FINDERLISTS, COMPUTER GENERATED, FOR BILINGUAL DICTIONARIES

David Nathan & Peter Austin

1. Introduction
2. Dictionaries and finderlists
3. Producing finderlists
4. The Computer generated finderlist system
5. Computing and lexicography: aspects of data and processing

1. Introduction[1]

Compilers of bilingual dictionaries often take on the task of producing a finderlist or index to the dictionaries they compile. As T. Crowley (1986:249) points out: 'making a reverse section of a bilingual dictionary is not a simple matter of taking the left-hand sides and the right-hand sides of bilingual dictionary entries, and just swapping them over'. The dictionary maker must process the data, and this can involve several tasks.

The dictionary finderlist system developed for The Gascoyne-Ashburton Languages Project (henceforth GALP) is one part of a research project begun by Peter Austin at La Trobe University in 1981. The aim of the project is to document the Aboriginal languages spoken between the Gascoyne and Ashburton Rivers in the north-west of Western Australia (see Austin 1988, 1989). It aims to produce comprehensive grammatical descriptions of the phonology, morphology and syntax of the languages, together with text collections and dictionaries of various types.

From the beginning, GALP has used computer resources for the storage, manipulation and retrieval of linguistic data. Over time increasing sophisticated data models have been developed (see Austin 1983 for a report on the implementation of an early model), stabilising in 1987 on a network database design (Austin 1989, Linger 1989). This is implemented on La Trobe University's VAX 11/780 computer using DEC DBMS and Datatrieve software (together with some programs written in Pascal). Programs for data entry, manipulation, query, and reporting have been written and the system has been successfully employed for bilingual and multilingual dictionary production (see Austin and Nathan 1989 for an example). GALP uses the opportunities provided by computer information storage and processing in the context of a defined type of dictionary reversal to produce finderlists or indexes to bilingual dictionaries. While the finderlists are not wholly 'computer generated', and are not self-contained lexicographic works, they are useful resources for linguists and other dictionary users.

In the first part of this paper we survey some bilingual dictionaries in an attempt to draw up a typology of finderlists. We give a classification of finderlist types and the methods which have been employed to generate them.

---

[1] Research for this paper was supported by grants from the Australian Research Grants Committee (1983-86), the Australian Research Grants Council, the Australian Institute of Aboriginal Studies and the School of Humanities, La Trobe University. For their assistance with computational aspects of the research we are grateful to Randy Austin, Henry Linger, Geoff Webb, and Tim Woolford. An earlier version of this paper was presented at the Lexicography Workshop, Australian Linguistics Society annual conference held at Monash University in August 1989. We are grateful to workshop participants for comments on our presentation. Needless to say, none of these people can be held responsible for errors in what follows. The final version of this paper was written up while Austin was a visiting researcher at Xerox Palo Alto Research Centre. He is grateful to Joan Bresnan and Kris Halvorsen for making it possible to spend time in the stimulating environment at PARC.

Over the past twenty years lexicographers have increasingly turned to computers to assist them in their dictionary compilation, including the preparation of finderlists. Authors have called for a system which can automatically generate finderlists from source dictionary data (e.g. Hsu 1985:185, T. Crowley 1986:562). We surveyed a number of computer systems designed for producing finderlists, including the *Lexware* tools prepared at the University of Hawaii (Hsu and Peters 1984, Hsu 1985), and the system in use by the Summer Institute of Linguistics, Australian Aborigines & Islanders Branch. Such systems typically require the lexicographer to mark up glosses in various ways so that reorganisation of dictionary data can be *partially* automated. We show that the marking up approach violates general principles of data analysis and in any case is largely unnecessary. We present in Section 4 of this paper an account of a general computational strategy we have developed for English finderlist generation that involves no marking up of the glosses. This system has been shown to be over 95% effective on a data set.

The CGFL (Computer generated finderlist) system which we describe in Section 4 exploits the following to maximise the automation possibilities in finderlist production:

a) finderlists are not true reverse dictionaries;

b) finderlists are companions to bilingual dictionaries;

c) the system uses database storage of lexicographic data; and

d) computer programs can parse the structure of dictionary glosses.

The skills of those working in the GALP Project lie mainly in linguistics and not computing. However this has not dampened the development of a 'computer generated' finderlist system because the driving principle behind the system is a definition of finderlists. A finderlist has a definable relationship to a bilingual dictionary, and functions as an adjunct to that dictionary.

In a short final section we mention some principles underlying current computing work, to show that, while we need not be excessively diverted by the particulars of computer software etc, those of use who take seriously the use of computers in lexicography can learn valuable strategies for the analysis and approach to data from computing theorists.

2. Finderlists and dictionaries

2.1 Reverse bilingual dictionaries

A true reverse dictionary is the second part of a symmetrical, bidirectional, bilingual dictionary.

Even such reverse dictionaries are usually smaller than a standard monolingual dictionary: there is limited coverage of the source lexicon (Hartmann 1983:49), and definitions often include less information (Landeau 1984:8).

Zgusta has urged that bilingual dictionary definitions should be translation equivalents where possible. However this is made difficult by the difference in lexical domains which can occur when dealing with languages of communities with even small cultural differences (Zgusta 197 1:294, 312). So where definitions are not given as true translation equivalents, for example where explanations or phrases expressing genus plus differentia are provided, then the resultant set of glosses may contain words and structure quite unlike 'ordinary' usage of the target language. For example, a definition might be "net made of plaited reeds"; this complex English phrase fails to express the simple naming of what is perhaps an everyday object. In this way, the words used in the target language definitions of a bidirectional, bilingual dictionary may not even appear in the lexicon listed for that language.

Differences between syntactic systems of the source and target languages may cause some glosses to be grammatical ('metalinguistic') in style and thus not representative of a general usage lexicon of the target language.

Semantic texture, for example homonymy (where there are two separate words which coincidentally share the same shape) and polysemy (a single word with two or more related meanings), may not be represented so fully in a reverse dictionary as in monolingual dictionaries. This may be for reasons of space, but may also be due to a difference in the lexical semantics of the two languages (and possibly a lexicographer's unequal mastery of them).

Another reason for reduced semantic representation is that bilingual dictionaries usually have practical rather than descriptive uses — especially, translation (Zgusta 1971:213). The user who is a speaker of the target language is able to discriminate amongst multiple senses given in a definition because, probably, the word being looked up arises from a meaning context which allows disambiguation *(cf* Hartmann 1983:42). On the other hand, speakers of the source language are in a more difficult position. For them, the choice of the appropriate target-language definition depends on their knowledge of the target language itself since they are most likely seeking a word *form* in a less familiar language, not for a *meaning* expressed in their own first language.

Thus several problems face the compilers of reverse dictionaries, problems which tend to make bidirectional bilingual dictionaries less than the ideal mutual exposition of lexicons. In making finderlists, we *expand* the difference between reverse dictionaries and full monolingual dictionaries to the point where we can define exactly what is to be found in the 'reverse' part of the dictionary. We formalise finderlists to be wholly derivative from a fully-fledged single direction bilingual dictionary, and serving the purposes of speakers of the dictionary target language.

2.2 Finderlists

(1) below is an excerpt from a GALP finderlist for the north-western Australian language Payungu (Austin 1988, Austin & Nathan 1989).

Finderlists of this type, like those often provided with bilingual dictionaries (and sometimes called indexes) are not reverse dictionaries, but provide entry points, or 'keys' (T. Crowley 1986:253), to an accompanying dictionary (they also have other uses; see Section 3.2.2).

(1)

to keep quiet	karlaru-rri-ma
kerosene	wilukuru (d)
kick	kikama
kikama-rni-nma	
wirinyi-nma (a)	
kidney	kapurtutiny
kidney fat	manartu
to kill	pilarn-ma-nma
thinka-ma-nma	
killer	thinka-ma-lpaja (a)
kin, pair of	kurntal-karra
kin, pair of (father and son)	papu-yarra
kin, pair of (mother and son)	pipi-yarra
to kiss	punyja-nma (b)
wuparni-nmayi	
to kiss one another	wuparni-lparri-ma
kite, black	kirrkinyja
kite, grey	pujurra

to knead	thanuwa-rni-nma (b)
knee	murtinykaji (a)
nyama	
purtupuriny	
knee, sore	nyama-wartu

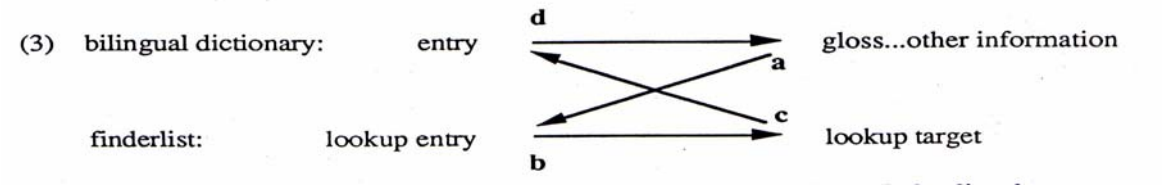
The finderlist is not a self-contained lexicographical resource because its alphabetised entries (on the left side in (1)), are not based on a source language lexicon but on a set of definitions or glossing expressions. These glossing expressions derive from definitions in an accompanying bilingual dictionary in the target language of that dictionary (English, in this case). For example, the finderlist entries "to kiss', 'knee" in (1) above derive from the information in the accompanying GALP bilingual dictionary for Payungu (Austin and Nathan 1989). See (2) below.

Alphabetised on the left in (2) are the Payungu entries and subentries, followed by their grammatical category and definition. To prevent ambiguity while referring to entries in both dictionaries and finderlists, we refer to the entries of the finderlist as 'lookup entries'; similarly, the word found as a result of looking up a lookup entry is a 'lookup target'. Thus, in (1), "knee" is a lookup entry, and "purtupuriny" is one of its (three) lookup targets.[2]

The main use of the finderlist is to allow bilingual dictionary entries to be located by a lookup of the finderlist. The lookup entry derives from a dictionary definition (or *gloss)* in finderlist production; in use, the lookup entry leads to the lookup target, which can then be found in the dictionary. See (3).

(2) **puni-ma** *vi* (a) to go
puni-marri-ma *vi* to go together
(b) to come
punthu Noun (1a) locked up
punthu-rla-nma *vi* (a) to close
(b) to lock up
(1b) confined
(1c) jail
(2) blunt, opp. jurirri, yin
puntirrpa *n* type of tree
punyja-nma *vi* (a) to lick
(b) to kiss, syn. wuparni-nmayi
purtiwarra *adv* (a) soon
(b) later
purtiya *adv* (a) almost (requires counterfactual inflection)
(b) nearly, (requires counterfactual inflection)
purtulya *n* brown bird, (possibly stubble quail Coturnix novaezeelandiae)
purtupuriny *n* knee

[2] Hsu (1985:33) calls 'the entire body of material associated with a given headword' an entry. We use 'entry' to mean the lemma, or entry point to the dictionary/finderlist; that is, the entry is the alphabetised information which is looked up. We call the 'body of material' an 'item'.



A lookup target, then, is a word or words found in a finderlist and also listed as an entry in a bilingual dictionary of the relevant language. Use of the term 'lookup target emphasises that it refers not to a meaning, but to the result of a lookup process which is only fully completed by a consultation of the lookup target's full bilingual dictionary entry.

A finderlist lookup entry can derive from a gloss for a single sense within a complex dictionary definition. Any one gloss (e.g. "confined" from (2)) exists in relation to all the dictionary entry's senses, which only together give the meaning of the entry. Thus the true role of a finderlist lookup entry can only be found by looking up the full glossing structure, showing the senses in their appropriate relation, within the lookup target's entry in the bilingual dictionary.

In our survey of finderlists we found that the relationships specified in (3) are in general not explicitly recognised by dictionary makers and instructions are rarely given to potential users of the finderlist on how to read the finderlist lookup entry plus lookup target information so that the user can navigate through these relationships.

In the simplest systems a lookup target is simply a dictionary entry in whose gloss(es) the lookup entry is to be found. In example (1) above, a lookup target for "knee" ("purtupuriny') is of this type. For the reader then, relationships c and d in (3) are instructions to go to the relevant part of the language-English dictionary and search for an entry of that form. A more complex system is where the lookup target incorporates further instructions to search within the *gloss(es)* provided (in the dictionary within the body of glossing information). So, the lookup target may consist of the relevant entry plus some index to the gloss body.

For example, consider the lookup target "punyja-nma (b)" from (1) above (it is to the right of 'to kiss'). This gives a reader the information that there is a dictionary entry "punyja-nma", and it also tells the reader to look at the subsection of the glossing for that entry labelled "(b)". See (2) above. Thus a lookup entry indexes more than a word form in the bilingual dictionary — it also 'navigates' the user through the semantic structure of the dictionary. A finderlist lookup is not completed until a reader has traced a path from finderlist to dictionary, from lookup entry to dictionary gloss, or, as shown in (3), b — c — d.

GALP finderlists make no explicit attempt to show semantic relations such as polysemy, homonymy, or synonymy among the lookup entries. Lookup entries are derived from the set of glossing definitions in the main dictionary, so they are really semantic labels, used metalinguistically but corresponding in form to ordinary expressions of the dictionary target language (see Section 3.2.3 for further discussion). And since the set of finderlist lookup entries is not a true language lexicon, we do not provide other 'normal' lexicographic information such as usage notes, paradigms and suppletion, and so on.

3. Producing finderlists

3. 1 Introduction

Since a dictionary has many layers of structure, making a finderlist is not as simple as reversing 'the left-hand. ..and the right-hand sides' of dictionary items (T. Crowley 1986:249). Consider a Payungu dictionary example:

(4) **nyina** (a) to live, to stay, to camp
(b) to exist (of animates)

On the left-hand side is one entry ("nyina"); on the right are four English expressions (glosses) which have been grouped on the basis of the semantics of Payungu. Each English expression (e.g. "to live") is syntactically independent, having structure according to the grammar of English. There is no simple way to flip this entire item about.

There are three steps in constructing finderlists (adapted from T. Crowley (1986:247ff) — dismantling, regrouping, and reorganisation:

a) Dismantling the glosses. In the example above, dismantling means extracting from gloss expressions the semantically relevant words — "live", "stay", 'camp", "exist". These words are called *keywords.*[3]

b) Regrouping involves ordering all the dictionary items **so** that those sharing the same keywords (from the dismantled glosses) come together. Thus all the lookup targets which have "live" in (one of) their glosses will be found adjacent in the finderlist.

c) Finally in the third, reorganising, phase the full semantic specifications of each gloss are restored, and each lookup target is placed under a complete glossing expression.

Dismantling glosses is usually the lexicographer's task of deciding which words within a gloss have key semantic significance. The process involves working with a bilingual dictionary, examining the glosses, and then recording in some way the keywords to be extracted. It may also involve 'adjusting' the original glosses to make more useful keywords available (Kari 1989:10; see also Sections 4.3 - 4.4). Regrouping is essentially an ordering and formatting operation. The third phase, reorganising the new finderlist entries, usually involves some kind of copying, perhaps copying the entire original gloss string next to each of its keywords.

We carried out a survey of a representative set of dictionaries with finderlists. We found that it is only in the implementation of the third or reorganising phase that most finderlists differ significantly. In the following section we examine the commonest methods used — prekeyword suppression, keyword fronting, and keyword classification.

3.2 Finderlist strategies: listing the lookup entries

In the following sections we survey some methods used to present finderlists, especially their lookup entries. Note that a finderlist's presentation may not display the way it was produced, including the extent to which production was aided by computing techniques.

3.2.1 Pre-keyword suppression.

In this system, full gloss or definition expressions are written out as finderlist lookup entries, but parentheses are inserted to 'suppress' the part of the gloss occurring before the selected keyword. In this way, a sorting routine gathers together lookup entries on the basis of the unsuppressed material outside the parentheses. An example of this common strategy is shown in *(5)* below, adapted from the *Fore Dictionary* (Scott 1980:151):

(5)

fire	yakune
(create) fire	ikuwe
(light) fire by friction	kigi'na aeyuwe
(edging on step or around) fireplace	tuke yawe
fireplace	tane
⋮	
light fire by friction	kigi'na aeyuwe

The advantages of pre-keyword suppression are that any word within the gloss may be chosen as a keyword. There is no constraint on the form of the lookup entry string — it may

[3] Landeau (1984:279) calls these words 'sortwords'; Tilley (1966:xi) calls them catchwords'.

be of any length, and of any syntactic construction, although in practice length limits will be imposed for typesetting. Once located, the lookup entries are read by ignoring the suppression characters (usually parentheses).

The keyword in context format, used by the Oxford Concordance Program to produce concordances, marks its keywords by aligning them down the page, placing the context, or 'suppressed' words to the left and right of the keyword column. Following is a sample adapted from Butler *1985:54:*

(6)

	DREAM
GIVE CHARGE TO MORPHEUS THAT HE MAY	DREAM
	A GOLDEN DREAM, AND OF THE SUDDEN WALK
	FATHER
THEN,	FATHER, HERE RECEIVE THY HAPPINESS
	FATHER, IT DRAWETH TOWARDS MIDNIGHT NOW
	HAPPY
AS BUT TO BE ABOUT THIS	HAPPY PLACE
TIS NOT SO	HAPPY YET, WHEN WE PARTED LAST

This system has also been used for finderlists (cf Hsu and Peters 1984:240. With vertical alignment, the access and readability problem mentioned below is decreased, although the length of suppressed material will need to be fixed in advance to prevent words spilling off to the left of the page or leaving too much blank space in the case of short lookup entries.

Another variant of pre-keyword suppression involves a change of type style to emphasise the keyword(s). This is used in a dictionary of proverbs by Tilley (1966), who explains the system:

> Each proverb has a catchword, which is the word (or words) appearing in boldface.. .small capitals in the entry form. The first substantive (excluding pronouns) in the entry form is chosen as the catchword. If no substantive occurs in the entry form, the first finite verb is the catchword... Phrases of two or more words are treated as compound catchwords.. .all important elements of the compounds are entered in the Index of Significant Words.
>
> (Tilley 1966:xi).

Thus we find entries such as (7) (cf also (12)):

(7) When a **DOG** is drowning everyone offers him a drink.

We found one example of an attempt to 'suppress' pre-keyword material by placing it above, and indented relative to, the keyword. This formatting solution makes the finderlist virtually unreadable:

(8)

	digging	
stick	wanna	
	little	
stick	kadi(r)i	(from Hall 1971:66)

Ultimately it is difficult to read and locate items where the keyword occurs following suppressed material. Access and readability are compromised by having an arbitrary number of words appearing before the alphabetised keyword. The keyword, which is the word being sought by the user, may be located anywhere within the gloss string, either following the closing suppression character or in first position on the left-hand margin. Thus keywords - the points of entry to the finderlist - are *embedded* in the lookup entry rather than systematically 'fronted' for easy location. Increased length lessens the accessibility of keywords and the

readability of the final product, and in principle there is no upper limit to the length of a lookup entry.

3.2.2 Keyword fronting

Most pre-keyword suppression systems which are computer implemented insert parentheses into lookup finderlist entries on the basis of flagged keywords in source data in order to mark keyword positions for the user. However, the main function of keyword flagging is to provide a 'handle' for the computer sorting of finderlist items during automated finderlist production. Thus a suppression system conflates user keyword access and the computer sorting method. This conflation is a necessary part of any indexing system, but both user access and sorting are optimised by placing keywords in initial positions. This is *keyword fronting.*

Keyword fronting involves moving parts of the input gloss to the beginning of lookup entries before alphabetised sorting of the finderlist. A benefit of using keyword fronting is that semantic and historical (in comparative finderlists) data can become explicit as a result of placing together, via alphabetisation, items with shared keywords. When lexicographers specify keywords, they do so on the basis of the particular keyword's contribution to the semantics of the whole gloss string. So gathering together in the finderlist the like keywords, by selection and sorting, assembles sets of language items which share a degree of semantic composition (ignoring here the possibility of homonymy among keywords; see Section 3.2.3). (9) is an example from the Payungu Finderlist (Austin & Nathan 1989):

(9)

tree	**jurla (b), ngampu (a)**
tree, banksia	**thakanhungu**
tree, bark of	**thantha (a)**
tree, blackheart	**wirlukamparra**
tree, bloodwood	**kulijikuliji**
tree, bluebell	**pikurrka**
tree, branch of	**munhula (c)**
tree, cork	**pujalangu**
tree, fork in	**milka**
tree, ghost gum	**ngapuwarra**
tree, paperbark	**mirli**
tree, river gum	**kurrurtu**
tree, type of	**jarliri, jirtirti (b), karntuwa, kunyjarra, kurlurn, marntarru, martaru, minturnkura, minyjiliny, pinyjakunti, pujapirti, puntirrpa, thalpany, thamparli, wilharri, wurlany**
tree, wattle	**parrumpa**

There are several types of keyword fronting systems used by lexicographers. The simplest type copies the alphabetised keyword to the beginning of the lookup entry. The following extract from the index in Ross and Walker 1983 illustrates this:

(10)

rock (=exposed rock) 40
Rock Lobster (=Tropical Rock Lobster) 53
rock made smooth by water 40
rock which is smooth, hard and not easy to break 40
rocky area (=flat rocky area) 40
rocky ground 40

This type is discussed more fully in Section 3.2.3. In practice we usually find the embedded keyword omitted and replaced by a holder symbol such as a dash. This is shown in the following examples adapted from the *Yagaria Dictionary* (Renck 1977:306-7):

(11)

stockade	**lavi**
stockade, build a -	**lavi ao**
stomach	**agupa**
stomach, full -, have -	**amupa ao**
string game hitakota	**gaveda**
string game, play -	**hitakota gaveda hu**
string, hook - on bow	**havu ao gi**

Keyword fronting reduces the accessibility problem associated with pre-keyword suppression. However it is essential that readers be made aware of how a particular system works so that they know how to interpret the lookup entries they are reading. Unfortunately, lookup entries with complex structure and including characters with special function (here, the hyphen 'holder') can be difficult to read. Our readers are invited to try to reconstitute the lookup entry for "amupa ao" in the example above, and to work out the 'rules' underlying it without relying on native speaker expectations about the form.

In some cases, it is *necessary* to be able to reconstitute exactly an original source expression. Such cases include dictionaries of proverbs and quotations, where an entry can consist of a complete expression because it is an 'inventorized unit' like a word or idiom (Norrick *1985:2)*. Notice, in the following example entry from *The Oxford Dictionary of English Proverbs* (Wilson 1970), the use of the comma and the bar ("I") in indicating how the proverbs are to be reconstructed by the user (cf also (7)):

(12) Dog is drowning, When a | every one offers him a drink

The examples above show that a flagging system can become complex if it is to allow for the fronting of keywords without sacrificing readability. In particular, consider the problem of keeping words together. Usually only one keyword is fronted; but in some cases certain words are best kept together. This situation often occurs where the gloss which is the source of the lookup entry contains certain types of verbs, and a user cannot find the disambiguating context for the keyword(s) without venturing an arbitrary distance into the lookup string. Consider the following example from a hypothetical finderlist:

(13)

make.	To make bread	word-1
make.	To make off with	word-2
make.	To make someone happy	word-3

The fronted keyword for word-2 is semantically an unsatisfactory point of entry to the finderlist. To solve the problem a flagging system could be expanded to allow the indication of the number of words to be fronted with the keyword (cf Hsu and Peters' system (1984:26) and the SIL-AAIB system discussed in Section 3.2.4). This invites further problems in deciding which words to include. For example, in the (made up) example above, should the alphabetised keyword for "to make off with" be "make", "make off", or "make off with"? Similarly, should the alphabetised keyword for "to make someone happy" be "make happy"? Should the grammatical object of "to make bread" be a part of the keyword? Should (13) be reorganised as:

(14)

make bread.	To make bread	word-1
make happy.	To make someone happy	word-3
make off.	To make off with	word-2

In Section 4 we show how these questions are handled by computed parsing in our CGFL system.

3.2.3 Keyword classification

A third finderlist type involves explicit regrouping of lookup entries under keywords (for 'regrouping' see Section 3.1 and T. Crowley 1986:248). We found two versions of regrouping, one acceptable and effective, the other unacceptable.

Valid regrouping techniques extend 'fronting by repetition' shown above in (10). *Keyword classification* is like keyword fronting except that the lookup entries are classified under a keyword. A good example is Rehg and Sohl's *Ponapean-English Dictionary,* where finderlist nouns and verbs are listed under a single keyword classification (cf also Hansen and Hansen's *Pintupi Dictionary):*

(15) **cross**

cross or crucifix:	lohpwu	
cross or tough, of a person:	pwisinger	
to cross one's arms:	epidipe	
to cross over an obstacle:	kote	(from Rehg and Sohl 1979:150)

The purest example we found of this type is the finderlist in Fox's *Arosi Dictionary,* which simply lists lookup targets under a keyword, without any 'reorganisation' at all of lookup entries (cf Section 3.1). In the following excerpt (from Fox *1978:503),* the lookup targets (after the user locates them in the main dictionary) display a range of meanings similar to the lookup targets of (15); for example, we find in the dictionary the entry "horo (4)" with the definition "to cross over, lie across..".

(16) **cross** aharaul, ahoro 2, atohoro, bo'u 2, dadahoro 3, dadaro 1, ha'ahoro, hagu 7, haiharo, hauriogo, horo 4

More commonly systems of classification are wildly inconsistent. In these excerpts from the finderlist of S. Crowley's *Tolo Dictionary* (1986:91, 98), we find the lookup entries grouped and structured:

(17) shoot vanasia

shoot with gun	suda
shoot with spear	baloa
shoot out in steady stream (as urine)	rarasa
new shoot of taro, banana	kabona

wave

wave (to beckon someone)	kalopia
wave hand (to say hello or goodbye)	titutivua
wave (ocean)	lualua

Similarly, in Kilham et al. (1986), we find listed under the lookup entry "wet' both adjectives and transitive and intransitive verbs. But there are separate lookup entries for "spear" (to spear) (VERB) and "spear" (NOUN).

Finderlists such as shown in (17) try to deal with 'homographous' keywords (Hsu *1985:* 164f). These are keywords which are ambiguous because identical spellings can mask the distinction between a single word with related senses (polysemy), and separate words which have independent meanings or different grammatical categories but which have the same form (homonymy).

A format like (17) shows relationships which may not exist. It suggests a hierarchical relationship where "vanasia" is superordinate to "suda", "baloa", "rarasa" etc. In fact, this format (see also Hsu 1985:17 1, Hsu and Peters 1984:26) seems to conflate keyword classification (a way of presenting finderlist lookup entries) with the number of words in a source gloss. "Vanasia" is so placed in (17) because it has a single word English gloss, viz.

"shoot", whereas "suda" etc are glossed by English phrases. These facts are irrelevant to finderlist format.

In any case, the problem of homonymy and polysemy amongst keywords in a finderlist is really a non-problem. In (17) S. Crowley has tried to display,. through grouping and format, some semantic relations among the lookup entry keywords (viz "suda", "baloa", "rarasa' and "kabona" are claimed to be mutually related in a manner not shared by "kalopia" and "lualua". Similarly, "suda" and "kabona" are claimed to be related in the same way as "kalopi&' and "titutivua"). This is not correct. As Ianucci (1985:620 shows, speaking about bilingual dictionaries, multiple meanings of words in target language definitions are 'irrelevant to the entry'. In fact, it is the dictionary entry itself which 'serves as a kind of sense discrimination for the polysemous target word[s]'.

Thus, when dealing with finderlists, whose entries are derived from target language definitions, there is no justification for grouping senses because such information is simply not derivable from the primary lexicographic data. It may even be spurious, as seems to be the case in (17). Keywords capture a semantic *component* of the complete definition. Bilingual dictionary definitions from which lookup entries derive (see Section 2.2) are essentially metalinguistic (Singh 1982:130); they are semantic symbols which share form with a subset of English expressions. While such semantic symbols could be grouped, it would need to be on an explicit and principled basis, not in an ad hoc manner. However, if it is semantic structure to be displayed, it is that of the *source* language of the main bilingual dictionary. The main bilingual dictionary itself is, of course, exactly such a semantic description. The finderlist, by contrast, is a key to the main dictionary.

3.2.4 Computer implementation

Finderlist extraction systems are usually computer implemented by internally flagging the source gloss data of a bilingual dictionary (usually stored in a sequential file on magnetic tape or disk) with a special character to enable the correct finderlist extractions, for example (cf (5) above):

(18) +light +fire by friction

Computer programs using this system will be sensitive to a flagging control character (here "+"), and operate as follows:

a) for each gloss, control characters are iteratively searched for; and for each control character found:

b) the part of the gloss prior to the 'found' control character is placed within the suppression characters; and

c) the new string is output, with all control characters stripped out.

The details of subsequent sorting, addition of other information, and printing are ignored in this discussion.

This type of source gloss flagging is in widespread use by practicing lexicographers (see for example Hsu 1985, Hsu and Peters 1984). The simplest system requires a single flag to be embedded in the glosses (such as ÷ in the example above, or * in the *Lexware* system in use at the University of Hawaii (Hsu 1985). A more complex flagging system has been developed by the Summer Institute of Linguistics - Australian Aborigines and Islanders Branch.[4] It involves four types of keyword flags, both beginning and ending flags, as well as the possibility of coding for multi-word keywords and 'invisible keywords' (where the lookup

[4] We are grateful to David Nash for providing us with derails of the system together with examples.

entry string is sorted for a keyword which is not part of the source gloss itself). The system uses the following flags:

FLAG	FUNCTION
*	start of keyword
%	terminate keyword before end of word
=	extend keyword to include next word
(...)	invisible keyword

(19) sets out examples of the SIL-AAIB flagging system:

(19) SIL-AAIB GLOSS FLAGS

definition field	*unintelligible ***** speech
printed definition	unintelligible speech
keyword in index	unintelligible, speech
sub-entry in index	unintelligible speech
definition field	*grateful%ly
printed definition	gratefully
keyword in index	grateful
sub-entry in index	gratefully
definition field	un*grateful
printed definition	ungrateful
keyword in index	grateful
sub-entry in index	ungrateful
definition field	*un*grateful%ly
printed definition	ungratefully
keyword in index	ungrateful, grateful
sub-entry in index	ungratefully
definition field	to *hold tight%ly
printed definition	to hold tightly
keyword in index	hold, tight
sub-entry in index	to hold tightly
definition field	to *****fall=down hard
printed definition	to fall down hard
keyword in index	fall down
sub-entry in index	to fall down hard
definition field	(‘tooth) baby teeth
printed definition	baby teeth
keyword in index	tooth
sub-entry in index	baby teeth

Some of the flag types in (19) - especially the braces which delimit the ‘invisible’ keyword “tooth” - represent implementations of a kind of lemmatisation strategy. We do not hold to this strategy for finderlists. It is incompatible with the nature of finderlists (at least with the way we conceive them — see Section 2.2), where there is no licence to show a relationship between target language words. Nevertheless, we perhaps do want to show a relationship between words (i.e. lookup targets) which have meanings including a shared component ‘tooth [± plural]’. In general, our system will place items with only inflectional difference adjacent in the finderlist by alphabetic sorting, exploiting the fact that English, the glossing

language, has final grammatical inflection. Alphabetising words from the beginning tends to neutralise inflectional effects on order, just as a reverse-spelling dictionary or concordance is used to highlight it. Exceptions like 'teeth" can be treated as exceptions: their lookup entries can be manually written to an externally indexed file (see Section 4.1.2 and footnote 7) and used in finderlist production.

The suppression systems of Section 3.2.1 could perhaps be improved by computer flagging and/or selection of keywords, although we find no example of this in the literature we have examined. We have found that it would be possible to write programs which perform recognition/parsing to distinguish candidate finderlist keywords and so enable a (semi-) automated insertion of control characters. In the following sections of this paper we show not only that it is possible to computer generate sets of lookup entries from source dictionary glosses, but also that other advantages flow from further automating finderlist production. The main advantages are:

a) separation of primary lexicographic data from keyword specification;

b) labour savings; and

c) improved lookup entry readability through constant keyword fronting.

(c) is due to the fact that computer generation also enables grammatical transformation of gloss strings with fronting of relevant keywords for alphabetisation.

Despite the simplicity of all flagging systems for the computer extraction of finderlist lookup entries, they are ultimately problematic because they involve *corrupting* primary lexicographic data — the set of glosses for a lexicon — by the insertion of special characters which relate to a completely different function (viz, the output format of a finderlist). This issue is amplified in Section 5.

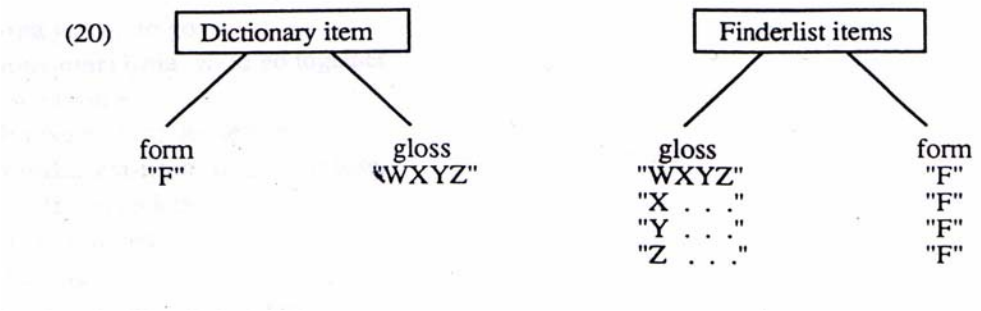
4. The Computer-Generated Finderlist (CGFL) system

4.1 Design

4.1.1 Design strategy

The CGFL system is the component of the reporting implementations of the GALP project which produces a finderlist for a language or languages whose lexicon(s) are stored in the GALP Database. Algorithmic methods are exploited as much as possible. The following sections outline the strategy for creating sets of lookup entries, which are alternate versions of word definitions or glosses. These sets are used by programs to produce finderlists. Each set of 'gloss-alternates' consists of expressions which alphabetise the lookup target under all of the desirable keywords. Each member of a set of retains the meaning of the source gloss, with the keyword fronted (Section 3.2) while retaining readability. The design strategy is schematised in (20).

Deliberately unspecified in (20) are the two aspects which represent the main complexity of the CGFL system. The first is: which of the gloss constituents "W", "X" etc are candidate keywords? For example, it would not be useful to have a language word (in **(20),** "F") which has a gloss "type of river gum" alphabetised under "of'. "Of' is not sufficiently semantically specified to be a point of entry to a lookup entry, and clearly this kind of gloss (with fronted "of') would proliferate, making the final product cluttered.



The second aspect is the configuration of the 'new' gloss alternates; that is, the problem left open by the use of ellipses in (20). The expressions "X...", "Y..." etc should be easily readable, while faithfully representing the semantic content of the original gloss (in (20), "WXYZ").

Fortunately these two aspects find common ground. Glosses written in English share the grammar of English, and various structures recur.[5] As noted by Hsu and Peters (1984:24), 'some of the phrases that most commonly precede...keywords, [are] "to", "to be", and "a kind *of*".' Although treated as a formatting problem by Hsu and Peters, this observation, appropriately extended, suggests that the identification of keywords and nonkeywords, and parsing of gloss expressions, can be performed algorithmically (cf. Nagao et al. 1982:63).

Within the GALP finderlist system, common words are listed, and made available to programs which produce the gloss alternates. Not only will the list (or 'stoplist'[6]) be able to inform the programs which words *not* to place at the front of lookup entries, but it will also be able to signal the presence of potential keywords because its listed words 'commonly precede keywords'. In other words, the list contains various function words of English, and these function words can drive an algorithm which identifies both the keywords and the grammatical structure of the gloss in which they occur.

4.1.2 Creating data for finderlists

The algorithm which produces sets of gloss alternates is set in the context of **a** data creation strategy. As described above, we need to create for each database gloss a set of corresponding lookup entry expressions. Since the keyword requirements, and the forms of the various lookup entries, are not explicitly part of the primary lexicographic data, they are new, or created data. They are the result of decision making on the part of the lexicographer.

The creation of finderlist data is a separate matter from its representation. How the created data is represented is a relatively arbitrary matter. In Section 3.2 we surveyed some finderlists and noted how the keywording requirements could be stored within dictionary glosses as system-specific flagging symbols. Another method would be to determine the keywording, and store the finderlist lookup entries fully and explicitly as character strings in a computer file. This finderlist source file would be logically external to the dictionary files, and would be indexed to the appropriate dictionary entry form.[7]

---

[5] This is not strictly true, since glossing seems to have its own style, for example "to call by kinship term". In fact, the simplifying structure of this glossing style is exploited in the GALP system under description. Nevertheless, glosses retain typical head-final phrases, which are key input structures to the GALP parsing rules.

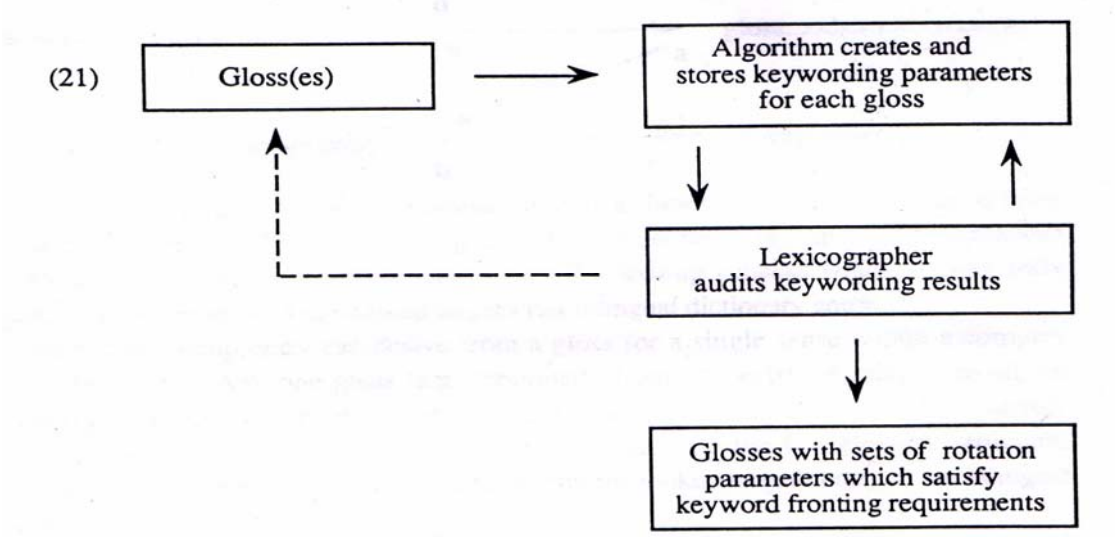
[6] This kind of 'blocking' list is called, in computing, a 'stoplist'. However note its double function in the CGFL as both a blocker' and also a signaller of phrase structure.

[7] Indexing of the external file would be by storing a string identical to the dictionary gloss together with the set of gloss-alternates. A finderlist program could then find the gloss-alternates by checking each dictionary item's gloss against the external file.

Alternatively, keywording requirements could be stored as the knowledge encoded in a finderlist-producing program including a set of 'smart' parsing rules. This program would use a stoplist (see Section 4.1.1) and a set of parsing rules to create gloss-alternates on-line as the finderlist is created.

Finally, there is the GALP data creation strategy which combines the use of parsing rules with the storage of parameters which identify the operation of particular rules. This is done prior to finderlist production: the parsing program works to create and store sets of symbolic parameters. These parameters subsequently apply together with the source glosses in finderlist output to produce the desired set of gloss-alternates (or 'rotations'). The sets of parameters correspond to the sets of gloss-alternates found as lookup entries in the final finderlist.

An on-line strategy was rejected because, despite the fact that the current GALP parser produces correct output in well over 95% of cases, we believe that lexicographers must audit all material coming out of their projects. If we allow computer programs to create some data, the lexicographer must be able to audit and authorise the results. Despite how much we refine the gloss-transforming algorithm, the lexicographer retains the final word (the precise way in which this is done is outlined in Section 4.4).[8]



4.1.3 Glosses in the GALP Project

The strategy outlined above exploits the nature of the glosses stored in the GALP Database. Glosses are almost always single words or simple phrases; that is, they do not consist of lists of words or phrases in explanations.[9] Each gloss expresses one kernel meaning; where the semantic range of a word is wider than can be handled in this way, a number of independent gloss strings are stored and then gathered together in dictionary production.[10] See Austin

[8] Readers oriented to syntax will note that this issue (the algorithm 'versus' the lexicographer), and two aspects discussed below (producing appropriate permutations 'versus' suppressing ill-formed permutations) reflect the tension between generation and constraint.

[9] Extra information about words is stored and displayed (in GALP dictionaries) in fields separate to the gloss, such as 'note', 'ethnographia etc.

[10] These 'extra' glosses are independent from the point of view of data storage and finderlist construction, but are ordered and formatted in the dictionary output programs to exhibit the semantic texture specified by the lexicographer.

(1989) for details of gloss storage. (22) illustrates the semantic texture, or levels of semantic analysis, allowed for:

(22) a. Form-1 1. Gloss-1
2. Gloss-2 etc.

b. Form-1 (a) Gloss-3, (b) Gloss-4 etc.

c. Form-1 Gloss-5, Gloss-6 etc.

Here (22a) is an example of homonymy, (22b) shows the representation of polysemy, and (22c) shows a case of gloss synonymy or gloss equivalence (the latter is used where there is no difference in denotational meaning, but rather a difference between the gloss expressions themselves in style or usage). Any combination of gloss relations is possible, e.g.

(23) Form-1 1. Gloss-1
2. (a) Gloss-2, Gloss-3, (b) Gloss-4
3. Gloss-5, Gloss-6
etc.

4 .2 Description of the core CGFL algorithm

4.2.1 Parsing the gloss expressions

At the core of the CGFL system is a recogniser/parser algorithm which has as its input GALP database glosses. The algorithm produces a set of parameters for each gloss, computed as a function of the structure and lexical content of the gloss. Each parameter corresponds to a successful matching of the gloss structure to a structural template and a corresponding predefined transformation rule which can apply to the 'recognised' structure and permute the words. Each rule type and its resultant permutation is called in our project a *rotation.* When a parameter's rule is applied to a gloss (by the reconstituter algorithm, see Section 4.2.2) the words in the gloss are permuted, producing a string which is a variant of the original gloss but with a fronted keyword. We allow for two basic types of rotation:

a) plain rotation, where a keyword is fronted by splitting the gloss in two, without regard to the internal structure of the gloss, and putting the second part first; and

b) contextual rotation, where complex permutations front keywords which are part of phrases within the whole gloss, by fronting the entire phrase, and then fronting the keyword by permuting the phrase 'locally'.

The algorithm does no backtracking, and simply reads through the gloss string word by word. As it does so, it may do some or none of four operations, where a) and b) are only applicable to contextual rotations:

a) set a flag which places the algorithm into a phrase parsing mode;

b) if in phrase parsing mode, label the currently read word with a system symbol;

c) send the gloss and a rotation rule type to the reconstituter,

d) append the register of rotation parameters for this gloss.

The operation of the algorithm is based around a stoplist lexicon of approximately 50 English function words, consisting mostly of prepositions, (possessive) pronouns and articles (see Section 4.1.1 and Appendix 1). Such words are not candidate keywords, so that a currently read stoplist word causes no action. On the other hand, words which need to be fronted as keywords are generally heads of noun or verb phrases occurring in final position of the

phrase. Since the function words are good indicators of the boundaries of phrases, the presence of stoplist words sets off a parsing sequence.

If the current word being read by the parser is found within the stoplist (we then call it a pivot word for that gloss), two actions occur: firstly, the plain, or context-less rotation which fronts the current word by splitting the gloss in two without regard to the internal structure of the gloss (see (24) below) is suppressed, and secondly, a flag is set to produce a 'top-down' parsing where the words following or preceding the pivot word are parsed as potential phrases ('pivot phrases'). Each subsequent or preceding word (up to a maximum of two preceding, three subsequent, where the third 'word' is actually the third word plus any remaining words in the gloss to ensure that the gloss is not truncated) is then stored and labelled in a register so that the reconstituter can put the gloss together again to show the effect of assigning the parameters.

We treat pivot words as linguistic operators which have scope over leftwards- and rightwards-adjacent content words. These content words are generally phrases which need to be kept together in the reorganisation of the gloss. Such assigned phrases are then *internally* permuted if necessary to satisfy the keyword fronting requirements. Further parsing is subject to continual checking against the stoplist lexicon so that the presence of subsequent function words will cause appropriate adjustment of the structural assignment.

Each generated rotation parameter corresponds to one fronted keyword version of the gloss expressed according to system-defined rules of structure. The factors influencing the assignment of parameters are:

a) the number of words occurring before and after a pivot word;
b) the number of words occurring before and after a 'current' word (a 'current' word is fronted to keyword position in output);
c) the presence in a gloss of words which are listed in a system stoplist ('pivot' words);
d) the special value(s) marked on the words occurring in the stoplist, which license internal parsing of 'pivot phrases';
e) whether another pivot (i.e. stoplist) word appears within the gloss following the assigned pivot word; and
f) the presence in a verb gloss of one of a small set of intransitive verbs (see Section 4.3.1)

Factors (a - e) can be understood through a study of the permutations they are associated with, shown in (24) below.

After each gloss has been processed, two types of data have been created: a set of rotation parameters (each represented by an arbitrarily chosen integer in the range 1 - 9), and a permuted (rotated) version of the gloss corresponding to each parameter. These two types of data are handled differently. The rotated gloss strings are 'virtual, computed 'on-line' by the reconstituter (Section 4.2.2) as the program runs and may be output to a screen, printer or a disk file. However the parameter register is maintained throughout the processing of a particular gloss, and at the end of that processing, the parameter register is available to be stored in the GALP database, output to an external file, or used by other programs (for example, the interactive Comparator described in Section 4.4).

It is the computation of the new parameter data, enabling the lexicographer to subsequently audit the assigned rotations before actual finderlist production, which is the main goal of this phase of the CGFL system. Programs have been designed to run in three modes:

a) in batch mode, ranging over the entire set of database glosses, generating and storing parameters and producing a report;

b) in batch mode, ranging over the entire set of database glosses, generating but not storing parameters, and producing a report showing existing database rotation parameters (if any), comparing them to the parameters which the system produces; and
c) in an interactive mode, using the Comparator, see Section 4.4.

4.2.2 Reconstituting rotation strings

The 'reconstitution' algorithm serves two phases of the CGFL system. **In the parsing** or parameter-generation phase, described above, it builds gloss alternates to provide an explicit, integrated report of parsing. But the main operation of the reconstituter is in the output phase, which extracts primary data and produces a finderlist.

In the parameter-generating phase, the reconstitution algorithm is supplied with the following by the recogniser-parser algorithm:

a) an input gloss string;
b) a rotation parameter code; and
c) the position of the 'current' word within the gloss string which is to be a fronted keyword.

In the output phase, the production of rotated glosses for a finderlist will work in the same way, except that only a) and b) will be available to the program, and reference will be made to the system's stoplist to check that the 'currency' computed from a) and b) produces a possible keyword.

(24) below is a schema of rotation types. While indicating the permutation associated with each rotation type and parameter, it does not exactly represent the workings of the algorithm in either method or completeness. Recognition/parsing is in fact interleaved with reconstitution in the parameter generating program, so that certain segmentations are supplied, labelled, direct to the 'reconstituter'. This is done because full permuted strings are always produced in the parameter-generating phase so that an explicit report is produced showing the new parameters together with the exact gloss permutation which would be produced by applying the parameters in the output phase (see Appendix 2 for samples). Also the category 'pivot' actually identifies the *first* sequence (one or more, contiguously) of stoplist words in a gloss. For example, some word sequences (written in (24) as **"word*")** may also include a non-initial 'pivot' word. There are extra restrictions on the '9' rotation, which cannot apply if a) word 1 is in the system-defined set of intransitive verbs (see Section 4.3.1) and if b) the gloss is nominal and word*2 is null. We also use a surface filtering device which checks all output strings and strips off initial articles and 'of's, and all trailing articles.

(24) a. 'PLAIN' ROTATION (CODE = POSITION OF CURRENT -1)
word* 1 current (word*2)
→* current (word*2) ',' word"1 [NOUNS]
→* current (word*2) ", to" word*1 [VERBS]
Examples: wedge-tailed eagle → eagle, wedge-tailed [N]
paint in coloured stripes → coloured stripes, to paint in [V]

b. '6' ROTATION
word* 1 pivot* 1 word2 current pivot*2 (word*3)
→* current "," word2 "," pivot*2 (word*3) "," word'1 pivot*1 [NOUNS]
→* current **","** word2 "," pivot*2 (word*3) ", to" word*1 pivot*1 [VERBS]
Examples: song of ancient heroes and villains
→heroes, ancient, and villains, song of [N]
paint in coloured stripes and circles →* stripes, coloured, and circles, to paint in [VI

c. '7' ROTATION

word*1 pivot* word2 current (word*3)
→* current (word*3) ,“ word2 ‘,“ word*1 pivot* [NOUNS]
→* current (word*3) “,“ word2 “, to” word*1 pivot* [VERBS]
Examples: pathway to initiation ground →ground, initiation, pathway to [N] make a repetitive clicking sound →*clicking sound, repetitive, to make [V]

d. ‘8’ ROTATION
word* 1 pivot* word2 word3 current (word*4)
→* current (word*4) “,“ word2 word3 “,“ word” 1 pivot* [NOUNS]
→ current (word*4) ‘,“ word2 word3 “, to” word*1 pivot* [VERBS]
Examples: small bone in leg of kangaroo →. kangaroo, leg of, small bone in [N] paint in different coloured stripes →stripes, different coloured, to paint in [V]

e. ‘9’ ROTATION
word1 current pivot (word*2)
→current ‘,“ word1 “,“ pivot (word*2) [NOUNS]
→current “, to” word1 pivot (word*2) [VERBS]
Examples: lower colours of rainbow → colours, lower, of rainbow [N]
put fat on body →* fat, to put on body [V]

The first line for each rotation type in (24) shows the parsed template of an input gloss produced by the recogniser/parser. It is solely the ability of the recogniser/parser to match a template to an input gloss which licenses assignment of the relevant parameter. The next lines show the rearrangement of the input constituents by the reconstituter. Reconstitution is sensitive to the syntactic category of the input gloss (which is supplied to the programs by a lookup of the syntactic category of the gloss’s entry in the GALP database). New material inserted by the reconstituter is shown within quotation marks — only commas and the infinitive marker “to” are allowed, and nothing may be deleted (except initial and final function words, in the post-reconstitution or ‘surface’ filter). The asterisk is like the Kleene-star operator in syntactic literature; it denotes a contiguous sequence of any number greater than zero of the category to which it is attached. Parentheses indicate material which is optionally present and does not affect the operation of the algorithm. The numbers found at the ends of constituents have no status, and are used for clarity to index the identity of constituents on either side of the ‘rotational arrow’.

The titles given to the rotation types (except ‘plain’) also have no numerical or mnemonic significance, and are written here as they are used by a numbering system within the programs.

‘Current’ is always a word which has been selected by the recognition/parser to be an eligible keyword. Thus, it does not appear in the stoplist. Note that the ‘plain’ rotation simply splits the gloss in two and inverts the halves. The other types are *contextual:* they depend on the presence of a stoplist word in a structural context. Only a subset of stoplist words are marked (in the stoplist itself) to license these contextual rotations.

4 .3 Discussion: examples in the development of the CGFL algorithms

4.3.1 Complex verb glosses

Several glosses of the GALP database include complex verbs of the type verb + particle. In simple cases where the particle follows the verb, rotation can conveniently occur following the particle (a ‘pivot’ word), fronting the next word, according to the descriptions in (24) above. A following noun phrase is kept together:

(25) peel off outer skin → outer skin, to peel off (parameter code: plain2)

peel off outer skin → skin, outer, to peel off (code: context7)

(The latter is a '7' rotation. The infinitive "to"s are inserted by the reconstituter). The particle, here "off", appears fortuitously in the correct position in the output even though it is not really part of a pivot phrase ("off outer skin") but rather the particle of the complex verb 'peel off.

In other cases the source gloss has a verb and particle separated by a noun, where the noun is the grammatical object of the complex verb. When the noun is fronted, the verb and its particle become adjacent:

(26) pull splinter out → splinter, to pull out (code: context9)

(26) is a result of the '9' rotation, which inverts the gloss-initial two words when the third word is a pivot. The algorithm is sensitive to the gloss's syntactic category, inserting "to" and also leaving out the comma which would be inserted into a similarly structured nominal gloss. A different situation occurs with verb glosses which are intransitive, especially those containing verbs such as "be", "become", "have" and "go". A gloss such as "be upset about something" does not read well in the following fronting of "upset":

(27) upset, to be about something (a '9' rotation)

This is improved by treating the source gloss as a stative verb + adjective phrase, that is, as "be + upset about something", and rendered as a plain rotation by suppressing contextual rotation (see Section 4.4 for method of suppression):

(28) upset about something, to be

'State' glosses are difficult to handle, and it can be better to change or paraphrase the input gloss. For example, "be in doubt" does not give a good fronting of "doubt" (viz. "doubt, to be in"); however recasting it as "be doubtful" gives the rotation "doubtful, to be".

4.3.2 Changing the database glosses

Within the finderlist format any particular gloss is functionally equivalent to any of its rotated permutations. Therefore, input, or GALP database, glosses are not restricted to any particular structure or wording. In fact database glosses can be constructed with the aim of providing the best possible set of alphabetised rotation strings (as in the case of "be doubtful" above). Of course the meaning of the gloss must be held constant.

Some situations are solved by adding hyphens to the source gloss to block rotation generation. Note, however, that we do ***not*** license 'metafunctional' flags — hyphens must be readable as a normal part of the gloss. For example, the rules described in (24) would provide spurious alternates for "black and white fantail":

(29) black and white fantail →* black and white fantail (code: null)
→ white fantail, black and (code: plain2)
→ fantail, white, black and (code: context7)

This can be solved by recasting the gloss as "black-and-white fantail":

(30) black-and-white fantail →* black-and-white fantail (code: null)
→* fantail, black-and-white (code: plain 1)

While the changing of database glosses to facilitate keyword rotations is catered for in the CGFL system, it is not systematically required and is of minor importance in finderlist production. It is only used for the minority of source glosses which are not well parsed by the system (perhaps 1 or 2 per cent of current glosses). And secondly, it is not crucial to the success of the system because refractory glosses can be handled as 'exceptions', and stored with their manually produced set of lookup entries in a separate 'external file'. Refer to Section 4.1.2 and footnote 7.

4.4 The interactive comparator

4.4.1 Purpose

Previous sections have described a system for producing sets of gloss alternates which is the main process involved in making finderlists. The system is creative: it attempts to make new expressions using the words supplied to it. And, as pointed out in Sections 4.1.1 and 4.1.2, the lexicographer must be able to control the created output of the computer programs. We have mentioned three ways in which gloss alternate production is controlled:

a) in the planning, writing and testing of the rules underlying gloss alternate production. The rules are illustrated in (24), and some of the factors influencing their derivation have been discussed in Section 4.3.1. This phase of the project is bounded — the rules may be revised, but are considered to be complete at any stage

b) by deleting parameters specifying ill-formed gloss alternates, as a result of auditing the special report produced by the recogniser/parser showing each input gloss together with its generated set of gloss alternates and their system rotation codes (see (21), and Appendix 2 for a sample report format)

c) by changing glosses to achieve a better gloss alternate set, discussed in Sections 4.3.1 and 4.3.2. Note that glosses are *not* changed with respect to the operation of rules (the lexicographer can ignore the details of rule operation), but with respect to the relation between an input gloss and the appropriacy of its set of gloss alternates (21).

The final phase in building the CGFL system was the design and implementation of an interactive program, called the Comparator, which the lexicographer uses to perform b) and c) above. The comparator's main function is to provide the user-interactive tool for the deletion of ill-formed computer-generated rotation options. This is done by indicating the appropriate rotation code(s) to be deleted. The comparator's secondary function allows the lexicographer to interactively create new glosses (or modify those existing in the database) and observe the effects of automated rotation. The effects on new or modified glosses can be checked, and then the glosses can be stored in the GALP database or discarded, as desired. This is a powerful and useful function which optimises the ability of the 'system' (i.e. software + lexicographer, as presented in (21)) to produce a full and readable set of keyworded gloss alternates for each gloss. Finally, the comparator also 'marries' the parameter-generating algorithm and the parameter-driven output reconstitution algorithm, to provide rigorous testing of consistency.

4.4.2 Method

The comparator is run with the GALP database in modify-enabled mode. The lexicographer is prompted for an input string, which is treated as a potential database gloss. A message is displayed indicating whether or not the string is actually an existing database gloss. If it is, the session involving that gloss makes available at the terminal both the currently-stored database rotation parameters for that gloss and also the parameters generated in real time by the rotation algorithm. As well as this, the comparator can display a set of rotations chosen by the user, by selection from the computer-generated set. In this way, the lexicographer can delete parameters for ill-formed gloss alternates, check the revised lookup entry set, and then store the results permanently in the GALP database.[11] Thus the comparator can display any set of gloss- alternates specified by the user, and compare it with any other set, whether a differently specified set, the set already occurring in the database, or the default algorithm-generated set. When the user is satisfied, current database parameters may be overwritten using the new parameter set. Alternatively, if a new or modified gloss is written to optimise the rotation set, this gloss and its audited parameter set can be stored in the database.

[11] The user may also add ungenerated rotation codes to the set (subject to error conditions), and their corresponding strings will be displayed.

5. Computing and lexicography: aspects of data and processing

5 1 Introduction

In the foregoing we have restricted the discussion to the linguistic and practical aspects of producing automated finderlists. Specific logical aspects of the CGFL system, for example the core rotation algorithm, are independent of the actual implementation; they would work as 'manual' rules for a stage of finderlist production done by lexicographic assistants. However it was mentioned in Section 3 that there are principles of computing which ought to inform and constrain an automated dictionary project. In this final section we mention some of these principles, and their implications for approaches to finderlist production. The discussion comes under three headings: data integrity, data independence, and algorithmic implementation.

5.2 Data integrity

Data integrity is an important concept for lexicographers, who typically deal with large amounts of arbitrary (i.e. factual, real world) data. The first step in maintaining the integrity of data is to define what is to be protected. In most electronic storage, data is stored in 'fields', or addressable units of data. In computing, data fields should be atomic units, that is, their possible values are to be drawn from a single logical domain of data type, corresponding to entities of the real world (Date 1977:10; Date 1986: 145ff). Glosses correspond to meanings, which occur in the real world, and finderlists also occur, but these are different logical domains and a single informational unit ought not represent them both.[12] This is one reason why lexicographic data in electronic medium should not have inserted flags. any large and complex store of information. In the CGFL system we treat lookup entry sets as 'virtual data' — they have 'no single stored counterpart; instead...values [are] materialised by means of...computation performed on a set of stored field occurrences' (Date 1977:12). In other words, GALP glosses are stored separately from rotation parameters and gloss-alternates are computed. This eliminates a common source of data corruption, namely mis-typing (Laughren and Nash 1983:127). There are risks involved in the input or revision of flags, since it involves altering primary data. The risk is highest where the gloss strings are manually edited to insert the flags, although algorithmic or software-controlled flag insertion is possible. [13]

5.3 Data independence

One of the goals of computer systems is data independence. This means that data ought to be independent of the computing processes they undergo. The 'corruption' of glosses with processing flags mentioned in Sections 3.2 and *5.2* has disadvantages beyond the reduction of data integrity.

The use of flags entails *interdependence* of lexicographic data and the programs which deal with them. Programmers will have to provide programs which 'know about' the special flags, and ensure strict compatibility between the flags and the programs which deal with the glosses. These programs will use flags to make finderlists and will need to edit out the flags to produce clean output. If the control characters are modified (or extra ones introduced), then programs will have to be reworked. There may be a number of programs using the gloss data. At least one will handle the extraction of a standard dictionary, for which the internal structure of individual glosses (and thus the control characters) will be irrelevant. This

[12] The CGFL system uses the lexicogrammatical structure of glosses to drive the finderlist algorithm, but this structure is data inherent in and identical with the actual gloss itself. Parameters, flags, and lexicographers choices etc are a different matter.

[13] For example by limiting access to the data by presenting the program user a 'menu' of word-positions to choose from, and having the program insert, or delete, the flags.

program will need to be written to remove parts of the data (the control flags). Interdependence also compounds the task of isolating and solving the effects of errors and 'bugs', and increases the difficulties of program maintenance and updating, due to cascading tasks, or the 'ripple effect' (Weinberg 1980:31).

Modern computing practice requires that data be logically separated from the way it is seen or used. This ensures that within a project, different programs and users can 'view' the data in the most appropriate or economical way. Conversely, a project redesign, where the application methodology is revised, will not require modification of primary data. For example, the GALP project has two major sub-systems: (a) a lexicon system, and (b) a sentence system. The sentence system relates sequences of lexicon items to free (English) glosses. The subsystems are integrated through the sharing of data (which enhances data storage characteristics and research potential). However the computer programs which run each system are independent, using the same data in different ways, using different strategies, and were written by different people in different programming languages.

Data independence also means that data can be maximally portable between projects (a useful benefit to working linguists and lexicographers) without being dependent on particular programs.

5.4 Algorithmic implementation

We use the term 'algorithmic implementation' to express the advantages of exploiting an automated system. In the CGFL, lookup entries are 'virtual' (see above), requiring significant computation, so that it is, in practice, the exploitation of computational power which enables the system to work. But there are other advantages: because the system uses a database where each unique data identity is stored only once and all input and output is via software, consistency checks are made during input, and in output, for example the production of a dictionary file, all lexical relations are symmetrically specified and all cross references are made. Consistency is not just achieved, it is a by-product of the overall project design.

Consider the kind of problems which arise for finderlists within more primitive designs. If a particular gloss exists more than once, there is no mechanism to ensure that its keyword flaggings are consistent. And an unflagged gloss in a flagging system is ambiguous; it is impossible to tell whether it is specified as having no finderlist keywords, or if the keywords are not yet flagged (unless some logging of work takes place). By contrast, in the CGFL system, the finderlist parameter-generation phase program is run over the entire database, producing an integrated report of its work. And later, when the researcher uses the Comparator to deal with a small number of unsatisfactory gloss-alternates, the program produces a special log of changes made.

Finally, there is a considerable saving of time and effort in producing a finderlist using a CGFL method. In the flagging systems described in Section 3.2, all of the keywordings must be specified and stored as flags. The output of the program which translates flagged data into a finderlist must be checked by the lexicographer for correct operation. However, in a CGFL system, the keywordings are computer generated - or, in fact, slightly overgenerated. The lexicographer checks the special system report showing the generated keywordings, and then uses the Comparator to fine tune the results. In other words, the lexicographer has to do *all* the work in a flagging system, but only works *subtractively* in a CGFL system to remove or repair a small number of ill-formed keywordings.

5.5 A hardware approach?

Twenty-seven years ago, Alan Pence, in his article *Punched card filing for linguists,* wrote:

> The equipment needed to get the system into operation consists of: punched cards, sorting needle, filing boxes, and clipper ... cards may be drilled by hand using a 1/2

> inch steel template and an 1/8 inch drill. Clipping may be done with scissors, and a knitting needle or a piece of straight wire may be used for sorting.(Pence 1962:77).

In Pence's system, traditional data cards are supplemented by holes and notches representing analysis of the data written on the card. While the implementation is technologically outmoded, it exemplifies some of the points made in this Section. Pence's source data (written on cards) is kept logically separate from drilled and punched created data. The separation of written data from physically coded analysis reflects the different domains of data and process, paralleling the GALP distinction between source database glosses and the parameters computed to produce gloss alternates and thus finderlists.

We hope this example shows the importance of defining relationships between data representation, data creation, and processing. There are improvements to be made to the CGFL system: but it embodies seeding principles and strategies. On the other hand, claims that typed- in flagging systems 'enable a usable finderlist to be automatically generated from material already in the definitions' (Hsu 1985:185) are clearly false. They exploit computer storage and processing only to reduce word-processing tasks.

REFERENCES

Austin, P. 1983. Southern Pilbara dictionaries. In Austin, P. (ed.) *Papers in Australian Linguistics* 15: 1-17. Pacific Linguistics A66.

Austin, P. 1988. Classification of Southern Pilbara languages. *Pacific Linguistics* A71: 1-17.

Austin, Peter. 1989. Dictionaries and concordances - application of a database model. La Trobe University. MS.

Austin, P. & Nathan, D. 1989. A dictionary of Payungu, Western Australia. La Trobe University. MS.

Butler, C. 1985. *Computers in linguistics.* Oxford: Blackwell.

Crowley, S. 1986. *Tolo dictionary.* Pacific Linguistics C91.

Crowley, T. 1986. *Dictionary making: course books 1 and* 2. Suva: University of the South Pacific.

Date, C. 1. 1977. *An introduction to database systems.* Reading, Mass: Addison-Wesley.

Date, C. J. 1986. *Relational database: selected writings.* Reading, Mass: Addison-Wesley.

Fox, C. 1978. *Arosi dictionary.* Pacific Linguistics C57.

Hall, H. 1971. *A partial vocabulary of the Ngalooma Aboriginal tribe.* Canberra: AIAS.

Hansen, K. & Hansen, L. 1974. *Pintupi dictionary.* Darwin: Sit.

Hartmann, K. 1983. *Lexicography: principles and practice.* London: Academic Press.

Hsu, R. 1985. *Lexware manual.* Honolulu: University of Hawaii.

Hsu, R. & Peters, A. 1984. Computers and Micronesian dictionaries: a chronicle of systems-design and fieldwork among lexicographers. In Byron Bender (ed.) *Studies in Micronesian linguistics,* Pacific Linguistics C80: 1-36.

Ianucci, J. 1985. Sense discriminations and translation complements in bilingual dictionaries. *Dictionaries* 7: 57-65.

Kari, J. 1989. Notes on the computerization of the Ahtna dictionary. MS.

Kent, W. 1978. *Data and reality.* Amsterdam: North-Holland.

Kilham, Christine, Mabel Pamulkan, Jennifer Pootchemunka and Topsy Wolmby. 1986. *Dictionary and Source Book of the Wik-Mungkan Language.* Darwin: Sit.

Landau, S. 1984. *Dictionaries: the art and craft of lexicography.* New York: Scribner.

Laughren, M. & Nash, D. 1983. Warlpiri dictionary project: aims, method, organization and problems of definition. In P. Austin (ed.) *Papers in Australian Linguistics* 15: 109-33. Pacific Linguistics A66.

Linger, H. 1989. Relational support for a linguistic database. Chisholm Institute of Technology. MS.

Nagao, M., Tsujii, J., Ueda, Y. & Takiyama, M. 1982. An attempt to computerise dictionary databases. In J. Goetschalckx and L. Rolling (eds). *Lexicography in the electronic age,* 51-73. Amsterdam: North Holland.

Norrick, N. 1985. *How proverbs mean.* Berlin: Mouton.

Pence, A. 1962. Punched card filing for linguists. *Oceania Linguistic Monographs* 6: 76-89.

Rehg, K., & Sohl, D. 1979. *Ponapean-English dictionary.* Honolulu: The University Press of Hawaii.

Renck, G. 1977. *Yagaria dictionary.* Pacific Linguistics C37.

Ross, J. & Walker, A. 1983. *Gumatj wordlist: part 1.* Working papers in theoretical and applied linguistics. Darwin: University Planning Authority.

Scott, G. 1980. *Fore dictionary.* Pacific Linguistics C62.

Singh, R. 1982. *An introduction to lexicography.* CIIL Occasional Monograph 26. Mysore:

Central Institute of Indian Languages.

Tilley, M. 1966. *A dictionary of the proverbs in England in the sixteenth and seventeenth centuries.* Ann Arbor: The University of Michigan Press.

Weinberg, V. 1980. *Structured Analysis.* Englewood Cliffs NJ: Prentice-Hall.

Wilson, F. (ed.) 1970. *The Oxford dictionary of English proverbs* (3rd edition). Oxford: Oxford University Press.

Zgusta, L. 1971. *Manual of lexicography.* The Hague: Mouton.

APPENDIX I

SYSTEM STOPLIST

"under":	"context"	"other":	"yes"	"from":	"context"
"towards":	"yes"	"others":	"yes"	into":	"context"
"before:	"yes"	"each":	"yes"	"after":	"yes"
"against":	"yes"	"over":	"yes"	"by":	"context"
"onto":	"context"	"your":	"yes"	"the":	"context"
"through":	"yes"	"our":	"yes"	"at":	"yes"
"forth":	"yes"	"his":	"yes"	"to":	"context"
"it":	"yes"	"their":	"yes"	"with":	"context"
"is":	"yes"	"her":	"yes"	"of":	"context"
"used":	"yes"	"my":	"yes"	"a":	"context"
"along":	"yes"	"down":	"yes"	"an":	"context"
"about":	"yes"	"or":	"yes"	"in":	"context"
"and":	"yes"	"one":	"yes	"on":	"context"
"around:	"yes"	"ones":	"yes"	"up":	"yes"
"ago":	"yes"	"ones":	"yes"	"for":	"yes"
"another":	"yes"	"oneself":	"yes"	"off":	"yes"
				"out":	"yes"

APPENDIX 2

SAMPLE REPORT OF PARSING SOME SELECTED GLOSSES

```
pathway     connecting two rings of initiation ground (H)              01235      00007
            connecting two rings of initiation ground1 pathway (1)
            two rings of initiation ground, pathway connecting (2)
            rings of initiation ground, pathway connecting two (3)
            initiation ground, pathway connecting two rings of (5)
            ground, initiation, pathway connecting two rings of (7)
become mixed up (U)                                                    0001       0000
            mixed up, to become
wild potato of double gee (N)                                          00013      00097
            wild potato of double gee (H)
            potato of double gee, wild (1)
            potato, wild, of double gee (9)
            double gee, wild potato of (3)
            gee, double, wild potato of (7)
Sound of chopping with axe (N)                                         00002      00008
            chopping with axe, sound of (2)
            axe, chopping with, sound of (8)
            % More spec contexts: first only used
```

edible gum of snakewood tree (N)	00013	00097
gum of snakewood tree, edible (I)		
gum, edible, of snakewood tree (9)		
snakewood tree, edible gum of (3)		
tree, snakewood, edible gum of (7)		
small bone in leg of kangaroo (N)	00013	00098
bone in leg of kangaroo, small (1)		
bone, small, in leg of kangaroo (9)		
leg of kangaroo, small bone in (3)		
kangaroo, leg of, small bone in (8)		
Z More spec contexts: first only used		
small intestine of sheep (N)	00013	00009
intestine of sheep, small (1)		
intestine, small, of sheep (9)		
sheep, small intestine of (3)		
pair of principal actor and initiand (N)	00025	00006
principal actor and initiand, pair of (2)		
actor, principal, and initiand, pair of (6)		
initiand, pair of principal actor and (5)		
make a clicking noise (U)	00002	00007
clicking noise, to make (2)		
noise, clicking, to make (7)		
make a repetitive clicking noise (U)	00002	00078
repetitive clicking noise, to make (2)		
clicking noise, repetitive, to make (7)		
noise, repetitive clicking, to make (8)		
younger brother or sister (N)	00013	00009
brother or sister, younger (1)		
brother, younger, or sister (9)		
sister, younger brother or (3)		
investigate with a stick (U)	00003	00000
stick, to investigate with (3)		
edible gum of cassia tree (N)	00013	00097
gum of cassia tree, edible (1)		
gum, edible, of cassia tree (9)		
cassia tree, edible gum of (3)		
tree, cassia, edible gum of (7)		
principal actor in mans initiation (N)	00013	00097
actor in mans Initiation, principal (1)		
actor, principal, in mans initiation (9)		
mans initiation, principal actor in (3)		
initiation, mans, principal actor in (7)		
kill one another by magic (U)	00004	00000
magic, to kill one another by (4)		
wedge-tailed eagle (H)	00001	00000
eagle, wedge-tailed (1)		
gang up on (U)	00000	00000

APPENDIX 3

SAMPLE DIALOGUE WITH THE COMPARATOR

(user responses to prompts are given in bold capitals)

```
Enter gloss to test ("ex" to exit):                    HAVE EYE TROUBLE
Enter category:       U

have eye trouble      U                 00012         00000
  Gloss in database (14168)
     have eye trouble (plain)
     eye trouble, to have (1)
     trouble, to have eye (2)
  These codes are the same as the current database codes.

You may now:
  Quit this gloss,
  Store these rotations,
  Delete or Add rotations,
  View current database rotations, or
  Reconstruct rotations from current rotation codes.

  Enter your choice: Q

  Enter gloss to test ("ex" to exit):                  HAVE TROUBLE WITH EYES
  Enter category:    U

have trouble with eyes (U)              00013         00009
  Gloss not in database
     have trouble with eyes (plain)
     trouble with eyes, to have (1)
     trouble, to have with eyes (9)
     eyes, to have trouble with (3)

You may now:
  Quit this gloss,
  Store these rotations,
  Delete or Add rotations,
  View current database rotations, or
  Reconstruct rotations from current rotation codes.

Enter your choice:     DEL

Enter digit code for delete rotation:                  9

Your current rotation codes: 13 0
Rotations from your current codes: 13 0
  ==>   have trouble with eyes (plain)
  ==>   eyes, to have trouble with (3)
  ==>   trouble with eyes, to have (1)

You may now:
  Quit this gloss,
  Store these rotations,
  Delete or Add rotations,
  View current database rotations, or
  Reconstruct rot at ions from current rotation codes.

Enter your choice:    Q
```