are their own subcategory; *and* has the specification (SUBCAT AND), etc.

Appendix 2 - Spelling Rules

The first five rules are based on those described in Karttunen and Wittenburg (1983). There are 15 rules in total:

```
Epenthesis
        +:e   <=>   { < {s:s c:c} h:h > S:S y:i } --- s:s
```

This allows the insertion of an *e* at a morpheme boundary, before an *s* and preceded by either *sh*, *ch*, one of *s*, *x* or *z*, or a *y/i* pair as in *fly/flies*.

```
Gemination
        +:X   <=>   < C:C V:V =:X > --- V:V
                    where X in { b d f g l m n p r s t }
```

This deals with doubling of consonant in words like *bigger*, *travelling*, etc.

```
Y-to-I
        y:i   <=>   { C:C c:t } --- < +:= NA:NA >
              or    =:= --- < +:c a:a { t:t l:l b:b} >   ;; c-insertion
```

This rule deals with changing a lexical y to an i as in *applies* (Note that this requires both this rule and the Epenthesis rule above as there is an *e* insertion too). The *c:t* change is to cope with words like *democratic* (from *democracy+ic*). The second clause of the Y-to-I rule deals with words like *application*.

```
E-Deletion
        e:0   <=>   =:C2 --- < +:0 V:= >
              or    < C:C V:V > --- < +:0 e:e >
              or    { g:g c:c } --- < +:0 { e:e i:i } >
              or    l:0 --- +:0
              or    c:c --- < +:0 a:0 t:t >   ;; A-deletion
```

This deals with e-deletion in words like *moved* (*move+ed*). The second clause deals with words ending in two vowels like *agreed*. The third clause deals with hard and soft *g*'s and *c*'s so that an *e* must be deleted in *faced* but not in *advantageous* (if it were the *g* would become hard).

```
I-to-Y
        i:y   <=>   =:= --- < e:0 +:0 i:i >
```

This handles words like *dying* and *lying*.

```
C-insertion
        +:c   <=>   y:i --- < a:a { t:t l:l b:b } >
```

This is required for *application*. Note that this rule requires a clause in the Y-to-I rule so that the the *y:i* in this left context is allowed by the Y-to-I rule.

```
K-insertion
        +:k   <=>   < V:V c:c > --- { < i:i n:n > e:e y:y }
```

This caters for words ending in *c* that keep a hard *c* when a suffix is added e.g. *panicky* and *picnicking* while it does not require *k* insertion for words like *criticise*.

```
A-deletion
        a:0   <=>   < c:c e:0 +:0 > --- t:t
```

A-deletion deals with examples like *reduction* (*reduce+ation*).

```
E-to-I
        e:i   <=>   c:= --- < +:0 { a:a o:o } NB:NB >
```

This covers words like *pronounciation*, *gracious*, *spacious* etc. (The NB ("not b") set is to stop the *+able* suffix from being affected.

```
I-insertion
        +:i   <=>   < C:C Vp:Vp NLR:NLR > --- < a:a { n:n l:l } >
```

Examples of this are *baronial*, *academician*, *civilian*, *dictatorial* (This could be extended to cope with *adverbial*, etc).

```
C-to-T
        c:t   <=>   n:n --- < =:i +:0 a:a NB:NB >
```

This should cope with *evidential*, *influential*.

```
Y-deletion
        y:0   <=>   g:g --- < +:0 i:i =:NG >
```

Examples are *allergic* (from *allergy+ic*).

```
L-deletion
        l:0   <=>   b:b --- < e:0 +:0 l:l >
```

This rule deals with matching *bly* to *ble+ly* as in *probably*.

```
L-to-I
        l:i   <=>   b:b --- e:l

E-to-L
        e:l   <=>   l:i --- < +:0 i:i { t:t z:z s:s } >
```

These above two rules deal with matching *ability* to *able+ity* as in *probability*, and similarly *abilize* to *able+ize* as in *stabilize*. These rules are an interesting example of how to deal with a change that happens over several characters. They deal with matching *bil0* to *ble+* where the *il* matches *le*.

Appendix 3 - Completion Rules

Each rule is preceded by a brief comment outlining its effect. ,LP Add (BAR -1) as default to all entries with FIX specifications.

```
(_ _ ((FIX _fix) ~(BAR _) _rest) _ _) =>
    (& & ((FIX _fix) (BAR -1) _rest) & &)
```

Add (LAT +) as default to all entries with V specifications.

```
(_ _ ((V _v) ~(LAT _) _rest) _ _) =>
    (& & ((V _v) (LAT +) _rest) & &)
```

Add (AT +) as default to all entries with (LAT +) specifications.

```
(_ _ ((LAT +) ~(AT _) _rest) _ _) =>
    (& & ((AT +) (LAT +) _rest) & &)
```

Add (INFL -) as default to all entries with AFORM specifications.

```
(_ _ ((AFORM _af) ~(INFL _) _rest) _ _) =>
    (& & ((INFL -) (AFORM _af) _rest) & &)
```

Add (INFL -) as default to all entries with VFORM specifications.

```
(_ _ ((VFORM _vf) ~(INFL _) _rest) _ _) =>
    (& & ((INFL -) (VFORM _vf) _rest) & &)
```

Add (INFL -) as default to all entries with FIN specifications.

```
(_ _ ((FIN _fin) ~(INFL _) _rest) _ _) =>
    (& & ((INFL -) (FIN _fin) _rest) & &)
```

Add (INFL -) as default to all entries with (PLU +) specifications.

```
(_ _ ((PLU +) ~(INFL _) _rest) _ _) =>
    (& & ((INFL -) (PLU +) _rest) & &)
```

Add (BAR 0) as default to all entries with V and N specifications.

```
(_ _ ((N _n) (V _v) ~(BAR _) _rest) _ _) =>
    (& & ((BAR 0) (N _n) (V _v) _rest) & &)
```

Add (PLU -) as default to all noun entries.

```
(_ _ ((V -) (N +) ~(PLU _) _rest) _ _) =>
    (& & ((N +) (V -) (PLU -) _rest) & &)
```

Add (INFL +) as default to all entries with (BAR -1) specifications.

```
(_ _ ((BAR -1) ~(INFL _) _rest) _ _) =>
    (& & ((INFL +) (BAR -1) _rest) & &)
```

Add (INFL +) as default to all entries with (BAR 0) specifications.

```
(_ _ ((BAR 0) ~(INFL _) _rest) _ _) =>
    (& & ((INFL +) (BAR 0) _rest) & &)
```

Add (QUA -) as default to all adjective entries.

```
(_ _ ((BAR 0) (V +) (N +) ~(QUA _) _rest) _ _) =>
    (& & ((BAR 0) (V +) (N +) (QUA -) _rest) & &)
```

Add (DEF -) as default to all entries with (QUA +) specifications.

```
(_ _ ((QUA +) ~(DEF _) _rest) _ _) =>
    (& & ((DEF -) (QUA +) _rest) & &)
```

Add (AUX -) as default to all verb entries.

```
(_ _ ((BAR 0) (V + ) (N -) ~(AUX _) _rest) _ _) =>
    (& & ((AUX -) (BAR 0) (V + ) (N -) _rest) & &)
```

Add (BARE-ADJ +) as default to all entries with (AFORM NONE) specifications.

```
(_ _ ((AFORM NONE) ~(BARE-ADJ _) _rest) _ _) =>
    (& & ((BARE-ADJ +) (AFORM NONE) _rest) & &)
```

Add (BARE-ADJ +) as default to all adjective entries with (INFL +) specifications.

```
(_ _ ((V +) (N +) (INFL +) ~(BARE-ADJ _) _rest) _ _) =>
    (& & ((BARE-ADJ +) (V +) (N +) (INFL +) _rest) & &)
```

Acknowledgements

This work was supported by SERC/Alvey grant GR/C/79114.

References

Bear, John. 1986 A Morphological Recognizer with Syntactic and Phonological Rules. In: *Proceedings of the 11th International Conference on Computational Linguistics.* Bonn, West Germany: 272-276.

Black, Alan W.; Ritchie, Graeme D.; Pulman, Stephen G.; and Russell, Graham J. 1987 Formalisms for Morphographemic Description. In: *Proceedings of 3rd Conference of the European Chapter of the ACL.* Copenhagen, Denmark.

Briscoe, Edward.J.; Craig, I.; and Grover, Claire. 1986 The Use of the LOB Corpus in the Development of a Phrase Structure Grammar of English. In: *Proceedings of 6th ICAME*, Amsterdam. (To be published eds. Meijs, W., and van der Steen, G.J.).

Gazdar, Gerald; Klein Ewan; Pullum, Geoffrey K. and Sag, Ivan A. 1985 *Generalized Phrase Structure Grammar* Blackwell, Oxford, England.

Ingria, Robert. 1986 Lexical Information for Parsing Systems: Points of Convergence and Divergence. In: *Proceedings of Workshop on Automating the Lexicon*, Grosseto, Italy.

Karttunen, Lauri and Wittenburg, Kent. 1983 A Two-level Morphological Analysis of English. *Texas Linguistics Forum* 22, Department of Linguistics, University of Texas, Austin, Texas: 217-228.

Karttunen, Lauri. 1983 KIMMO: A general morphological processor. *Texas Linguistics Forum* 22, Department of Linguistics, University of Texas, Austin, Texas: 165-186.

Kay, Martin. 1985 Parsing in Functional Unification Grammar. In: Dowty, D.; Karttunen, L. and Zwicky, A. *Natural Language Parsing.* Cambridge University Press, London: 251-278.

Koskenniemi, Kimmo. 1983a Two-level model for morphological analysis. In: *Proceedings of the Eighth International Joint Conference on Artificial Intelligence.* Karlsruhe, West Germany: 683-685.

Koskenniemi, Kimmo. 1983b Two-level Morphology: a general computational model for word-form recognition and production. Publication No.11, University of Helsinki, Helsinki, Finland.

Koskenniemi, Kimmo. 1985 Compilation of Automata from Two-Level Rules. Talk given at Workshop on Finite-State Morphology, CSLI, Stanford, CA July 1985.

Selkirk, Elisabeth O. 1982 *The Syntax of Words.* MIT Press, Cambridge, Mass.

Stanley, Richard. 1967 Redundancy Rules in Phonology. *Language* 43, No.2: 393-436.

Thorne, James P.; Bratley, Paul and Dewar, Hamish. 1968 The Syntactic Analysis of English by Machine. In: Michie, Donald, Ed., *Machine Intelligence 3* Edinburgh University Press, Edinburgh, Scotland: 281-309 .