A Lexicon Handler for The ProGram Grammar Development System.

W.R. Keller

Cognitive Studies Research Paper

Serial no: CSRP 40

June 1984

ABSTRACT

A lexicon handler for the ProGram Grammar Development System is described. Redundancy in the specification of word entries is considering the lexicon to consist of permanent lexicon or dictionary, and a cal component. The user may record and irregular word forms in the perman in a succinct and uniform way. The m component is further subdivided to formation rules and feature convent stated. These rules and conventions are capturing generalisations about the processes of inflectional morphology.

A version of the lexicon handler has been implemented in the programming language Prolog. The program performs two distinct tasks: the 'normalisation' of user-defined lexicon components, and the provision of an interface between the lexicon and the ProGram parser. Both aspects of program operation are explained by reference to examples.

Table of Contents

A Lexicon Handler for The ProGram Grammar Development System.

W.R. Keller

Cognitive Studies Research Paper

Serial no: CSRP 40

June 1984

1. Introduction.

The goal of the work described in this paper has been the design of a new lexicon handler for the Sussex ProGram Grammar Development System (Evans and Gazdar.1984) More specifically, the intention has been to produce a design which avoids much of the redundancy inherent in the existing system, and which allows the user to capture morphological regularities in a succinct and natural manner. In this context 'redundancy' can be taken to mean the exhaustive specification of all word forms, including those derived through regular inflectional processes*. For example, in the current lexicon separate entries must be made for each form of the regular verb, love, despite the fact that:

(*). For present purposes, inflectional morphology is taken to be the addition of suffixes to words which, (i) Does not change the category of the word (though usually modifying so-called minor features such as tense, number, case, etc.); (ii) Operates in a grossly regular (productive) and therefore predictable manner with respect to some subclass of words of a particular category.

a. In the present tense -five out of the six forms are identical (i.e. 'I love', 'you love' etc).

b. The third person singular present tense form exhibits regular inflectional morphology (i.e. the addition of the inflectional suffix 's' to give loves)

c. The past tense is also formed by a regular process, namely the addition of the inflectional suffix 'ed' (i•e. loved).

d- The past and present participle forms (loved and loving respectively) are also formed in a regular fashion.

Clearly such redundancy cannot be tolerated in a lexicon of any practical size <a thousand or more words say).

A step by step approach has been taken to the design and implementation of the new system. The advantage of such an approach is that at each stage of the design process, a working system is available for evaluation and testing. In many respects this process is incomplete, but further refinements should be possible without major design changes.

Section 2 introduces some important preliminary notions. The specification of the lexicon, from a user's point of view, is presented by example in section 3. Section 4 describes the operation of the lexicon handler, and the interface to the ProGram parser.

2. Some Important Preliminaries.

A few points concerning Generalised Phrase Structure Grammar (GPSG) and the ProGram parser should be noted. A thorough grasp of the points raised will be needed to understand the following sections of the paper. The reader familiar with both GPSG and the ProGram parser may skip most of this section, but should refer to the notational convention introduced in (P.8). Those interested in theoretical aspects of GPSG are recommended to consult the document "Generalised Phrase Structure Grammar: A Theoretical Synopsis" (Gazdar and Pullum.1982.) - henceforth GP82. Details of the ProGram parser may be found in "The ProGram Manual" (Evans and Gazdar.1984).

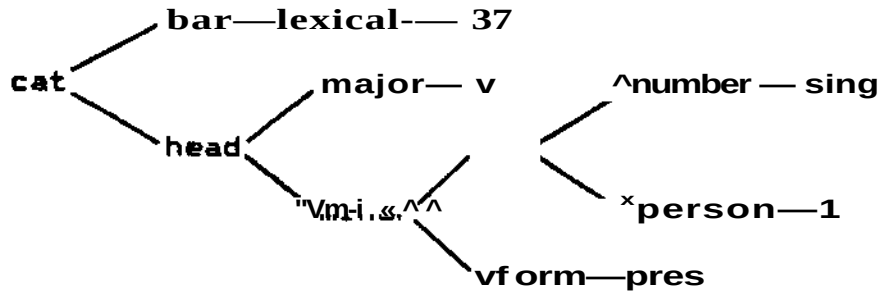
P.1. Bar level notation: GPSG employs category names such as $\bar{V}$, V and $\bar{N}$ (read as V bar, V and N bar respectively). Informally, the number of bars indicates the phrasal level of the category. Thus V, $\bar{V}$ and $\bar{N}$ correspond to the more traditional syntactic categories V, VP and NP. A category is LEXICAL if it has no bars (i.e. bar level zero). For example, V and N are LEXICAL.

P.2. Complex Symbols: In GPSG category names are not simple symbols, but have a complex internal structure. They stand for complexes of features, which we may call feature trees.

P.3. Feature Trees: A typical feature tree is

presented graphically in (1).

(1).



Feature tree (i) has the name CAT (i.e. it is a category). The -feature name CAT itself has two subfeatures, BAR and HEAD. Sub-feature LEXICAL of BAR indicates that this is a lexical category (bar level zero). The so-called HEAD features are further divided into MAJOR and MINOR. The MAJOR features specify the 'gross identity' of the category, in this case V for verbal. The MINOR features are an assortment of tense (VFORM), and agreement, markings for the category. Here they mark VFORM as PRES (present tense), PERSON as i (1st person) and NUMBER as SING (singular). In summary, the feature tree represents the category appropriate to a 1st person singular, present tense verb. We might have chosen to write this feature tree in the following way:

[cat, [bar, [lexical, 3733, [head, Cmajor, v3, [minor,*.mill*

The interpretation of this format should be clear.

P.4. Subcategorization: Words of the same lexical category may nonetheless 'prefer' different syntactic contexts (e.g. there are intransitive, transitive and ditransitive verbs). GPSG elegantly captures such facts about indi-

vidual words by giving syntactic rules 'rule-names' (e.g. 1, 37, trans, intrans). Roughly speaking, a word of some lexical category C, may only be introduced by a rule mentioning that category, if C bears the rule-name as one of its constituent features. This special feature is conventionally a sub-feature of LEXICAL so (1) has the "subcategorization feature' 37. The category may be thought of as belonging to a subclass, or subcategory of verb named 37.

F-5. Aliases; Aliases are a notational device, useful in specifying feature trees when writing grammars for the ProGram parser. Since identical feature trees may need to be written out many times, it is much simpler to write a single specification and give it a unique name. This name is known as the feature tree 'alias'. For example, category (1) might be given the alias vJL (or anything else appropriate).

P.6. Feature Syntax: So far the term 'feature' has been used freely without being given any precise definition. As might be expected, such a definition would involve details about the names and structure of features. However, it is not the intention to give a formal definition here, since it is not necessary for an understanding of the rest of the paper. (See GP82, pp 1 - 11 for a detailed account of syntactic features.) Instead an outline of 'feature syntax', as used to specify features for use by the ProGram parser, will be given. An example of a feature syntax definition is shown in (2).

(2>. feature Ccat, bar, head].
feature [bar, {lexical, 1, 2>]
feature [head, major, minor]
feature [major, Cv, n>]
feature [minor, agr, {case, vform>]
feature Cagr, person, number]
feature [number, {sing, plur>]
feature [vform, <pres, past}]
feature [case, tnom, poss}]
feature [person, (1, 2, 3>]

Amongst other things (2) indicates that CAT is a feature having two subfeatures BAR and HEAD. Similarly the feature HEAD has the subfeatures MAJOR and MINOR and so on. Curly brackets group mutually exclusive features. The feature BAR thus has one subfeature, with the value either LEXICAL i or 2. Feature tree (1) might well have been constructed in line with the syntax of (2>.

To make life a little easier in the rest of the paper, an effort will be made to adhere to the feature syntax of (2) whenever examples are given. The reader may then always refer to this syntax to clarify matters if confusion arises.

P«7« Feature Normalisation: It is permissible and convenient to omit certain details of features when using the ProGram parser. For example, (3) is an acceptable instance of the feature CAT although it apparently has no subfeature BAR

(3). [cat, [head, major, minor]]

However, before such features may be used by the parser program, they must be 'normalised'. The normalised version of feature (3) would appear much like <3'>, where F1,...,F4

stand in lieu of the subfeatures missing in the original specification.

(3'). [cat, F1, [head, [major, F2]], [minor, F3, F4]]

Normalisation thus brings out the precise structure of a given feature tree, though otherwise it does not supply any additional information.

P.8. A Notational Convention: Throughout the rest of the paper, where name is understood to be the name of an aliased feature, then Name may be taken as standing for its normalised form. For example, if verb is an alias, then so long as there is no danger of confusion, Verb will denote its normalised counterpart.

3. User Specification of the Lexicon.

It is convenient to think of the lexicon as divided up into two main parts:

1. The Permanent Lexicon*. That is, the repository of all unanalysable or otherwise idiosyncratic word forms, including word roots and irregular forms.

2. The Morphological Component. i.e. rules of word formation and spelling, etc.

(*). The term *permanent lexicon* is taken from Rochelle Lieber (1981), although the sense in which it is used here is not precisely as she intended. For Lieber, the *permanent lexicon* contains all simple morphemes, including affixes, whereas in the present system the affix dictionary is listed as part of the *morphological component*.

The specification of these components by the user will now be informally introduced using examples.

3.1. The Permanent Lexicon

To enter a word into the permanent lexicon the user must specify:

i. The category of the word (e.g. noun, verb, adjective, etc.), and

ii. Subcategorization information: i.e. the name of a particular syntactic rule introducing a word of this type.

The permanent lexicon consists of a number of category entries, each of which in turn is divided into several subcategory entries. For example, consider the simple category entry given below.

(4). verb: trans --> {love, hate},
ditran --> {buy # (past # bought)}.

Category entry (4) states that the words love, hate, buy and bought are all of the category verb (verb is an alias). The rule names indicate that love and hate are to be further classified as trans (i.e. transitive), while buy and bought are ditran (ditransitive). Exactly what features make up the category verb is up to whoever is writing the lexicon. In general however, it is to be expected that, apart from the features [cat, bar, head] with [bar, lexical], the category will bear little more than the MAJOR sub-

features. This is since the MINOR features (whatever they may be) must be free to be specified at a later stage. Feature conventions and/or morphological rules, will ensure that they become 'filled in' before the category is used by the parser.

It is best to consider the word entries love and hate in (4) as being verb roots (subclass transitive). Likewise buy is a verb root of subcategory ditransitive, but in this case extra information is supplied indicating that it is irregular in the past tense. The special symbol # is used to associate with a word root, a list of irregular 'forms'. A form may either be simple (i.e. a word) or complex, in which case it consists in a feature and a forms list separated by #. For the word root buy the associated forms list has only one member, made up of the feature PAST and the simple form bought. So, the verb root buy has a single irregular form in the past tense, specifically, bought rather than *buyed. The entry for buy is a simple example of a 'word tree'. A more complex example would be the word tree for the highly irregular verb be, shown in (5). (The layout is intended to make the word tree easier to read, but is otherwise unimportant).

```
(5).    be # (pres # {sing # {person # (1 # am,
                                        2 # are,
                                        3 # is)},
                      plur # are},
              past # {sing # {person # (1 # was,
                                        2 # were,
                                        3 # was)},
                      plur # were})
```

It should be reasonably clear, aside from certain details, just what is meant by (5). It states, for example, that the 1st person singular present of be is the word am, and that all of the plural past tense forms are were. In the latter case it is simply that the word tree specifies no more than that were is plural and past tense, which permits such an interpretation. Actually it would be permissible to specify each person separately, but this would introduce redundancy, exactly what we are trying to avoid. Returning to (4), note that the complete lack of any information concerning other word forms in the entries for love and hate effectively marks them as perfectly regular. The entry for buy, on the other hand, exhibits regularity "up to a point", that is buy is irregular only in respect of its past tense form bought.

One extra point about word trees is of interest. It may be noted that (5) uses sometimes curly, sometimes round brackets to group together particular terms. The round and curly brackets have different interpretations, and are useful in making precise the meanings of word trees. Suppose that instead of the feature names SING and PLUR used in word tree (5) (and the feature syntax of (2)) the grammar writer had, quite reasonably, used the names 1 and 2. If this were the case, then the feature names 1 and 2 would be ambiguous, being interpretable either as markings for NUMBER or PERSON. Use of round brackets gets over this problem. Any feature mentioned inside round brackets is to be taken as a

sub-feature of the -feature directly outside the brackets. Thus in <5> the -features 1, 2 and 3 *are* all subfeatures of PERSON Curly brackets impose no such constraint, and in the entry -for be, PERSON need not be a subfeature o-f SING nor SING a sub-feature of PRES (though in -fact they might be in reality; when there can be no possible con-fusion, round brackets are not mandatory)•

This means that the round brackets used in (5) *are* really unnecessary, since the features *are* unambiguous. Using them can make the structure of features *dearer* however. They may serve as a visual aid, helping to reduce the possibility of errors whilst specifying the lexicon. In addition their use will tend to make life easier for the program responsible for normalising the permanent lexicon (the normalisation process is described in section 4.1).

3.2. The Morphological Component

The morphological component of the lexicon may be further divided up into four sub-components. These sub-components are responsible for capturing the morphological regularities governing the words listed in the permanent lexicon. Each will be described in turn.

3.2.1. Morphological Rules.

Morphological rules are intended to capture generalisations about the regular morphological processes of a language. A typical rule for English might be the following

(clearly a rule would also be needed for verbs).

(6). nmorph: noun -> noun, nsuff marked [number].

The rule simply states that a noun may consist of a noun stem followed by a suffix marked for the feature NUMBER (i.e. giving subfeatures of NUMBER). Both noun and nsuff are aliases and stand for appropriate feature trees. The rule label nmorph (short for 'noun morphology') is analogous to the names associated with syntactic rules, i.e. it provides subcategorization data. In this case it is the suffix(es) to be introduced by the rule with which we are interested.

3.2.2. The Affix Dictionary.

The affix dictionary has entries detailing the various inflectional affixes that occur in a language. For example the following entry might appear as part of the affix dictionary for English.

(7). nmorph: nsuff --> s + s1.

Affix entry (7) states that the morpheme 's' belongs to the category nsuff (an alias) and may be introduced by a morphological rule having the name nmorph. It should be clear that this permits the suffix 's' to be introduced by rule (6) of section 3.2.1.

The appendage '+ s1' is an indication that in some cases regular word roots taking the suffix 's' may undergo a

certain amount of re-spelling. As an example, it is convenient to think of the noun box as forming a regular plural boxes with the noun suffix 's'. The root has been modified slightly however, by the addition of the letter 'e#'. To analyse such cases correctly, it is necessary to specify precisely what kinds of adjustments in spelling may occur. Each suffix has associated with it the name of a 'spell-rule' to do just that, and in this instance it is the spell-rule si.

3*2.3. Spelling Rules.

To continue the example started in the previous section, spell-rule sJL might be specified as follows:

(8).

```
si: ?/e  —> ?/C3,
    ?/ie —-> ?/y,
    ?/ve —> ?/f.
```

Rule (8) consists of a rule name, and then a number of re-write rules separated by commas. Each re-write rule states that some sequence of letters may be rewritten as a new sequence of letters. The special symbol '?' matches any letter sequence, but specifies the same sequence on both sides of the arrow. Thus the first rewrite rule of <B) indicates that a sequence of letters ending in 'e', may be rewritten as the same initial sequence ,but now ending in CD. By this rule boxe may be rewritten to box followed by C3. Since *LI* represents the null sequence of letters then this means that boxe simply becomes box as required.

Similarly the second and third rewrite rules will deal with putative word roots such as flie (from flies) and thieve (from thieves). Actually, both flie and thieve also match the first re-write rule (and would be respelled as fli and thiev respectively). This is not a problem since the lexicon will recognise that neither constitute valid word roots. Even so, spell-rule (8) should really be rather more choosy about just when it will allow 'e' to be stripped from a putative word root. As it stands, it is possible that panes might get taken as the plural of pan, rather than pane. Of course, if pane is listed in the permanent lexicon, then the correct interpretation will be located before this occurs. The difficulty is that any further analysis of pane will invoke (8), and eventually turn up the anomalous alternative. Preventing such "overgeneration" would require that the first re-write rule of (8) be replaced by several rules of a more specific nature.

3.2.4. Feature Conventions.

Typical feature conventions for the category verb might be:

```
(9.a).    verb convention number marked sing,
                          person marked 1 or 2,
                          vform  marked pres.

(9.b).    verb convention number marked plur,
                          person marked 1 or 2 or 3,
                          vform  marked pres.
```

where verb is an alias.

Feature conventions (9, a) and <9.b) allow any word of category verb to be construed as being of either 1st or 2nd person singular present, or lst_f 2nd or 3rd person plural present. The conventions could apply to the words love, hate and buy in category entry (4) of section 3.1. They won't apply to bought however, since this already has VFORM marked PAST. If v-past is a category alias of precisely the same form as verb, but with the additional feature specification tvform, past], then convention (10) will apply to bought,

(10). v—past convention number marked sing or plur,
person marked 1 or 2 or 3.

It says that any word of category v-past may be taken as 1st, 2nd or 3rd person singular or plural. A little reflection should show that this is just what is wanted, both for bought and in the general case.

4. The Lexicon Handler Program.

The lexicon handler program is made up of six independant modules. *Each of these modules is written in the programming language Prolog, some knowledge of which may help in understanding various details of their operation, it is not essential however. For a comprehensive guide to Prolog the reader is referred to "Programming in Prolog[11], Clocksin and Mellish (19B1).

Once the user has specified a lexicon as described in section 3, each of the resulting data files (i.e. permanent lexicon, morphological rules, etc.) must be normalised.

Broadly speaking, the normalisation process takes a data file, and converts it into a form that may be used directly by the lexicon interface. The six modules then consist of five normalisation modules and the lexicon interface itself. Normalisation will now be briefly described, followed by the operation of the lexicon interface module.

4.1. Normalisation

Normalisation of the permanent lexicon, converts each entry into a set of Prolog clauses, one 'word clause' for each word mentioned in an entry. This process is carried out by the predicate NORMPLEX, which produces Prolog clauses of the form:

(11). <key>(<Cat>, <rule_name>, <reference>).

The <key> is the name of the word for which the clause has been built, prefixed by 'l_'. <Cat> is a normalised syntactic category, <rule_name> is the name of some syntactic rule and <reference> gives information about other irregular forms of the same word.

For entry (4) of section 3.1, the following word clause would be produced for the verb love:

(12). l_love(Verb, trans, [])

Here Verb is the category, and trans the appropriate rule name. The reference field of word clause (12) is the empty list '[]' as love has no irregular forms. Appropriate clauses for buy and bought would be:

(13.a). 1_buy(Verb, ditran, *Itmi*nor,*t*vfarm$_f$past33,bought3>
(13.b), l_bought(V-past, ditran, buy).

The reference field for buy shows that it has an irregular form bought. A tree of MINOR features associated with this irregular form marks it as past tense. The word clause for bought has the (normalised) category V-past, which will be identical to Verb except that it will bear the additional MINOR feature specification tvform, past3. Note that these are just those MINOR features to be found along with bought in the reference of (13.a), the word clause for buy. The reference field of (13.b) on the other hand, names the word of which bought is an irregular form: i.e. the word root buy.

Normalisation of the morphological rules is performed by the predicate NORMMORPH. Each'rule is converted to a MORPHRULE clause of the form:

(14). morphrule(<r_name>, <Catl>, <Cat2>, <s_list>).

In (14), <r_name> is the name of the user specified morphological rule for which the clause has been built. Both <Catl> and <Cat2> *&r&* normalised categories, and <s_list> is a list of suffix specifications. For rule (6) of section 3.2.1 the appropriate MORPHRULE would be (6').

(6').
morphrule(nmorph, Noun, Noun, CNsuff marked Cnumber33).

Again nmorph is the rule name. The list of suffixes consists in a single specification for a suffix of category

Nsuff which should be marked for the feature NUMBER.

Normalisation of the affix dictionary is also relatively simple, and is carried out by NORMAFF. Entry (7) of section 3.2.2 has as its normalised form (7').

(7'). affrule(s + s1, nmorph, Nsuff).

The correspondence between affix entry (7) and its normalised counterpart (7') should be relatively clear. Things aren't always quite so straight forward though, sometimes an entry will specify more than one suffix. For example:

(15). vmorph: vsuff --> s + s1, ed + s2

This might be the entry for the verb endings 's' and 'ed' (i.e. loves and loved). The clause generated for (15) by NORMAFF would be:

```
(15'). affrule(A, nmorph, Vsuff) :-
                member(A, [s + s1, ed + s2]).
```

Clause (15') says that an affix A is of category Vsuff, subclass nmorph, if it is a member of the list of suffixes (15'').

(15''). [s + s1, ed + s2].

Predicate NORMSPELL converts each spell-rule into a number of clauses. For spell-rule (8) of section 3.2.3. three clauses will be produced, one for each re-write rule. In general however a single re-write rule may result in many clauses. For example, re-write rule (16) deals (in a lim-

ited fashion only) with so-called consonant doubling. It will re-spell putative verb roots such as gass, patt or sagg - which may result after removal of the inflectional suffix 'ed' - to their correct forms gas, pat and sag.

(16). ?/[s,t,g] --> ?/[]

Assuming that (16) is part of some spell-rule named s3 then three clauses will be produced by the normalisation process. Each has the form of (16').

```
(16').      spell(s3, X, Y) :-
                    append(Z, A1, X),
                    append(Z, A2, Y), !.
```

In the actual clauses A1 will correspond to one of the three letter sequences 's', 't' or 'g', and A2 to the null letter sequence []. Clause (16') will then re-write a sequence of letters Z followed by A1 ('s' say), as a new letter sequence Z followed by A2. Note that this is just Z, since A2 is the null sequence.

The predicate NORMCONV is responsible for converting the feature conventions specified by the user into various CONVRULEs. Each convention results in a single CONVRULE of the following form being produced:

```
(17).       convrule(<NCat>, C) :-
                    member(C, <Cat_list>).
```

Here <NCat> is a normalised category, and <Cat_list> is a list of normalised categories. Convention (18) would be normalised to yield clause (18').

(18). v convention number marked sing or plur

(18'). convrule(V, C) :-
member(C, [V-sing, V-plur]).

The <Cat_list> contains the two normalised categories V-sing and V-plur, identical to V, but with NUMBER marked SING and PLUR respectively. Thus (18') ensures that a word of category V may, by convention, either be taken as having the category V-sing or V-plur.

4.2. The Lexicon Interface Program

In the following description of program operation, it will be convenient to refer to the various normalised components of the lexicon by the names of their user-specified forms. Thus 'permanent lexicon' should be understood not as the file of user-specified lexical entries, but as the set of clauses produced by the program NORMPLEX during the normalisation process. Similarly for 'morphological rules' etc.

The top-level interface to the ProGram parser is the predicate LEXRULE. Given a word, LEXRULE's job is to ascertain the category, and the name of the syntactic rule which introduces it. To start with, LEXRULE just looks to see whether the word may be found amongst those listed in the permanent lexicon. It builds a word clause of the same form as those returned by NORMPLEX, and then checks to see if a matching clause was produced during normalisation. If such a clause exists, LEXRULE notes the specified category, rule-name and reference, and then goes on to perform any

feature conventions applicable. Finally, a check is made to ensure that the resultant category is acceptable for the current word. The check is necessary because feature conventions *Are* generalisations which do not apply in certain specific instances. For example, a feature convention might capture the generalisation that words of category verb may be taken as 1st or 2nd person singular present (e.g. convention (9.a) of section 3.2.4) This is fine far verbs such as love, but not for irregular verbs like be, since this already has the 1st and 2nd person singular present forms am and are. It is easy to prevent feature conventions from applying in such cases. All that is needed is to look at the current word's reference, which lists each of its irregular forms and their associated MINOR features. As an example, if be is the word which has just been looked up, then the reference field of its word clause will list the irregular forms am and are. Since these will have the MINOR features for 1st and 2nd person singular present respectively, the feature convention will be known not to apply. In this way, 'regular' interpretations of irregular words can be avoided.

The same interpretations would, of course, apply to a regular verb such as love, because its reference field is the empty list (i.e. love has no irregular forms). Consequently, LEXRULE will return the first such interpretation it finds (and a rule-name). If LEXRULE is made to re-analyse love, then further interpretations may be found. Each new analysis carried out by LEXRULE will find an alternative

category specification, until no more *arm* available.

If LEXRULE is asked for details of a word not appearing in the permanent lexicon, then the 'direct' strategy outlined above will clearly fail. Supposing that this has happened, then LEXRULE will adopt a new strategy, namely 'morphological analysis'. Morphological analysis involves the following operations:

1. Choose a (previously unselected) morphological rule - if none are left then FAIL.

2. Parse the current word according to the rule selected at 1. If this is successful then the result is a putative word root with a tentative category assignment. Additionally, a Combined Affix Category, bearing all those MINOR features associated with the various inflectional affixes found during the parse will be returned. Continue to the next stage of the process (3). Alternatively, if the parse fails, re-do 1.

3. Look up in the permanent lexicon the word root returned by stage 2. If successful (i.e. the word is listed) then continue to stage 4 with the appropriate word category and rule-name. If unsuccessful, then attempt to respell the root and if this can be done re-do 3; otherwise re-do 2.

4. Unify the category returned by stage 3 with the Combined Affix Category returned by stage 2. The result is the

Basic Category to be associated with the current word.

5. Apply any -feature conventions that pertain to the Basic Category, to give the Final Category.

6. Check that the Final Category resulting -from 5 is acceptable for the current word.

This process may be clarified by means o-f a simple example.

To start with, suppose that the user has specified the following morphological rules, affix dictionary entry and spell-rule:

```
nmorphs noun —> noun, nsuff marked [number].
vmorph: verb —> verb, vsuff marked Cvform, number, person]

nmorph: nsuff —> s + spell27.
vmorph: vsuff —> ed + spell33, s + spel!27.

spell27: ?/e --> ?/[].
```

where for present purposes the category vsuff is marked for the features:

```
Cvform, pres]
tnumber, sing]
Cperson, 33
```

Additionally, suppose that catch is listed in the permanent lexicon as a verb of subcategory trans, and that LEXRULE has just failed to locate catches in the permanent lexicon. Lexrule therefore changes its strategy to morphological analysis and begins by selecting the morphological rule

»

nmorph (stage 1). An attempt is now made to parse catches according to this rule (stage 2). The affix dictionary is first consulted to see if there is a listed suffix which may be introduced by the rule nmorph. This succeeds since the suffix 's' (and its associated spell-rule spell27) may be selected. As catches ends in 's', it may be analysed into the putative word root catche and inflectional morpheme 's' in accordance with the chosen morphological rule. A category noun is tentatively assigned to catche, and a note made of the category belonging to the suffix (nsuff). Stage 2 therefore concludes successfully and stage 3 begins. Lexrule now looks up catche in the permanent lexicon. It can be assumed that this fails (since catche is not a word) so re-spelling is applied to the putative root. The spell-rule chosen is spell27, as this is the rule-name linked with 's' in the affix dictionary. The re-write rule

?/e --> ?/[]

will then re-spell catche to catch, which may again be looked up in the permanent lexicon.

At this point, LEXRULE has almost succeeded in correctly identifying catches as a plural noun. It will go on to return this interpretation after completing stages 4, 5, and 6. For the sake of argument however, suppose that the plural noun catches is not required. Alternative syntactic categories may be obtained by Prolog backtracking. The effect of this is to restart morphological analysis at stage

2v just as if stage 3 had actually failed.

Stage 2 begins as be-fore with the affix dictionary being consulted, but as no -further suffixes may be found to be introduced by nmorph. it quickly fails.

Stage 1 now recommences with the selection of a new morphological rule vmorph. As a result, consultation of the affix dictionary yields the suffix 'ed' which may be intro-duced by this rule. This time however, it is impossible to analyse catches as a letter sequence ending in 'ed'. The affix dictionary is consulted yet again, and the further inflectional suffix 's' is returned. The word catches is now successfully parsed into the constituents catche and 's'. The category tentatively assigned to catche on this occasion is that of a verb.

At stage 3 LEXRULE will initially fail to find a verb called catche in the permanent lexicon. After re-spelling - again with spel127 - an entry for the word catch is identi-fied, with category verb and subcategory trans. This con-firms that the analysis is essentially correct. The unifica-tion process at stage 4 takes the category found for catch in the permanent lexicon at stage 3, and effectively adds to it the category vsuff associated with the suffix 's'. This gives the Basic Category specification for catches, indicat-ing that it is a verb with Cvform, past], [number, sing] and Cperson, 3]. Analysis is now almost complete; all that remains is to apply any appropriate feature conventions and

to check that the resulting category is acceptable. Assuming that no conventions apply to the Basic Category, stages 5 and 6 now succeed. Lexrule will return a Final Category *or catches appropriate to a 3rd person singular, present tense verb, and in addition indicate that the subcategory is transitive. This is just as required.

5. Concluding Remarks.

What has been described is a program designed to aid in the specification of a lexicon for the ProGram parser. The lexicon handler allows the user to make generalisations concerning listed words, thereby eliminating much of the redundancy inherent in the existing system. For example, rules may be written to capture regularities in the formation of words through processes of inflectional morphology, and feature conventions may be stated allowing significant reductions to be made in the number of word entries which need to be specified.

Despite the major advantages offered by the ^lexicon handler over the existing system, further work is envisaged to reduce still further the effort required in building a lexicon for the Grammar Development System. A part of this work will be directed towards finalising certain details of the current program implementation, and the formalisms devised for the specification of the different lexicon components. In particular, it is felt that certain aspects of the morphological component could be improved upon. The

modular design approach ensures that any modifications may be implemented and evaluated against the background of a substantially complete, working system.

For the rest, there is still more to be done in eliminating redundancy. The lexicon handler is currently incapable of dealing with sub-regularities, regularities holding between small classes of otherwise irregular words (examples in English include: teach and taught, seek and sought, catch and caught, etc; as well as, sing sang and sung, ring, rang, and rung). Neither can it cope with homonyms, i.e. different words having the same morphology (an example is the English verb do which has both an auxiliary and a non-auxiliary variant). It is not anticipated that incorporating mechanisms to deal with such phenomena into the lexicon handler framework will present serious difficulties.

References

Clocksin, William, and Christopher Mellish<1981) *Programming in Prolog,* Berlin: Springer—Verlag.

Evans, Roger, and Gerald Gazdar(1984) *The ProGram Manual. University of Sussex Cognitive Science Research Paper 35.* (CSRP 035).

Gazdar, Gerald, and Geoffrey Pullum(1982) *Generalised phrase structure grammar: a theoretical synopsis.* Bloomington: Indiana University Linguistics Club mimeo. Also available as *University of Sussex Cognitive Science Research Paper* Wo *7.* (CSRP, 007).

Lieber, Rochelle(1981) *On the organisation of the lexicon.* Bloomington: Indiana University Linguistics Club.

H.R. Keller. June 1984

Appendix.

A Lexicon Fragment.

What follows is a small lexicon fragment, just to demonstrate various aspects of lexicon specification. A slightly expanded feature syntax to that used in the main text is assumed.

Top level features: ROOT has two subfeatures CAT and FOOT. FOOT features can be ignored for present purposes.

```
feature [root,  cat,    foot].
feature [cat,   bar,    head].
feature [bar,   {lexical, 1, 2}].
feature [head,  major,  minor].
```

Major features:

```
feature [major, {v, n}].
```

Minor features: VFORM may be AUX (auxilliary) as well as either PAST, PRES, PSVE (passive) or PRESP (present participle).

```
feature [minor, agr, {case, vform}].
feature [agr,   number, person].
feature [vform, aux, {past, pres, psve, presp}].
feature [case,  {nom, poss}].
feature [person, {1, 2, 3}].
feature [number, {sing, plur}].
```

The following alias specifications are used in the lexicon fragment. To begin with, here are the NOUN and VERB categories (note the minimally specified MINOR features).

```
alias( noun, [root,[cat,[bar,lexical],
                   [head,[major,n],minor]]] ).

alias( verb, [root,[cat,[bar,lexical],
                   [head,[major,v],minor]]] ).
```

Rather more complicated aliases for past tense verbs and auxilliary verbs:

```
alias( v_past_f  troot,Ccat,Cbar,lexical3,
                 Chead,Cmajor_f v3,
                 [minor,Cvform^past]]]]] >.

alias( v_aux,    Croot, Ccat,Cbar,lexical 3,
                 Chead,Cmajor,v3,
                 Cminor, Cv-form,aux3 3 3 3 ).
```

Suf-fix category aliases -for past tense, third person singular present, passive and present participle respectively. In this case it is the MAJOR -Features which are absent.

```
alias ( vsu-f-fl,  Croot, Ccat, Chead,
                   Cminor ,agr , Cv-form,past3 3 3 3 3 ).

alias( vsuf*2,     Croot,Ccat,Chead,
                   Cminor,
                   Cagr,Cnumber,sing 3,Cperson,33 3,
                   Cvform,pres33333 ).

alias( vsu-f-f3,   Croot, Ccat, Chead,
                   Cminor,agr,Cvform,psve3 3 3 3 3 ).

alias( vsuff4,     Croot,Ccat,Chead,
                   Cminor,agr,Cvform,presp3 3 3 3 3 ).
```

A single noun suf-fix category NSUFF. This has NUMBER marked PLUR and CASE marked marked NOM.

```
alias( nsu-f-f,  Croot, Ccat, Chead,
                 Cminor,Cagr,Cnumber,plur33,
                 Cease,nom33333 ).
```

Finally a minimal AFFIX category:

```
alias( af-fix,  Croot, Ccat, Chead, Cminor3 3 3 3 )•
```

The Permanent Lexicon.

The permanent lexicon consists o-f VERB, V_AUX and NOUN category entries. The VERB entry is subdivided into three subcategory entries:

VP_2s Transitive verbs.

VP_3: Ditransitive verbs.

VP_4: Verbs taking sentence complements.

```
verb:   vp_2 --> {  love, hate,
                    catch #  (past # caught,
                              psve # caught),
                    see   #  (past # saw,
                              psve # seen)    },
        vp_3 --> {  hand,
                    give  #  (past # gave,
                              psve # given),
                    buy   #  (past # bought,
                              psve # bought) },
        vp_4 --> {  believe,
                    think #  (past # thought,
                              psve # thought),
                    know  #  (past # knew,
                              psve # known)   }.
```

Category V_AUX (auxilliary verb) has a single member in this lexicon fragment: the irregular verb be.

```
v_aux: vp_5 --> { be # (pres # {sing # {person # (1 # am,
                                                  2 # are,
                                                  3 # is)},
                               plur # are},
                       past # {sing # {person # (1 # was,
                                                  2 # were,
                                                  3 # was)},
                               plur # were},
                       psve # been)  }.
```

Finally a category entry for (count) nouns.

```
noun:   nb_1 --> {  book, thief, fly, hand, catch,
                    man   #  (plur # men),
                    woman #  (plur # women)  }.
```

The Morphological Component.

i. Morphological Rules.

The following rules deal with verb morphology (VMORPH) and noun morphology (MNMORPH). The category for which AFFIX is an alias does for both rules in this case.

Verbs may take suffixes marked for AGR and VFORM:

```
vmorph: verb -> verb, affix marked [agr, vform].
```

Nouns may take suffixes marked for NUMBER:

```
nmorph: noun -> noun, affix marked [number].
```

ii. The Affix Dictionary

Five affix dictionary entries are given here in all, four of which pertain to verbs. The suffix categories VSUFF1,..,VSUFF4 are for past tense, third person singular present, passive and present participle respectively.

```
vmorph: vsuff1 ->  ed + s1.

vmorph: vsuff2 ->  s + s2.

vmorph: vsuff3 ->  ed + s1.

vmorph: vsuff4 ->  ing + s1.
```

A single entry for the plural suffix 's' which attaches to nouns. The category NSUFF has NUMBER marked PLUR.

```
nmorph: nsuff  ->  s + s2.
```

iii. Spelling Rules.

The spell-rules S1 and S2 are not particularly practical, but suffice for the present example.

Spell-rule S1 corrects the spelling of word roots following the removal of 'ed' or 'ing'. For example, the first rewrite rule will take lov to love.

```
s1: ?/v --> ?/ve,
    ?/t --> ?/te,
    ?/c --> ?/ce.
```

The second spell-rule will strip an 'e' following the removal of 's'; e.g. catche to catch.

```
s2: ?/he --> ?/h,
    ?/ve --> ?/f,
    ?/ie --> ?/y,
    ?/se --> ?/s.
```

iv. Feature Conventions.

Feature conventions do quite a lot of work. The first two allow anything of category VERB to be either 1st or 2nd person singular present, or 1st, 2nd or 3rd person plural present.

verb convention number marked sing,
person marked 1 or 2_f
vform marked pres.

verb convention number marked plur,
person marked 1 or 2 or 3_f
vform marked pres.

The following convention is for past tense verbs (V_PAST> and permits words such as caught or loved to be taken as 1st, 2nd or 3rd person singular or present*

v_jpast convention number marked sing or plur,
person marked 1 or 2 or 3.

Finally, two conventions for nouns. They allow anything of category NOUN to be singular and nominative, or singular with case unmarked (which can be taken to mean accusative).

noun convention case marked nom_f
number marked sing.

noun convention number marked sing.

A Lexicon Handler for The ProGram Grammar Development System.

W.R. Keller

Cognitive Studies Research Paper

Serial no: CSRP 40

June 1984

ABSTRACT

A lexicon handler for the ProGram Grammar Development System is described. Redundancy in the specification of word entries is avoided by considering the lexicon to consist of two parts: a permanent lexicon or dictionary, and a morphological component. The user may record base forms and irregular word forms in the permanent lexicon in a succinct and uniform way. The morphological component is further subdivided to permit word formation rules and feature conventions to be stated. These rules and conventions are useful in capturing generalisations about the processes of inflectional morphology.

A version of the lexicon handler has been implemented in the programming language Prolog. The program performs two distinct tasks: the 'normalisation' of user-defined lexicon components, and the provision of an interface between the lexicon and the ProGram parser. Both aspects of program operation are explained by reference to examples.

Table of Contents