Order Number 9300331

A computational model for the testing of linguistic hypotheses concerning language change

Lindsey, Francis Lynn, Jr., Ph.D.

University of Hawaii, 1992

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

A COMPUTATIONAL MODEL

FOR THE TESTING OF LINGUISTIC HYPOTHESES

CONCERNING LANGUAGE CHANGE

A DISSERTATION SUBMITTED TO THE GRADUATE DIVISION OF THE
UNIVERSITY OF HAWAII IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

LINGUISTICS

August 1992

By

Francis Lynn Lindsey, Jr.

Dissertation Committee:

Stanley Starosta, Chairperson
David Watson
Anatole Lyovin
David Stampe
Robert Cheng
William O'Grady

ACKNOWLEDGEMENTS

I have been studying linguistics for over 20 years. During that time I have been fortunate in having a number of excellent scholars as my teachers. To them I owe a major debt for the ideas in this study.

The work on this dissertation was continually hindered by and occasionally halted by circumstances irrelevant to scholarship. The Starosta family have been extremely kind in helping me to find ways to solve many of those problems. In addition, the group of linguistics professors, students, and friends that meet weekly at Mama Mia's pizza shop have been wonderful in their support and friendship, without which I would never have finished. Then, during the final hectic stages of writing and defending this dissertation, my colleagues in the English Department at Ramkhamhaeng University provided me with more support of various sorts than should be reasonably expected from friends. The assistance of all these people is greatly appreciated.

As the originator of the lexicase grammatical theory, Dr. Stanley Starosta introduced many of the ideas found in this work. As my committee chairman and a scholar, he has been appropriately critical of my inadequate presentation. And as a friend, he has been extremely supportive during its preparation. For all of this (and his ability to keep the roles separate), I thank him.

ABSTRACT

This dissertation describes a computational program designed for the modeling of language change. It differs from most other work in computational linguistics in that its goal is the examination of scientific hypotheses rather than their efficient implementation.

At the heart of the program (called FLRN) is an implementation of the lexicase (Starosta 1988) syntactic framework. In the discussion of this implementation, a formalization of regular morphology is provided for the theory. A simple parser is set up using a combination of lexicase filters and a performance (phrase structure) grammar learned through experience. The language learning portion of the program includes a minimum number of general learning abilities: classification, simplification, and analogical reasoning or generalization.

Hypotheses concerning language change may be placed in either a FOCUS module (for claims about perceptual strategies and limitations), a pragmatic module (for claims related to discourse and situation), or an all powerful GUIDE module (for claims concerning universals of language or learning). A comparison of the output of the program and observed linguistic structures provides a measure of the effectiveness of the claims in accounting for language change.

The study includes examples of FLRN's use to model a simple language learner and a change from postpositions to prepositions in the Chinese languages.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The computer program FLRN described in this dissertation was begun as an attempt to model language change. Unfortunately, language change, with all its complexity, is quite an uncooperative subject for a computer model. To trace historical developments in a language, a study must make reference to a number of factors: social influence (both from within the language community and from without), rules of phonology and sound change, language learning processes from generation to generation, communication theory and functional load within phonology and syntax, literary traditions, and perhaps even the speakers' cultural world view. Since logic indicates that a model ignoring any of these interacting factors could easily lead the linguist to simplified conclusions in misleading directions, providing little insight into how languages actually change, the simple computer program I envisioned when I began this study grew quickly into an unwieldy model of all of language (both its grammar and its usage).

With our current level of knowledge, it is impossible to build an accurate model of a complete language system. At the same time, a lack of knowledge has not prevented linguists from dividing language into various putative subsystems and creating elaborate theories for each of them. Given this tremendous amount of work, it could be argued that a state of the art computer model of language should include the most recent (or best) of these theoretical models

for each of the generally recognized subsystems. But there are at least two major problems associated with building such a system:

First of all, most of these frameworks are excessively complex. In combining complex theories developed by linguists with different theoretical biases and different theoretical goals, it is impossible to be sure of writing a computer program in which all inconsistencies and incompatibilities have been eliminated.

It should be clear that any inconsistencies which exist would severely limit the testability of a computer model. A failure of the program to accurately model the facts of language could be due to the undiscovered inconsistency; conversely, any successes of the program might well be the result of the availability of two theoretically incompatible paths hidden in the program. In neither case would the performance of the computer program provide information concerning the adequacy of the linguistic theory on which it is based.

Secondly, very few linguistic theories are written in an explicit form. There are some fairly rigorous descriptive frameworks in the traditionally core areas of phonology and syntax, but even in such work it seems that each author discussing a different detail of language feels the need to introduce a new type of rule or a new procedure (generally without bothering to check whether this new idea is compatible with the ideas presented by other authors). While this approach may be desirable as a way to extend theories, it is not at all helpful to the computer programmer

who wishes to set up a model using a well defined, internally consistent theory.

Rather than create a monster out of bits and pieces of various theories, I have chosen to write the simplest program I could while paying at least token attention to all the various aspects of language performance. At the center of the program is a detailed implementation of lexicase (Starosta 1988, 1990), a lexically based dependency theory which provides the framework for the syntactic component and a launching point for an interpretive (semantic) component. Surrounding it are empty rule applying modules in which specific hypotheses may be stated and their predictions tested.

In spite of the fact that it has been designed to model all of language, FLRN is not a complex program. There are a few (not many) assumptions about language built into the system, but I have made an effort to make any such assumptions clear and explicit in the pages that follow. In most of its modules, FLRN is empty of content, allowing the user to test different claims.

It is customary at this point in a dissertation to provide a critical review of the relevant literature. In this work, the reader will notice few references to earlier studies in computational linguistics. Very few computational studies are direct antecedents of this study because of a difference in goals: FLRN has been designed as a tool for testing theories about language, while most computational work has been concerned with building an efficient natural language processor.

In Gazdar et al. 1987, a book-length bibliography listing over 1700 significant academic publications in the field of computational linguistics from the 1980s, there are no studies focusing on historical change and there are fewer than half a dozen items concerned with modeling the learning of language. However, a few studies modeling language change and language learning do exist.

Apparently inspired by Hymes 1965, Klein (1966) did a computer simulation of historical change, modeling language drift (as described in Sapir 1921) in a speech community (as defined by Bloomfield 1933). His approach was to model the linguistic interactions in a hypothetical community, with changes in the grammars of auditors determined by the linguistic structure of what they heard and by random numbers. This model showed a directed language drift, as expected. These results are interesting, but the extreme simplicity of the generative-recognition grammars that he used leads to questions as to whether the study is relevant to actual human language.

I am not aware of any other computer simulations designed to test specific hypotheses about how languages change. There are, however, a small number of computational models of language acquisition. The most famous is perhaps that of Berwick (1985). His model assumes a number of built-in, language specific abilities. During the learning of the language, the program merely examines the data for the correct choice of a specific parameter at each stage. This approach to language learning stems from an assumption that

grammar is too complex for a child to learn without the aid of a complex innate "universal grammar." O'Grady (1986), on the other hand, sets up a logical set of processes through which a child can learn language, providing an effective argument against this approach.

As would be expected, given the associated economic rewards, the majority of the work in computational linguistics focuses on natural language understanding--with large numbers of articles on parsing and the implementation of semantic rules and discourse logic in understanding and generating speech.

FLRN is not the first system to employ a syntactic theory as a base for its parser; it is possible to find at least one parser using nearly every currently popular grammatical framework. The lexicase model itself has been employed in two approaches to parsing (Starosta and Nomura 1986, Lindsey 1987). The Starosta and Nomura algorithm, restricted to English, is well organized and straightforward but pays too little attention to some key problems of homonymy. For instance, they suggest that prepositions be found early in the parsing procedure and that noun objects be immediately associated with them--but the majority of prepositions in English are homonyms of adverbs:

(1) John put the jacket on the table.
[+P]

(2) John put his jacket on.
[+Adv]

Similar problems exist due to the large number of noun-verb homonym pairs in English. Starosta and Nomura recognize the problem and suggest that a "place holder" consisting of shared features be used in such cases to minimize the need for backtracking. However, since prepositions and verbs in English are frequent homonyms with words of other classes, I believe the amount of juggling necessary with the use of place holders while forming early links in the parsing process severely limits the effectiveness of their algorithm.

Lindsey's (1987) FLX was an expert system built to handle grammars of any language. While it included an accurate model of the lexicase lexicon builder, it failed as an efficient parser because it lacked an effective approach to limiting the number of potential parses to be checked by the system.

This problem of focusing on a limited number of potential parses has attracted a great deal of attention, from the early parsing studies concerned with "garden path" sentences to the most recent work in discourse models. The failure of FLX clearly indicates that this problem must be solved in order for a lexicase parser to function, but I have hesitated to utilize any of the specific solutions in the literature in spite of my admiration for the careful work of such people as Grosz and her colleagues. I have, however, made allowance for the later inclusion of a pragmatic guide to lead the parser.

As stated above, the goal of FLRN is to provide a testable model of linguistic theories. The vast majority of the parsers

described in the literature have a very different goal; they are meant to be efficient at processing language. When they make use of a syntactic theory, it is with the idea that theory (written in stone) is an effective tool for parsing. The primary goal is that of efficiency. If the parser is slow or incompetent, the programmer is not averse to implementing shortcuts, which act effectively precludes the use of the program as a test of the original theoretical framework.

AN OUTLINE OF FLRN. FLRN is a computer program written to serve as a tool for examining various hypotheses concerning historical change. It has the form of a computer model (for both the competence and performance aspects) of language. It is designed so that different claims can be written into the system. When sentences from a proto-language (as suggested by those claims) are fed into FLRN, it can write a grammar for them (guided by the claims). This process can be repeated for as many generations as desired--until the modern version of the language (or something similar with the predicted characteristics) is produced or until it becomes clear that the changes are not proceeding in the predicted direction.

FLRN was not designed to model child language development (although it may turn out to be a useful tool for examining certain claims in that area). It was written as a means of modeling gross changes that occur over long periods of time; it lacks the

flexibility for finely tuned focus on the various stages that occur during the acquisition of language by a single generation.

The program is organized into major modules (Figure 1.1), which are briefly described in this section. The PARSER will be examined in more detail in chapter 2; the modules associated with learning will be described in chapter 3.

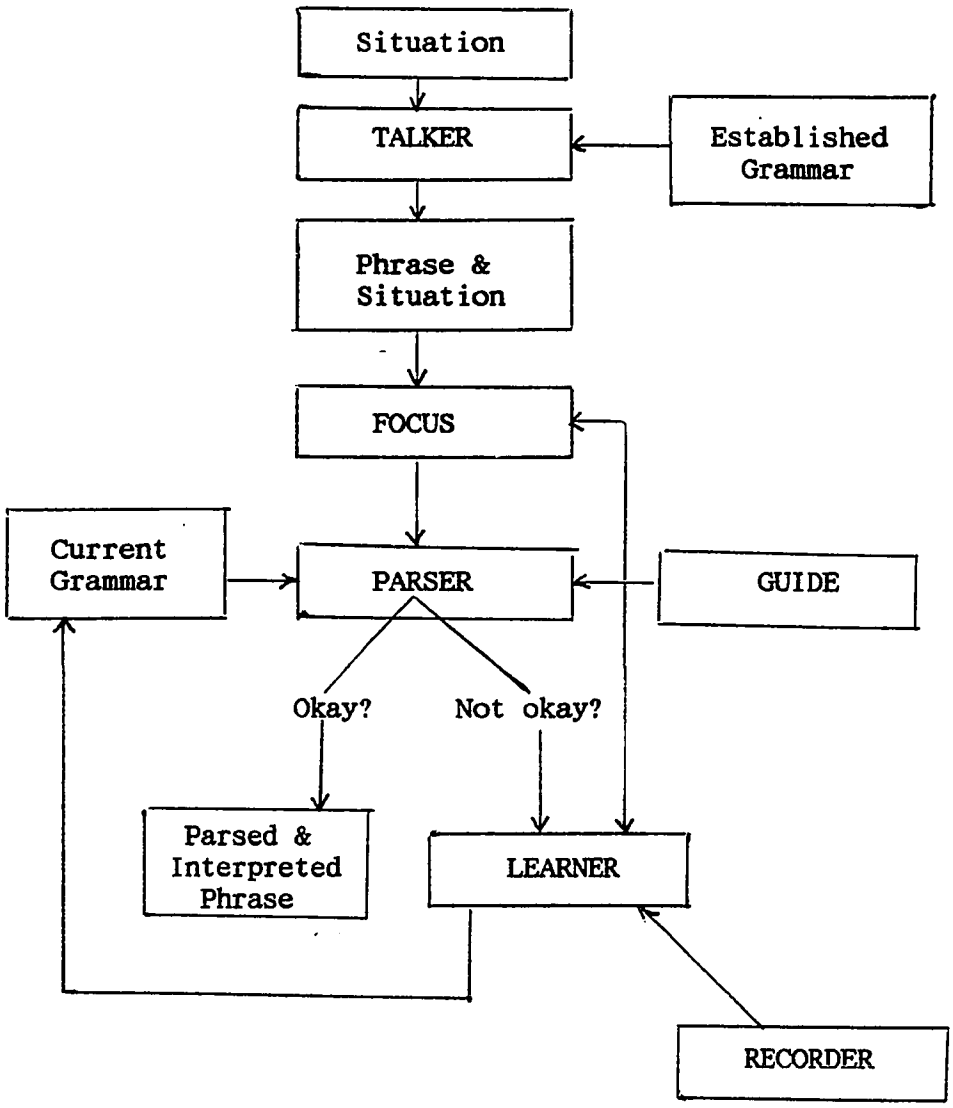


Figure 1.1. Diagram of FLRN.

The TALKER is not strictly speaking a part of FLRN. It is a simple language generator that takes an input situation (which generally includes a message to be communicated), refers to an established grammar, and produces a phrase (sentence or utterance) of the language to pair with that situation. The TALKER's primary use is to provide the input phrases and sentences from intermediate, hypothetical languages descending from the initial proto-language. The situations from which it works might be a single set, used with each of successive generations of the language, or new sets created for each generation (in order to account for differences in material culture through time).

The input to the learning portion of FLRN (starting with the FOCUS module) will normally consist of a phrase and its associated situation. This input may be provided directly by the user or it may be provided through the TALKER module.

The pairing of a phrase and a situation seems necessary in that language learning occurs in a human context, where phrases are not normally uttered without a meaningful purpose. The phrase will generally be a sentence in the parent language. The situation is, in theory, a list of all the information available to the hearer when the phrase is spoken; practically, of course, it can only be a small subset of such information.

The FOCUS module modifies the input before passing it on to the PARSER. In form, FOCUS is a simple expert shell which applies rules (representing the claims of the programmers) to eliminate parts of

the input or to change it in some other way. It contains no inherent rules; the rules must be added as explicit claims. Such rules may describe either static or developmental systems. For instance, rules focusing perception could define a simple synchronic system (static) or a series of systems which apply at different stages of maturity (developmental).

The PARSER attempts to interpret the input phrase (as modified by the FOCUS). It is the most nearly complete module in FLRN. Its core is a lexicase syntactic system, a model of the linguistic framework described in Starosta 1988 with extensions based on Starosta 1989, 1990, and Lee 1989. FLRN also includes in its PARSER a performance assisting module which may use experience to anticipate possible parses.

Following the flowchart in Figure 1.1, the PARSER will attempt to parse and interpret the FOCUS-modified phrase. If the parsing is successful, the correctly parsed phrase (and its interpretation) are the output of the system; no learning is considered necessary. But if an acceptable parse is not found by the PARSER, the phrase and situation are sent to the LEARNER.

The LEARNER module uses general learning functions, strategies stored in the GUIDE, and the experience retained in the RECORDER in an effort to parse the original input. As a result of these efforts, the LEARNER may propose changes to the current grammar, in which case the newly modified grammar will become the current grammar in the system. The basic learning functions built into

LEARNER are quite limited in number and in power; I have made an effort to include only functions considered necessary for general learning, avoiding language specific learning processes.

The LEARNER may fail, in which case it provides neither an acceptable parse nor an effective modification to the grammar.

The RECORDER is simply a record of experience. It may be set up to record any of the inputs, states, or outputs of the various modules of the program.

The GUIDE, like the FOCUS, is nothing more than an expert system shell. However, the rules which it applies should encode universals of language (both for performance and competence), special learning strategies beyond those in the LEARNER, or any other special claims the programmer might associate with the structure of language or with the processes involved in the learning of a language.

Hypotheses about language change or language development will normally be stated in the rules of the GUIDE and FOCUS modules. All of the special rules or theoretical biases built into FLRN will be made explicit in the next two chapters.

CHAPTER 2

THE PARSER

FLRN's PARSER contains both performance strategies and a competence model of syntax. The performance strategies provided are minimal--and clearly incomplete. They are placed in separate modules where they can be modified according to discoveries from future research. The competence model is a specific attempt to implement the lexicase syntactic framework.

The PARSER consists of LXBR (a lexicon builder--the core of a lexicase grammar), ASTR (a performance based parsing assister), PRSR (a parse suggester) and CHKR (a parse checker and a semantic interpreter). These modules and their functions are described in this chapter.

THE LXBR (The Lexicon Builder). The LXBR module models the portion of a lexicase grammar that is diagrammed in the flow chart in Figure 2.1 on page 13. It applies lexical rules to a set of minimally specified lexical items to generate a dictionary of words (fully specified entries).

THE WORD. The grammar of a language is contained in its words, according to the panlexicalist lexicase theory. Each word is defined as having "three parts: sound, meaning, and distribution" (Starosta 1988:45). These three aspects of a word are provided in FLRN by a phonological or orthographical form describing the sound and a feature matrix (that is, a list of features) describing the distribution and meaning.

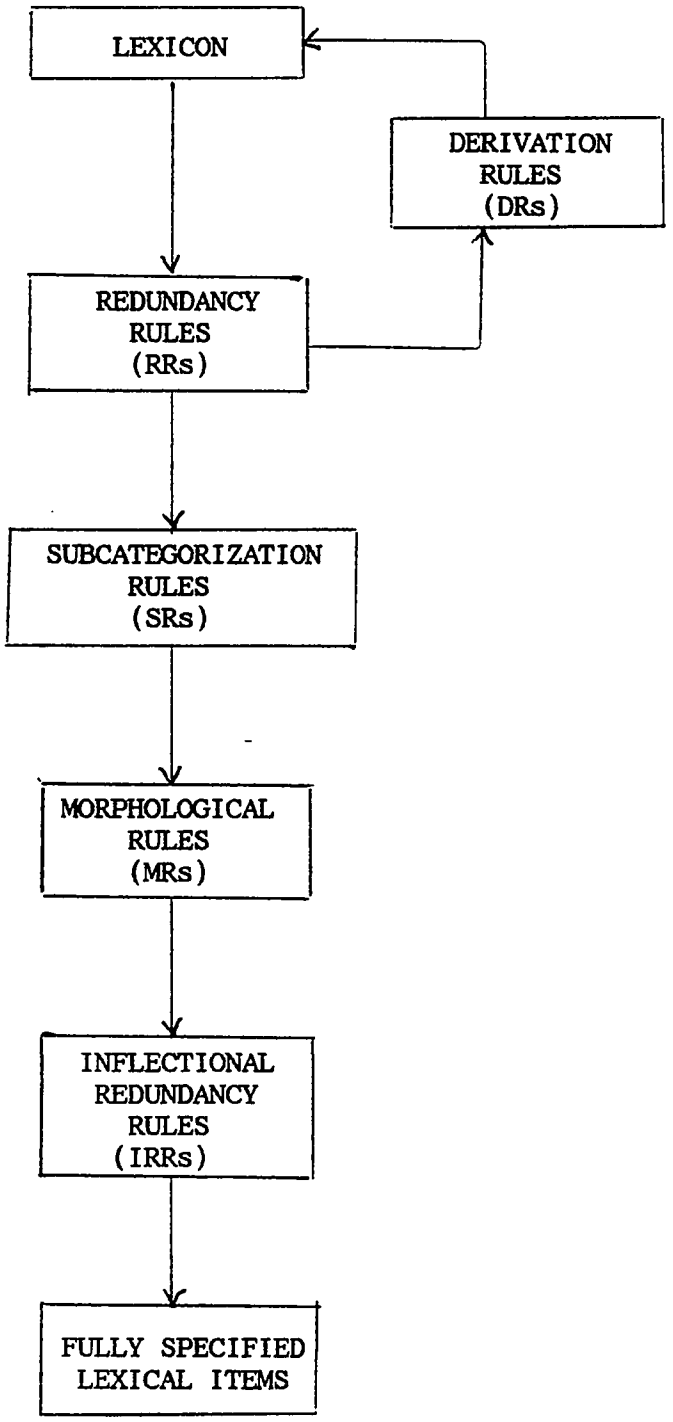


Figure 2.1. Flowchart of LXBR Functions.

THE FORM. The LXBR may operate quite independently of any phonological theory, allowing "phonological forms" to be written in any symbols acceptable to the computer. Obviously, standard orthographic forms may be used. The rules contained in other modules of FLRN, however, may make extensive use of phonological information. In such cases, the user must be sure that the formalism used for phonological forms and the formalism in the GUIDE or FOCUS rules is the same.

THE FEATURE MATRIX. Both the meaning and distribution of words are stated as features. No distinction is made between syntactic features (those which describe distribution) and semantic features (those which give meaning) in terms of type; Starosta (1988:53), in fact, mentions some as being "in-between" features having both syntactic and semantic functions.

A MINIMALLY SPECIFIED LEXICAL ITEM. As stated above, a lexicase grammar is contained in its description of the words in a language. However, a number of regularities can be extracted from the matrices of words and these generalizations can then be captured in rules. The irregular or non-predictable information which remains is the minimally specified lexical item.

A minimally specified lexical item, then, is a word or stem containing a phonological form in addition to (or in FLRN, which is written in LISP, consed to) a matrix containing only its non-predictable features.

THE LEXICAL RULES. The regular generalizations about the lexicon are captured in several sets of rules, namely lexical redundancy rules (RRs), subcategorization rules (SRs), morphological rules (MRs), and inflectional redundancy rules (IRRs).

The RRs state general relationships between classes of words. For example, it is predictable that all pronouns are nouns. This fact can be stated as a redundancy rule adding the feature [+N] to any matrix containing the feature [+prnn]:

(1) RR-a. [+prnn] --> [+N]

The SRs define the inflectional categories of the language. The fact that English finite verbs are inflected for tense would be written in a rule such as (2):

(2) SR-a. [+fint] --> [+/-past]

Starosta 1988 allows for SRs describing lexical subcategorization as well as inflectional subcategorization; I will argue against their inclusion in an empirical, testable theory.

The MRs describe the inflectional morphology associated with the inflectional categories introduced in the SRs. In the version of lexicase described in Starosta 1988, a rule such as (3) marks the past tense of English verbs:

(3) MR-a.] --> d] / [+past]

Below, I will argue that most MRs should be included as part of individual SRs--another deviation from current lexicase practice.

The IRRs are similar to RRs. The difference is that RRs are triggered by basic lexical features whereas IRRs follow from

inflectional features. The primary function of the IRRs is to account for agreement.

A FULLY SPECIFIED LEXICAL ENTRY. The lexicon of a language (comprising its grammar) consists of all its fully specified words. Fully specified lexical entries are generated by the application of all relevant lexical rules to minimally specified lexical entries. Each fully specified lexical entry is then a true word in the language.

WORD BUILDER. The processes described for the LXBR are actually accomplished by the application of a word builder (WDBR in the LISP code) to each input item.[1]

The initial input to the word builder is a single minimally specified lexical entry, either a word or a stem.

Redundancy rules (RRs) are applied first.

Derivational rules (DRs) may be applied to the output of the RRs. Any new word created by the DRs must be considered as a separate initial input to the word builder; that is, each newly derived word must be exposed to the RRs, SRs, etc.

Subcategorization rules (SRs) are applied to the output of RRs, separating words into different inflectional categories. FLRN allows for the inclusion of MRs within the SRs. See the section below on categ rules for details.

The output of the SRs is a list of words. Morphological rules (MRs) and inflectional redundancy rules (IRRs) are applied to each of these words in turn.

The final output of the WDBR is a list of fully specified lexical entries, that is, words of the language.

REDUN RULES. LXBR uses one format for RRs, MRs, and IRRs. Rules written in this format are called redun rules. Each is a list containing the following parts:

```
(rule-number left-matrix right-matrix
  morphological-change).
```

The rule-number may be any expression; within FLRN, it is used only for reference. The left-matrix and right-matrix must be feature matrices. The optional morphological-change (which will be discussed in more detail later) is a list containing the following parts:

```
(old new environment).
```

The environment may be omitted. The morphological-change may also be written as a list of such morphological-changes.

The fact that RRs, IRRs, and MRs are all processed by FLRN as redun rules does not imply that these rules should have the same contents. The rules must follow standard lexicase guidelines, meaning that RRs and IRRs will have no morphological-change and that MRs will always have an empty left-matrix. Items (4) through (6) are examples of how rules in standard lexicase notation are written as redun rules.

(4) A typical RR. (Note that there can be no morphological-change in the redun rule.)

Standard notation:

RR-a. [+prnn] --> [+N]

FLRN redun rule:

```
( RR-a  ((+ prnn))  ((+ N)) )
```

(5) A typical MR. (Note that the environment comes first in the redun rule and that the right-matrix is always nil. The actual contents of the morphological-change will depend on the morphophonemic or phonological notation used.)

Standard notation:

```
MR-a.  ] --> d]  /  [+past]
```

FLRN redun rule:

```
( MR-a  ((+ past))  ()  (] d])  )
```

(6) A typical IRR. (Again, the morphological-change must be omitted in the redun rule.)

Standard notation:

```
IRR-b.  ¦+mass¦ --> [-[+rtcl]]
        ¦-dfnt¦
```

FLRN redun rule:

```
( IRR-b  ((+ mass)(- dfnt))  ((- ((+ rtcl))))  )
```

A redun rule applies to a word if the rule's left-matrix is a submatrix of the word's (current) matrix. If the rule applies, the features in the rule's right-matrix may be added to those in the word's matrix provided that they meet certain conditions, specifically that the new features are not already in the word's matrix and that they do not conflict with any of the features already there.

If all of the features from the rule's right-matrix are successfully added, the phonological form of the word will be changed as described in the morphological-change of the rule.

Of course, morphological-changes will occur only in redun rules representing MRs or in redun rules created during the application of SRs (see below).

CATEG RULES. Subcategorization rules divide words into lexical and inflectional categories.

Lexical SRs, which never apply non-vacuously, are included by Starosta in his grammars because "they are part of the specification of the notion 'possible lexical item' ... and they may turn out to have a function in constraining the application of lexical derivation rules, leveling exceptional items, or naturalizing borrowings." (Starosta 1988:76).

Their use in defining possible lexical items is obvious, but they do so redundantly. In all cases, they state a fact that is already captured by the combination of a feature in at least one minimally specified matrix and an RR. The facts accounted for in (7) below are the same as those stated in the lexical SR in (8):

(7) lexical item: he [+N +pron]
plus
redundancy rule: [+N] --> [-pron]

(8) lexical SR: [+N] --> [+/-pron].

To date, no effort has been made to implement the claim that lexical SRs constrain DRs, level exceptional items, or naturalize borrowings. These lexical SRs currently have no empirical content; since they never apply non-vacuously, they cannot be tested. Until lexical SRs can be demonstrated to have some observable effect, I suggest their use be avoided.

Inflectional SRs, on the other hand, classify words according to inflectional classes; the features that they add have syntactic and/or morphological importance.

All SRs in FLRN are written as categ rules. Each categ rule contains the following parts:

(rule-number left-matrix category-list).

The rule-number and left-matrix are similar in form and function to those in redun rules. Each category-list is a list of (two or more) pairs, each consisting of

(right-matrix morphological-change).

A categ rule applies if the left-matrix of the rule is a submatrix of the feature matrix of a word. If it applies, the categ rule is applied as two (or more) simultaneously applying redun rules, with each member of the category-list providing a right-matrix and a morphological-change for a separate redun rule. Items (9) and (10) are some examples of SRs in the categ rule format and the redun rules into which they can be converted.

(9) A typical SR (after Starosta 1988, with no morphology included):

Standard notation:

```
SR-a.  [+V] --> [+/-past]
```

FLRN categ rule:

```
( SR-a  ((+ V))  (  ( ((+ past)) )
                    ( ((- past)) ) ) )
```

Converted to FLRN redun rules:

```
( SR-a  ((+ V))  ((+ past)) )
( SR-a  ((+ V))  ((- past)) )
```

(10) A typical SR (with inflectional morphology included):

Modified standard notation:

```
SR-a.  [+V] --> [+past] (] --> d])
            --> [-past]
```

FLRN categ rule:

```
( SR-a  ((+ V))  (  ( ((+ past))  (] d]) )
                    ( ((- past))         ) ) )
```

As FLRN redun rules:

```
( SR-a  ((+ V))  ((+ past)) (] d]) )
( SR-a  ((+ V))  ((- past)) )
```

As noted by Starosta (1988:83), there is a formal problem with inflectional morphological rules. "They apply to the output of SRs, but if fully specified irregular forms follow the same route as minimally specified stems, then words like <u>put</u> [+past] and <u>went</u> [+past] will ... come out as *<u>putted</u> and *<u>wented</u>" when the past marking morphological rule "] --> d] / [+past]" is applied.

One solution is to claim that, since the syntactic (competence) theory cannot handle the problem, it really belongs in the realm of performance. But the mere fact that a syntactic theory cannot account for certain phenomena is not enough reason to shunt them off into performance. And one argument against this claim is that adult speakers of English regard forms such as *<u>goed</u> and *<u>wented</u> as "wrong." Judgments of this sort are generally taken as evidence about the speaker's internal grammar (or knowledge of the language) and as facts which must be accounted for in a theory of competence.

(Of course, there is still a certain amount of discussion about what a speaker means when he claims a form is wrong.)

As an alternate solution, Starosta (1988:89) states that by "convention, morphological rules do not apply to words whose features were not introduced by earlier SRs...This convention has not yet been formally implemented in any full-scale lexicase analysis." One possible reason for this is that lexicase grammarians have developed a fetish for neat formulae. Successful description of inflectional morphology may require tearing apart some of the beautifully concise formalism in SRs in order to look at exactly what it represents:

In effect an SR functions as if it were two simultaneously applying redundancy rules. A rule such as that in (11) below can be rewritten as the two rules in (12), only one of which can be applied in any derivation.

(11) SR-1. [+V] --> [+/-past]

(12) SRX-1a. [+V] --> [+past]
SRX-1b. [+V] --> [-past]

In a graduate seminar in 1984, Starosta discussed the inclusion of morphological rules within subcategorization rules as an alternative to the approach he later preferred in <u>The Case for Lexicase</u> (1988). Dividing SRs into their parts (as in (12)) makes it possible to associate a specific inflectional marker with the inflectional category it marks, providing a formalism to ensure that morphological markers are only added when the inflection(s) they

mark are regular (that is, when their inflectional categories are provided by SRs). To make things clearer, SR-1 could be written as in (13):

```
(13) SR-1.  [+V]  -->  [+past]  ( ] --> d] )
                  -->  [-past].
```

Unfortunately, morphology is not generally as cooperative as in this simplified example. The handling of inflectional markers for two or more inflectional categories (as with portmanteau morphs) involves extra problems of ordering within the SRs when the morphology is included. Schematically, rules of abbreviated form like this one,

```
(14) SR-2.  [+feat1] --> ¦+/-feat2¦
                         ¦+/-feat3¦,
```

might require rewriting as in (15).

```
(15) SR-3a.  [+feat1] --> ¦+feat2¦ ( ] --> a] )
                          ¦+feat3¦

     SR-3b.  [+feat1] --> ¦+feat2¦ ( ] --> b] )
                          ¦-feat3]

     SR-3c.  [+feat1] --> ¦-feat2¦ ( ] --> c] )
                          ¦+feat3¦

     SR-3d.  [+feat1] --> ¦-feat2¦ ( ] --> d] )
                          ¦-feat3¦.
```

The two sets (14) and (15) are equivalent except for the inclusion of morphological details in the latter. Successful use of this approach requires careful examination of proposed SRs and more work in setting up correct rule order since neat feeding orders between syntactic classes might not necessarily correspond to the order in which inflectional affixes are added to a stem.

FLRN may operate with SRs (and MRs) using either Starosta's (1988) approach (and ignoring the irregularities of inflection) or the approach outlined here of including the regular inflectional morphology in the SRs. However, I believe that the latter is preferable in order to maintain the strong emphasis on explicitly testable hypotheses that is such an important part of the lexicase tradition. The careful attention to the relationship between morphology and inflectional categories this approach entails is preferable to the lack of such attention associated with the current approach.

DERIV RULES. The DRs are written as deriv rules. Each deriv rule is a list containing the following elements:

```
(rule-number productivity old-matrix
  new-matrix morphological-change)
```

The rule-number is simply a reference number. The productivity is a measure of how frequently the rule creates a new word when the conditions for application are met; it is expressed as a percentage. A deriv rule may apply if the old-matrix is a submatrix of the input word. If it applies, all of the features in old-matrix are removed from the word matrix and all of the features of new-matrix are added to the word matrix. In addition, the morphological-change is implemented. When a deriv rule is successfully applied to a word, the output is a new (different) word.

A deriv rule is a production rule which generates a derived word from a (more) basic word, schematically

(16) Basic >--> Derived.

If derivational processes are to be encoded as merely correspondences of derivationally related forms, then two deriv rules are necessary:

(17) Basic >--> Derived
Derived >--> Basic.

In a sense, DRs are not an integral part of the grammar. Derivation is typically irregular, so the DRs cannot be relied upon to create exactly the words contained in a speakers's lexicon. This means that individual (derived) lexical items must be listed separately in the lexicon, since their existence cannot be accurately predicted from the DRs. DRs are included however since they describe a knowledge of the relationships between words of the languages that speakers may use in interpreting a novel word and perhaps in creating new words (Starosta 1988:92-3).

MATRICES AND FEATURES. A matrix is a list of zero or more features. A feature is the notation used to mark a form as belonging to a certain class. Lexicase non-contextual features are binary features with, ideally, the positive (+) values used for marked classes. In recent work (Pagotto 1985; Pagotto and Starosta 1986; Lee 1989; Starosta 1989, 1990; for example), the notation has evolved to include other feature values to account for meaning and reference.

Within FLRN, each feature contains two parts: a value and a feat. (Note the terminology here: a feat is one part of a feature, not an abbreviation for feature.)

VALUES. The value may be any of the following:

+	(non-contextual)	'defines class membership'[2]
	(contextual)	'required sisterhead(s)'
-	(non-contextual)	'defines class membership'
	(contextual)	'prohibited sisterhead(s)'
?	(contextual)	'implied referent'
>	(contextual)	'implication'[3]

Other features specify possible adjuncts, that is, they name optional dependent sisterheads. In standard lexicase notation, these features would have a value of + or ? with a feat enclosed in parentheses, signaling an optional matrix. Because they affect feature addition differently from other + and ? features, FLRN uses slightly different notation; this is simply a notational variant and does not imply a modification in the standard lexicase marking and use of such features:

*	(contextual)	'optional sisterhead'
*?	(contextual)	'implied adjunct'

The final type of value possible in the lexicase grammar is a number. Such values are only used to index words in a phrase and to specify the referents (by index) which satisfy ? or > contextual features (Lee 1989, ch. 4).

FEAT. A feature's feat may contain either a single classification category (within FLRN, a LISP atom), as in the FLRN (+ N) notation of the standard lexicase feature +N; a single matrix (for simple contextual features), as in the FLRN (? ((+ AGT))) notation of ?[+AGT]; or a series of matrices

(for contextual features specifying ordering restrictions or requirements), as in FLRN's (- ((+ Adj)) ((+ Det))) notation of the standard lexicase -[+Adj][+Det].

Table 2.1 below provides some examples of lexicase features written in FLRN notation.[4]

Standard lexicase notation	FLRN equivalent
+N	(+ N)
-[+N]	(- ((+ N)))
?[+PAT]	(? ((+ PAT)))
+([+LOC])	(* ((+ LOC)))
?([+LOC])	(*? ((+ LOC)))
⇒¦+PAT ¦ ¦+anmt¦	(> ((+ PAT) (+ anmt)))
-[+Adj][+Det]	(- ((+ Adj)) ((+ Det)))
-___[+Det]	(- __ ((+ Det)))

Table 2.1. Lexicase Features Written in FLRN Notation.

THE SUBMATRIX PREDICATE (SUBM). The lexical rules (redun, categ, or deriv) apply to a word if their leftmost matrix is a submatrix of that word's matrix. The submatrix predicate used (the function SUBM) is not strictly speaking a subset predicate; it tests whether the features of one matrix match features in another, not whether they are the same features. Two features match if their feats are the same and their values match (with an output of 1) when

compared by the value matching predicate M-VAL. The table of outputs for M-VAL is provided in Table 2.2.

		LEFT VALUES						
		+	-	*	+/-	?	*?	>
RIGHT VALUES	+	1	0	2	1	2	2	2
	-	0	1	0	1	2	2	2
	*	2	0	1	2	2	2	2
	+/-	1	1	2	1	2	2	2
	?	2	2	2	2	1	2	2
	*?	2	2	2	2	2	1	2
	>	2	2	2	2	2	2	1

M-VAL returns
0 or nil for conflicting values,
1 for matching values, and
2 in all irrelevant comparisons

Table 2.2. The M-VAL Function Table.

Following are a few examples using the SUBM function:

(18) Is the matrix [+mass] a submatrix of
```
¦+N      ¦?
¦-mass   ¦
¦-plrl   ¦
¦?[+Det]¦
```

No. The feature +mass has the same feat as -mass, but their values conflict; that is, M-VAL returns 0 when comparing + with - values.

(19) Is the matrix [-mass] a submatrix of
```
¦+N      ¦?
¦-mass   ¦
¦-plrl   ¦
¦?[+Det]¦
```

Yes. The feature -mass has the same feat as -mass, and their values match; that is, M-VAL returns 1 when comparing - and - values.

(20) Is the matrix [*?[+Det]] a submatrix of

```
¦+N      ¦?
¦-mass   ¦
¦-plrl   ¦
¦?[+Det]¦
```

No. The feature *?[+Det] which equals the standard lexicase feature ?([+Det]) has the same feat within FLRN. M-VAL returns 2 when comparing *? and ? values, indicating that the values do not conflict but neither do they match.

PARSING (ASTR, PRSR, and CHKR). The job of a parser is to assign a syntactic structure to a phrase. Ideally, the parser should assign a single structural description into which all of the words of the phrase fit--or, in the case of ambiguous phrases, only those structural descriptions which correspond to distinct meanings.

The task of parsing within FLRN is handled by a combination of ASTR, PRSR, and CHKR. ASTR assigns structural readings using a combination of pragmatic rules and experience; its contents represent performance strategies for parsing. PRSR and CHKR do not correspond to specific human functions. PRSR is a strictly computational approach to determining how words might be fitted into a single hierarchical structure. CHKR applies the lexicase restrictions on structures, filtering out unacceptable structures.

THE PARSE CONTROLLER. All parsing attempts are controlled by a routing function T-PRS (Top Level Parse Controller). It is outlined in (21).

(21) T-PRS (Top Level Parse Controller)

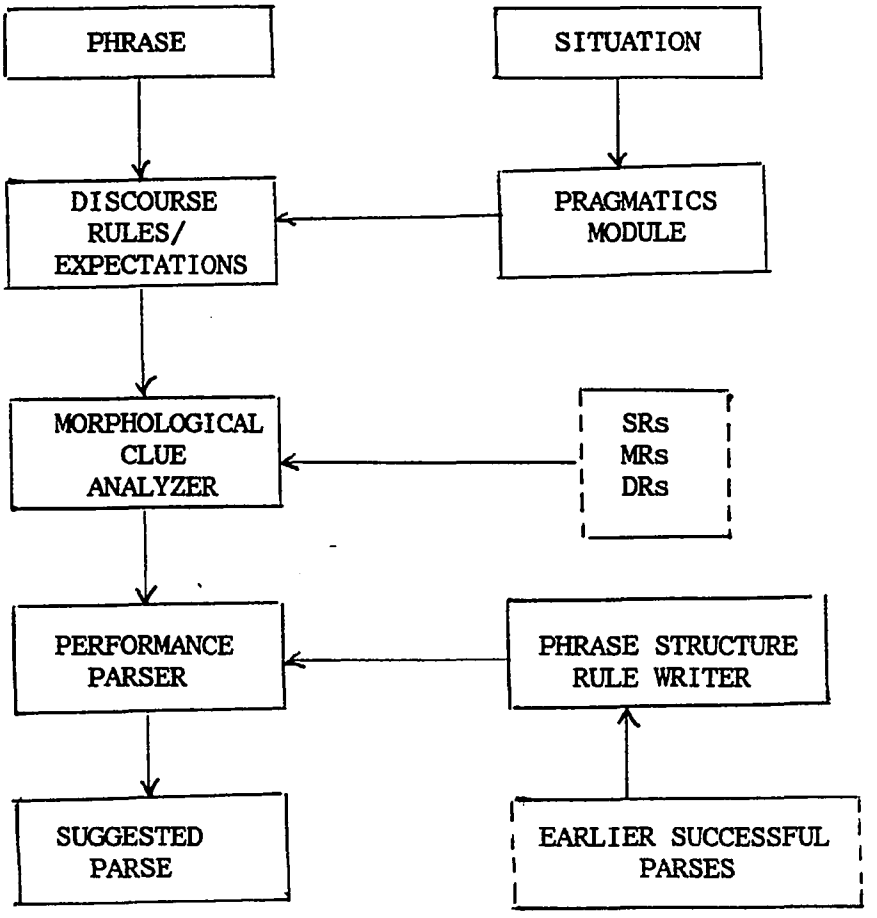
```
Send phrase to ASTR
    If successfully parsed, send to CHKR
        If okay, output
        Otherwise, send to PRSR
    If not successfully parsed, send to PRSR
        If successfully parsed, send to CHKR
            If okay, output
            If not okay, send to LEARNER
        If not successfully parsed, send to LEARNER
```

First, the input phrase is sent to a performance assister (ASTR). The ASTR may use pragmatic expectations, morphological clues, and a simple phrase structure (chart) parser built from experience to impose a structural description on the phrase. If it succeeds, the suggested parse is sent directly to the checker and interpreter (CHKR), which either successfully processes it as output or sends it back to the more thorough computational parse suggester (PRSR). The ASTR, representing a performance aspect of parsing, does not correspond to any portion of current lexicase descriptions.

The PRSR suggests structural descriptions for those phrases which cannot be successfully parsed by ASTR. Again, suggested parses are sent to the CHKR for final checks and interpretation. A failure by either the PRSR or the CHKR causes the input to be sent to the LEARNER.

ASTR (The Parsing Performance Assister). The inclusion of the ASTR in FLRN involves a set of implicit claims, to wit: (a) that people use semantic/pragmatic information to help them assign structural descriptions, and (b) that they recognize and use morphology in determining structural descriptions, and (c) that they

recognize linear sequences of word types in their language and use that knowledge to anticipate hierarchical relationships.



The items marked in broken lines are not integral parts of ASTR; they are from other modules of FLRN.

Figure 2.2. Diagram of ASTR.

The ASTR includes those sections diagrammed in Figure 2.2. Its input is a string of words. It attempts to impose a structural

description on this phrase on the basis of discourse/pragmatic expectations, morphological clues, and previous experience.

The separation of ASTR from the strictly computational PRSR is a claim that people, in their attempts at parsing, develop a set of performance strategies (heuristics) that are distinct from (although often referring to) the knowledge described in a competence grammar. This is not an original claim. Parsing has always been regarded as a performance task. Mistakes and false starts such as those associated with "garden path" sentences are typical performance errors. In these cases and with other parsing problems, it is obvious that reference is made to external (non-linguistic) information--and thus parsing processes cannot be strictly captured in a description of linguistic competence.

A parse may turn out to be incorrect according to the hearer's knowledge (competence) of the language in cases like "garden path" sentences. It may also be arrived at through the use of strategies based on partial truths. For example, most English speakers assume that adjectives precede nouns in noun phrases. Such an assumption allows a parser (human or computer) to anticipate a modified noun upon encountering an adjective, providing a useful shortcut to a successful parse. Stated as a strict rule, however, it may not account for the facts of English:

(22) A happy man rarely has fleas.
A man happy in his work is seldom a sewer cleaner.
A teenager happy to be seen with his parents is rare.

FLRN's core is a lexicase (competence) grammar. In it, all claims must take the form of explicit rules with observable effects. If such rules generate incorrect forms or structures, the rules are wrong. Period. On the other hand, parsing performance seems to be assisted by strategies--heuristics that may be incomplete or even partially wrong, but which nevertheless assist in the performance of a task. ASTR has been included in order to keep these performance strategies separate from the more rigorous statements describing the generative competence model.[5]

PRSR (The Parse Suggester). Unlike the ASTR, the PRSR is not at all concerned with accurately modeling details of human performance. It is strictly a computational approach to determining how a string of words can be fit into a single hierarchical structure satisfying the requirements of the lexicase theory.

The lexicase generative model of syntax functions as a set of filters; the problem of proposing likely candidates for well-formedness has generally been ignored. The grammar merely provides a set of filters for determining which phrases (out of all those possible) meet the requirements of the grammar. An accurate computational model of the lexicase grammar would thus examine all structurally possible trees as potential parses. This approach, while ideal in terms of completeness, is impractical in terms of computational time.

The PRSR uses a modification of this method. Words which allow no dependents are parsed as single word phrases; all other subtrees

with that word as a head are impossible and therefore their possible existence can be ignored. In addition, certain well-formed complete phrases suggested by ASTR and accepted by CHKR may be used as partial parses.

There are two possible trees for a phrase containing only two words; one in which the first word is the head and a second in which the second word is the head. With an increase in the number of words in the phrase, the number of possible trees increases rapidly. If all the words in the phrase are considered potentially capable of filling either the role of head or dependent, the number of possible trees associated with a short sentence of only ten words runs well over a thousand. And since a lexicase dictionary includes a large number of homonyms, the standard approach to parsing (that of looking up the class of a word in the dictionary and plugging it into a standard phrase structure) is extremely time consuming. Without the assistance of ASTR, the PRSR is so slow as to be nearly useless for sentences of normal length.

CHKR (The Interpreter and Checker). The CHKR performs two distinct roles. It applies interpretive rules to the trees proposed by ASTR and PRSR, and it also checks for well-formedness.

Interpretive rules include linking rules (LRs) and chaining rules (CRs), structural semantic rules, and situation sensitive pragmatic rules.

LRs and CRs are established lexicase rule types, although relatively recent additions to the framework. They are the result

of careful early study by Pagotto (1985) into the interpretation of missing dependents, but their use has been extended by Starosta (1989, 1990) and Lee (1989) to make explicit the type of relationship existing between individual regents and their dependents.

A number of meanings are predictable from the dependency relationships of the words in a sentence. Certain semantic features may be imposed upon dependents by their regents, a fact normally captured by selectional restrictions such as those proposed in Chomsky 1965. In addition, the normal function of dependents (or modifiers) is to restrict in some way the meaning of their regent. Such rules have been assumed to have an important role in interpretation, but to date there has been no formal implementation by lexicase grammarians in actual grammars. Situation sensitive pragmatic rules are currently nothing more than "pie in the sky." It is not unlikely that they can never be effectively implemented since they may require an encoding of all of a speaker's knowledge of the world. However, a complete performance model of language clearly must include a component for recognizing and using situational semantics, so FLRN includes provision for later inclusion by means of this empty module in CHKR.

CHKR also examines the parse for well-formedness.[6] Since all of the parses submitted by ASTR or PRSR will satisfy the basic structural requirements of a lexicase grammar, the CHKR need make

only a limited number of checks. It must check that no ordering violations exist and that all dependencies are interpreted.

The CHKR performs its functions in the order given in (23):

(23) 1. Apply LRs and CRs.
2. Check for well-formedness.
3. Impose implications.
4. Apply situational pragmatic interpretation rules.

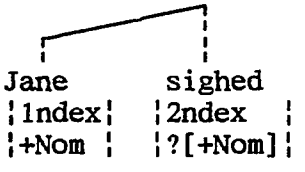
LINKING RULES. There are 2 types of linking rules (LRs): valence linking rules and internal linking rules.

A valence linking rule has as its domain a regent and its immediate dependents. Such rules generally assign the index of one of the sisterheads to replace the ? value of a required implicational feature on the regent, thus indicating which dependent satisfies the role described by the ? feature.

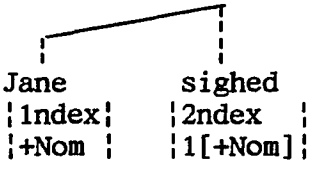
(24) LR-1 [?[+Nom]] --> [n[+Nom]] / ¦+Nom ¦
¦nndex¦

Rule (24) applies to phrase (25), assigning an index to yield the interpretation (26).

(25)



(26)

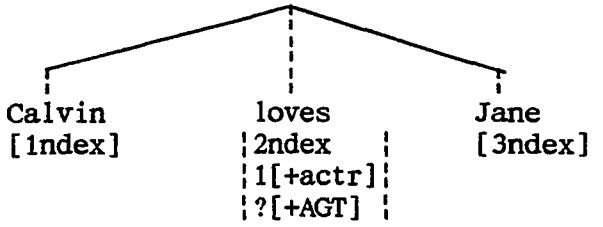


An internal linking rule refers only to the matrix of a single word. As with the valence linking rule, it replaces the ? of a ? feature with an index to indicate what satisfies the role indicated by the ? feature.

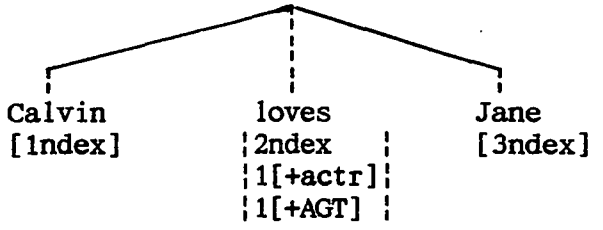
(27) LR-2. ¦n[+actr]¦ --> [n[+AGT]]
¦?[+AGT] ¦

Rule (27) simply equates actors and AGENTS. It would apply within the matrix of the verb in (28) to give the interpretation in (29).

(28)



(29)



LRs in FLRN's notation have the same parts as LRs in the standard lexicase notation:

(rule-number left-matrix assignment environment)

Rule-number is used only for reference. Left-matrix is the matrix which must be matched in order for the rule to apply. Assignment states what index value is to replace the ? value in the left-matrix. Environment states the context in which the rule applies; it is omitted for internal linking rules. Item (30) is an example of a lexicase LR written using FLRN's form of notation:

(30) A typical LR.

Standard lexicase notation:

LR-1 [?[+Nom]] --> [n[+Nom]] / ¦+Nom ¦
¦nndex¦

FLRN notation:

```
(LR-1 ( (? ((+ Nom)) )  ( (n ((+ Nom)) )
  ( / ((+ Nom) (n ndex)) )    )
```

CHAINING RULES. Chaining rules (CRs) are the lexicase approach to the problem of establishing reference for "missing" components. They are similar to LRs in that they mark the indices of items which fill specific roles. However, unlike the LRs, they are not limited to the immediate grammatical domain; they may refer to other matrices besides those of their sisterheads.

The environment in CRs must refer to a word in a command or cap-command relationship:

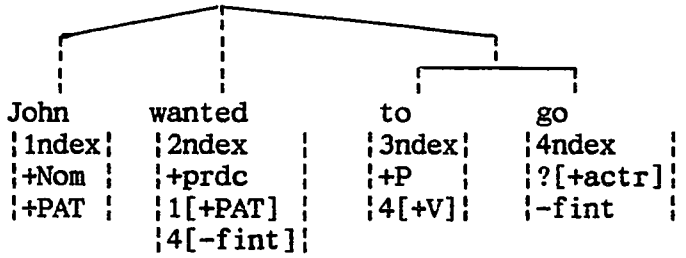
(31) Definition of cap-command. A word X cap-commands a word Y if X is the head of the phrase containing Y and Y is a sisterhead of X.

(32) Definition of command. A word X commands Y if either (a) X cap-commands Y, or (b) X cap-commands Z and Z commands Y. (Starosta 1988:109)

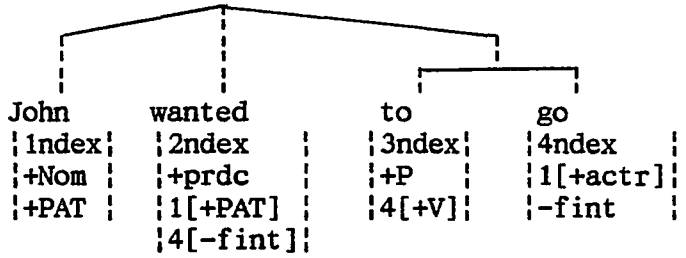
The chaining rule in (33) provides for interpretation of the missing subject (actually, the actor) of infinitival complements. It states that the missing actor is to be interpreted as the PATIENT [+PAT] of the cap-commanding predicate. (The backward slash \ indicates a cap-commanding matrix; a double backward slash \\ indicates the nearest commanding matrix meeting the appropriate description.)

```
(33) ¦?[+actr]¦ --> [n[+actr]] \ ¦+prdc    ¦
     ¦mndex   ¦                  ¦n[+PAT]  ¦
                                 ¦m[-fint]¦
```

(34)



(35)



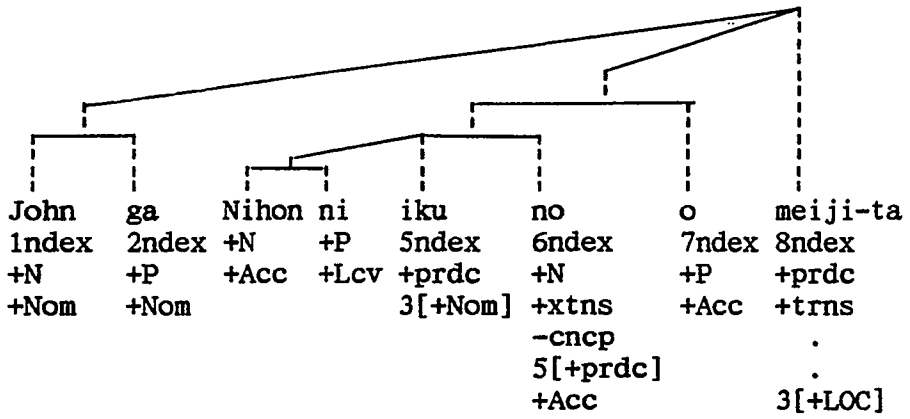
The FLRN CRs are identical in format to the LRs:

(rule-number left-matrix assignment environment)

They differ in that the CR environment must refer to a cap-commanding or commanding matrix (indicated by the listing of an appropriate \ or \\ as the first element in the environment), while the LR environment may only be null or refer to a sisterhead matrix.

To my knowledge, there has been no detailed discussion of the order and manner in which LRs and CRs should be applied. Lee (1989) applies LRs before CRs, but the resultant trees are not fully specified; her item (21) from chapter 4, repeated here in abbreviated form as my (36), for example, includes the subordinate verb <u>iku</u> without a PATIENT index--anathema to lexicase dogma. Presumably, LRs must be applied again after CRs to assign the necessary PATIENT index.

(36) An abbreviated version of Lee's ch. 4, item (21).
John ga Mary ni Nihon ni iku no o meiji-ta.



Lee follows a longstanding linguistic tradition in pointing out the desirability of ordering rules from the more specific to the more general, but she provides limited examples among her LRs and CRs. Springer (1992) in her detailed discussion of LRs for describing Japanese topicalization appears to have imposed no special ordering on her rules; she seems to apply them globally.

Lacking clear guidance in the lexicase literature, I have set up the following procedure for the application of LRs and CRs in CHKR. The CHKR begins its application of LRs and CRs by examining each word in the tree (in order from the uppermost head to the most deeply embedded dependent) for ? or *? features. The complete set of LRs is applied in order (allowing more specific rules to bleed more general rules) to each word with uninterpreted ? or *? features. If, after each word has been examined and the relevant LRs applied, the tree still contains unindexed ? or *? features, CHKR makes a second pass.

In the second pass (and any subsequent passes necessary) the words are examined in order from the most deeply embedded to the head. The complete set of CRs is applied in order, then the complete set of LRs, to any word with ? or *? features. The CHKR will continue with similar passes until there are no longer any ? or *? features unindexed or until a pass results in not a single index being marked.

The order in which LRs and CRs are applied is summarized in (37) below.

(37) 1. Apply LRs in order from top (head) to bottom (most deeply embedded).

2. Apply CRs and then LRs in order from bottom to top.

3. Repeat 2 until either (a) no unindexed ? or *? features remain or (b) not a single CR or LR applies during one complete pass.

The CHKR performs its check for syntactic well-formedness immediately after application of LRs and CRs. The decision to implement syntactic checks before the application of other semantic interpretation rules is the result of the following considerations.

In a lexicase analysis of English, it is possible to imagine two methods of noting that the subject precedes the verb. One approach (outlined in (38) below) is to mark all nouns as having two lexical entries. One of these is inherently [+Nom] and the other inherently [-Nom], based on the principle of maximal distinctions, in accordance with the fact that one type of nouns (pronouns) clearly has such distinct forms. The alternative approach (in (39))

does not mark (most) nouns in the lexicon as being [+Nom] or [-Nom]; it provides these features through implications associated with the verb.

(38) Lexicon:

```
people      people      play
¦+N  ¦      ¦+N  ¦      ¦+V      ¦
¦+Nom¦      ¦-Nom¦      ¦+fint   ¦
                        ¦?[+Nom]¦
```

(39) Lexicon:

```
people      play
¦+N  ¦      ¦+V      ¦
            ¦+fint   ¦
            ¦?[+Nom]¦
            ¦>¦+N  ¦¦
            ¦ ¦+Nom¦¦
```

In both cases, an ordering feature [-___[+Nom]] will filter out any postverbal [+Nom] forms. The second approach is appealing in that it corresponds to intuition; the nominative role is assigned by the verb and not due to an inherent subjectness of the lexical item.

While writing an earlier lexicase parser, I (Lindsey 1987) used the latter approach in order to reduce the number of homonyms in the lexicon, so I am not opposed to it on general principles. However, throughout this discussion of FLRN I have insisted on explicit, testable hypotheses and clear, straightforward principles. The availability of two alternative types of solutions to a specific syntactic problem is to be avoided unless there is an empirical method of choosing between them.

In checking for well-formedness before applying semantic implications, the CHKR prohibits syntactic descriptions from using

the second approach; a lack of the required dependents will cause the CHKR to discard the parse before the semantic implications can add the necessary features.

The CHKR's check for syntactic well-formedness involves a search for violations of ordering features, negative contextual features, and uninterpreted ? features. If any of these is found, the proposed parse is rejected.

The second interpretive task (after application of LRs and CRs) of the CHKR is to impose implications. Implicational features (those with > values) describe expected semantic features of sisterheads; such features are the means of marking semantic subcategorization. Sentence (40) provides an example of their use. This sentence is unusual in that the subject is not animate. By imposing the implications (that is, copying the implied feature onto the appropriate sisterhead), one can clearly mark the strangeness--the subject is both animate [+anmt] and [-anmt] in (41) after the imposing of the implication.

(40)

```
the       rocks      cried.
¦+Det¦    ¦+N    ¦   ¦+V        ¦
          ¦+PAT  ¦   ¦>¦+PAT  ¦¦
          ¦-anmt¦    ¦ ¦+anmt¦¦
```

(41)

```
the       rocks      cried.
¦+Det¦    ¦+N    ¦   ¦+V        ¦
          ¦+Nom  ¦   ¦>¦+Nom  ¦¦
          ¦-anmt¦    ¦ ¦+anmt¦¦
          ¦+anmt¦
```

The use of implicational features (or their equivalent in other theories) has not been carefully examined. They are normally referred to in a short note commenting on their usefulness in accounting for metaphor, and then ignored. In the absence of reasons to the contrary, the FLRN implementation requires that these implications always be imposed.[7]

A second type of intensional semantic interpretation in dependency grammars is the limiting effect of modifiers. A common example demonstrating this type of limiting is the phrase 'the toy soldier.' The meaning of the word <u>soldier</u> would normally include an assumption of humanness; the modifier <u>toy</u> clearly limits or changes the meaning of its regent, the word <u>soldier</u>.

CHKR does not include provision for this type of interpretation because it is as yet unclear exactly how such limitation is to be implemented: Does the modification of nouns by adjectives differ from the modification of predicates by noun dependents?
In what way?

If the CHKR accepts a proposed parse as well-formed, the T-PRS (Top Level Parse Controller) routes the accepted and interpreted parse as output. If the proposed parse is not accepted as well-formed, it is passed on to the LEARNER and related modules, which are described in the next chapter.

NOTES FOR CHAPTER 2

1. The program code for LXBR is provided in Appendix A. Programmers will notice that the WDBR function is written to provide for clear separation of derivational processes from other, more central word building processes. In this and many other cases, FLRN provides examples of inefficient programming. Much of this has been intentional since a more important goal than efficiency has been that of providing a clear implementation of linguistic models. In the case of the LXBR, the goal was to match the functions of the program with specific lexicase ideas or principles.

2. Positive (+) contextual features are being phased out in lexicase grammars. They are being replaced by ? implications, primarily because the latter require an explicit statement of which sister satisfies the specific contextual feature.

3. In FLRN and in this dissertation, the symbol > is used in place of the horseshoe implicational symbol ⇒.

4. Since FLRN will normally load features and rules via access functions, it is not necessary for the user to write features in FLRN format. The details here are for information purposes only.

5. Of course, a long range goal is to find strict generative rules for performance as well. In fact, the original goal for FLRN was to discover such rules for historical change--a subject clearly related to performance.

6. Since Khamisi's (1985) observation that case frames are often not overtly filled, failing to fulfill the syntactic

requirements marked by positive implicational features, there has been some hesitation among lexicase grammarians in the use of the term "ungrammatical." Currently, the practice is to call a failure to satisfy requirements a type of "ill-formedness."

7. The ? features originally developed from implicational features, following Khamisi's (1985:340-1) suggestion. In their current form, however, they are used in the locating and marking of items meeting specific syntactic/semantic requirements. It is not clear whether they should still be awarded the ability to impose an implication on a dependent.

CHAPTER 3

LEARNING

FLRN should be able to learn the grammar of a language from experience. The input from which it develops that grammar consists of a paired phrase and situation.

PHRASE. Ideally the input phrase should be a detailed phonetic representation marking segmentals, syllable and word junctures, and supersegmentals (especially word and phrase accents). This may seem odd in that a child, in learning a language, masters its phonology. However, certain details of the phonology may be acquired later than some parts of the syntax, so FLRN has been set up to handle raw phonetic input so that it can model even the earliest stages of language learning. (It also seems probable that phonetic details are occasionally important in language change, in which case phonemic notation might preclude the discovery of such developments.)

Of course, an accurate phonetic description of defunct languages is generally unavailable, and for such languages, something less than the ideal representation must be used.

Several formats are possible for the input phrase. I have been using a list consisting of the following:

(label string-of-sounds).

The label is just a number or simplified form that allows for easy reference to the item. The string-of-sounds is the actual input phrase to be modified by FOCUS and processed by the various modules of FLRN.

THE SITUATION. The situation includes all of the information relevant to the utterance of the phrase. It normally includes (but is not limited to) information about the speaker, information about the hearer, information about the environment (including the speaker's and hearer's views, attitudes, goals, etc.), and the message or content.

There are a variety of formats possible for entering this type of information. I have been using a simple list format with only four required items:

(speaker hearer context message-communicated).

Each of these items may be null whenever that information is considered irrelevant to the hypothesis being tested.

While it is possible for the LEARNER of FLRN to work effectively with situations containing little information, I have made provision for more complex situations in the hope that FLRN will adequately test complex models of language learning with quite detailed semantic and/or pragmatic components.

One must be aware that learning situations do not necessarily contain explicit messages. Part of what a child must learn in mastering a language is what meaning is to be associated with a particular word or phrase. While caretakers (that is, whoever provides the information children need to learn a language) may often make a meaning clear when speaking, it seems unlikely that they are always capable of effectively doing so.

Of course the normal communication of information (once the language is mastered) involves the decoding of the phrase rather than an extralinguistic signaling of meaning. Once FLRN has learned the grammar of a language, the PARSER should be able to interpret the meaning of the input phrase without reference to a situation. Even in such cases, however, pragmatic considerations may make it desirable to provide for reference to the situation, a possibility provided for by the inclusion of a pragmatic module within ASTR.

THE LEARNER MODULE. The LEARNER contains a limited number of inherent learning procedures, along with functions for accessing rules within the FOCUS and GUIDE modules.

The basic learning abilities included in LEARNER are of types considered necessary for general (cognitive) learning. In designing it, I have been strongly influenced by Pinker's (1984) description of bootstrapping and O'Grady's (1988) discussion of the acquisition of categorial grammars.

I wish to avoid the assumptions associated with what O'Grady (1986) has labeled special nativism. The proponents of special nativism claim that a child has a great deal of inborn knowledge which strictly defines the type of grammar he will build from the input he is exposed to. The opposing point of view is that no special inborn linguistic knowledge is necessary--that the child can successfully build his grammar without reference to any innate syntactic knowledge. This general nativism approach assumes that

the notions needed to construct a grammar are either nonlinguistic or they are derived from more basic nonlinguistic notions (O'Grady 1991:250-1).

The learning abilities inherent to LEARNER do not correspond in exact detail to either the strategies described in Pinker 1984 or in O'Grady 1986, nor do they correspond to the modules proposed by O'Grady 1991. Rather, they are basic abilities which I believe are necessary to the learning described within all of these approaches.

The LEARNER has four built-in learning abilities. First, it can put items into general classes (classification). Second, it can recognize sequences of items as forming one larger item (combination). Third, it can impose a dependency or hierarchical relationship between items (ranking). And, fourth, it can form generalizations on the basis of similarities (analogy).

CLASSIFICATION. The ability to put items into classes is clearly a basic cognitive ability. I know of no description of cognitive behavior (nor is it possible to imagine one) which does not assume this ability.

The FLRN implementation is quite straightforward. Any item may be classified according to any characteristic or feature. This version of classification is probably not a realistic model of the ability in humans. The FLRN LEARNER uses all of the information about an item, while humans most likely focus on particular salient information when making their classifications, ignoring other features. But the question of which features are salient seems

quite complex, focusing as it does not only on the prominence or inherent importance of the features but also on the goals or aims of the person doing the classification. For this reason, FLRN relegates the problem of choosing salient classification criteria to the group of unknowns to be built into the FOCUS or GUIDE modules for closer examination.

One type of saliency may be handled within LEARNER: recurring features assume some importance as a result of the analogical reasoning ability described below.

COMBINATION. Combination is the ability to consider one or more items as forming a single item. It might also be labeled a substitution ability in that it allows the LEARNER to replace a single item with a sequence of items. It is doubtful that humans (with short term memories retaining seven, plus or minus two, items) would be able to perform complex thought processing without this ability to simplify sequences of large numbers of items by the use of combination.

Combination is a learning ability concerned with the formation of linear structural relationships. Unlike the similar portion of O'Grady's 1991 "computational module," it is not limited by binarity; it may combine any number of elements into a single structure.

Also, it does not automatically impose dependency relationships; it merely allows for the definition of a structure as a particular (linear) sequence of items.

RANKING. Of couse, dependency relationships are basic to the lexicase grammars that LEARNER is to write. To successfully write such grammars, FLRN must be able to determine (or at least impose) hierarchical relationships.

O'Grady (1991) follows a long-standing tradition of assuming a logical semantic system, basic to human thought, that is quite distinct from language. The "propositional module" which he proposes as part of the language acquisition device "provides a means to represent propositions in an 'inborn language of thought.'" He builds into this module the ability to indentify logical predicates, arguments and modifiers, as well as contrasts between actor and theme. In his proposed version of language acquisition, after the child has learned to associate specific words (as Ns, IVs, TVs) with the specific semantic predicate types, a requirement that everything fit into one complete proposition (O'Grady's Completeness Requirement and Dependency Requirement) allows him to impose the hierarchical relationships found in the semantic form on the syntactic form (sentence).

I do not completely disagree with this approach; at times I have stated my belief that syntax is little more than a frozen, stylized and simplified version of semantic form. But I am somewhat concerned that a semantic form so easily converted to a syntactic form would be accepted as "inborn" by someone arguing against specific nativism. The type of semantic form proposed by O'Grady

(and most other linguists), while not exactly a syntactic form, can easily be converted to one by relatively simple algorithms.

As an alternative, I suggest that a child must learn semantic relationships--that they are not inborn. The LEARNER is built in line with this suggestion; it has no a priori information about dependency relationships between items, whether words or semantic entities.

Having thrown out the easy solution, I must now search for an alternative. One possible approach to the learning of dependency relationships is that of automatically imposing a hierarchy on any sequence of items. The simplest version of this algorithm would be to impose a single randomly selected direction of dependency on all inputs, creating exclusively right or left branching structural descriptions. At later stages of learning, subcategorization would indicate which items were the correct regents. If the original direction of branching were incorrect, it could be reversed--changing all right branching trees to left branching or vice versa. But, unfortunately, while most languages are primarily right or left branching, not all are exclusively so, resulting in some problems of detail for this approach.

A similar approach is simply to impose regent-dependent hierarchies on all sequences of items. Again, later learning would indicate the direction of subcategorization, and incorrect initially imposed relationships could be reversed. If all relationships are described in a pairwise manner (as required by the binarity

principle of categorial grammars, for example), the creation or discovery of the dependency hierarchies necessary to most generative grammars should be quite straightforward.

Both of the preceding approaches include the implicit claim that general learning involves the imposing of dependency relationships between items. While few would argue against the necessity for hierarchies in grammar (or even in thought), it is not clear that such dependencies are required for all behavior. This being the case, I would prefer to force LEARNER to build its hierarchies using a different motivation.

The determination of hierarchies provided for in LEARNER is accomplished by the examination of the contents of various combinations of the same general type. A type of analogical reasoning is employed. If one class of item occurs in them all (or more) of the time, this important item is designated the regent. The primary drawback to this approach is that it demands a number of input sequences in order to build its hypotheses, or alternatively, the ability to constantly revise assigned hierarchies in light of new data. But there is a certain logic in assuming that an item showing up constantly in a type of sequence or combination will be basic to that sequence.

ANALOGY.[1] The fourth and final inherent learning ability built into LEARNER is the ability to form generalizations by analogy. Analogical thinking is clearly a basic human learning strategy. However, it is also clear that generalization is not given free rein

in learning; otherwise, we would expect exceptionless grammars, exceptionless thought patterns and generally exceptionless behavior.

For this reason, FLRN follows the lead of others (recently, for example Berwick 1985 and O'Grady 1991) in limiting the use of generalization. Analogical reasoning may be used whenever possible--but only the most conservative generalizations consistent with the input are permitted.

THE FOCUS MODULE. The FOCUS module modifies the initial input. It is included in FLRN to model perceptual limitations or perceptual strategies relevant to language use or language learning. The following is a sampling of the types of rules which might be included in this module: (These rules are not original with me, but rather follow ideas presented in Slobin 1967,1971; Grice 1975; and various works of Piaget.)

(a) Learning strategies. For example, pay attention to the beginning of words.

(b) Attention or perceptual limitations. For example, ignore unstressed words or word final consonants.

(c) Pragmatic predisposition. For example, expect the message in the sentence to be relevant to the discourse topic.

(d) Goals. For example, ignore all messages not concerned with food.

(e) Assumed innate abilities. For example, divide the input string into separate words according to specific juncture markings.

(f) Maturational limitations. Ignore all messages not directly concerned with here and now.

In short, any claim concerning the form of the input as it is processed by the PARSER or the LEARNER should take the form of FOCUS rules.

The phonological component of the grammar is included in the FOCUS module, following the logic that a hearer does not focus on fine phonetic detail when parsing but rather focuses on the distinctive units (phonemes and words) of the utterance.

The phonology of a language must be learned. To that end, the FOCUS module will include a phonology acquisition system if FLRN is to be used to model learning from phonetic inputs (the ideal approach).

Each FOCUS rule is made up of the following parts:

(number ifs thens)

The number is a guide to the ordering of the rules. The ifs is the set of conditions which must be met in order for the rule to apply. The thens is the set of situations effected by the rule. The ifs and thens may be null lists, single item lists, or multiple lists.

The rules of FOCUS may vary with the state of the system. A rule in FOCUS which is part of a maturational or learning system would include among its conditions specific requirements describing the stage at which the rule takes effect.

The FOCUS rules modify the initial input before sending it on to the PARSER and the LEARNER. However, the initial pre-FOCUS input may be referred to by rules in the GUIDE, so it should generally be retained in the current memory of RECORDER.

THE GUIDE MODULE. The GUIDE is similar to the FOCUS module in that it is nothing more than a set of rules. In fact, the rules of GUIDE have exactly the same form as those of FOCUS:

(number ifs thens).

The function of the GUIDE however is significantly different. The rules in it are the final word; they take precedence over any other rules in FLRN. They may tell the program to ignore the FOCUS rules, or they may extend the use of analogy beyond its normal limits; in fact, they may guide FLRN in any direction desired. They are the final authority for LEARNER, FOCUS and RECORDER, as well as for the ASTR portion of the PARSER. (The other parts of the PARSER--LXBR, PRSR, CHKR--are inaccessible to GUIDE.)

The GUIDE rules are claims about universals of language and/or learning. Such claims may include rules which are exceptionless, generally true, or statistically likely.

Exceptionless rules are the ideal. Such rules are clear statements of hypotheses, easily supported or disproven by examination of their effects.

Rules which are generally true, while inferior as hypotheses, may be more typical of language learning. These rules, when written in GUIDE, may be ignored if another direction is indicated by input. Such rules simply allow experience to overrule a universal claim.

Rules which are statistically likely are inferior as hypotheses for the same reason as rules that are generally true. They are different only in that their exceptions are randomly determined on

the basis of statistics rather than caused by experience. Since there is no well defined principle behind their exceptions, these rules must be considered inferior to those with clear explanations for their failure to apply.

The GUIDE rules are constantly available to LEARNER (and other relevant modules). They may be considered computational daemons, applying globally when the right conditions are met.

The power and pervasiveness of GUIDE rules would make it easy to misuse FLRN by putting all claims into the GUIDE. However, this module has been designed to model innate universals of language and learning. Hypotheses about other aspects of language should be tested by the appropriate module: FOCUS for perception and attention, ASTR and FOCUS for pragmatics, ASTR for parsing strategies, etc. Because of the narrower focus of these other modules, proposed hypotheses can be subjected to a more careful examination in a less powerful version than they would be if included in GUIDE.

THE RECORDER MODULE. The RECORDER is simply FLRN's area for storage. A future goal is to organize RECORDER in accordance with our knowledge of human memory, but the current version is nothing more than a collection of two types of information: the current grammar and experience.

The current grammar always takes the form of a lexicase grammar; in other words, all of the grammatical information is found in the fully specified lexical entries of the language (the DICT).

An equivalent notation comprises a minimally specified lexicon and a set of generalizations or lexical rules (as described in the discussion of LXBR in chapter 2) which can be applied to provide fully specified lexical items.

The current grammar would normally also include interpretive rules such as LRs and CRs (as described in chapter 2 under the discussion of CHKR).

Experience refers to previous inputs or states of FLRN. The current version of FLRN is set up to separate three types of experience:

(1) The current memory. This contains only the current input (phrase + situation).

(2) The hot memory. This contains items which the LEARNER has direct access to.

(3) The cold memory. This contains other items.

The user may set the program to store specific types of items. For example, the current memory may accept only pre-FOCUS versions of the input, only post-FOCUS versions, or both.

As the choices made in setting up the RECORDER may result in very different access to quite different types of experience, it is very important to be consistent when comparing hypotheses.

NOTES FOR CHAPTER 3

1. The ability labeled "analogy" here is actually much more basic than the simple ability to form analogies. It is the ability to recognize a pattern and impose that pattern on new input. Such an ability could be labeled with a variety of names: analogical reasoning, generalization, rule extension, deduction from example, or prototype directed thinking, to list just a few.

CHAPTER 4

SOME EXAMPLES OF FLRN'S USE

In the earlier chapters of this report, I have described a computer program for the modeling of language and language change. That program consists of a limited syntactic core and some extremely simple learning functions. It includes a number of rule applying modules so that modifications can be made. Such additions to the model are claims about language or learning, hypotheses that should be tested before being made a permanent part of the model. In this chapter, I will document the use of FLRN in the examination of some simple hypotheses.

In order to insure consistency in these tests, I will use the following outline in reporting each of them.

I. MODIFICATIONS TO FLRN.
II. THE INPUT.
  A. Input form.
  B. Input sequence.
III. THE FOCUS RULES.
  A. The phonological component.
  B. Perceptual limitations.
  C. Perceptual strategies.
IV. THE ASTR RULES.
V. GUIDE RULES.
  A. Rules pertaining to input.
  B. Rules for FOCUS.
  C. Rules for ASTR.
  D. Rules for LEARNER.
  E. Rules for RECORDER.
VI. RECORDER ORGANIZATION.
VII. RESULTS.

TEST ONE: A SIMPLE LANGUAGE LEARNER FOR ENGLISH. In this example, I will use FLRN to examine a much simplified model of language learning. The model makes the following claims:

The first stage in the learning of syntax is a naming stage. The learner, at this stage, pays attention to objects in the environment and the language forms which name them. The learner is incapable of understanding long input forms so focuses on short units at the end of sentences.

The second stage is similar to the naming stage, but at this stage the learner begins to build a vocabulary of forms naming actions or descriptive predicates. At this stage, the perceptual abilities are improved to the point that the learner can recognize recurring chunks of language input in somewhat longer utterances.

The third stage in this simplified model is a "pivot grammar" in which certain words assume the role of heads. At this stage the learner's lexicon consists of words classified by two different systems: (1) an "object-action-attribute" distinction based on meaning, the result of the first two stages of learning, and (2) a "pivot-dependent" distinction in which the classification is based on comparative frequency of words.

A fourth stage involves a collapsing of the two separate systems of classification into one syntactic system.

This model is based on several of the ideas presented in Braine 1963, which describes a "pivot grammar." It also makes use of the idea that objects are coded as nouns and actions as verbs, an idea as old as Plato but more recently discussed in Langacker 1987.

TEST ONE: THE REPORT.

I. Modifications to FLRN. The basic FLRN is used. The morphological clue analyzer portion of ASTR is turned off (following an assumption that a learner at these early stages will not use morphological clues in setting up word classes).

II. The input.

A. The input form. Throughout the learning stages the input will have this form:

```
( ( label string-of-sounds)
  ( speaker hearer context message-communicated) ).
```

A typical example is the paired phrase and situation in (1):

```
(1) ( ( (Can you see the bird?)
        (knyu:  si:  dhe.  br.::d) )

      ( (mommy)
        (bobby-me)
        ( (sitting in the park) )
        ( (COMMAND
            (ACTION observe
              (OBJECT animate small feathered
                gray&white flyer water-splashing
                crumb-eating))) ) ).
```

The first two lines of this example are the phrase. An English sentence "Can you see the bird?" is the label. It is provided for ease of reference. The second part of the phrase is a more or less phonetic representation of the input. How much "more or less" is important. In this test, all of the input was broken into syllables. Such divisions in the string-of-sounds constitute assumptions about perception. Users must examine all inputs carefully to determine what assumptions underlie them since the program will not provide a test for claims built into the input.

The situation could contain a good deal more information than is included in this example. The speaker, hearer, and context in this example are simply labels. It is assumed that the LEARNER will not make use of the speaker, hearer, and context information during this test. It will, however, make extensive use of the message-communicated. This is written as a predicate-argument type of logical form.

B. The input sequence. Some order has been imposed on the inputs in order to decrease the amount of time necessary for the test. Those inputs most useful for naming are provided first and they are grouped around items named. However, a few inputs which are assumed to be useless for the stage being modeled are included randomly in order to check the ability of the system to ignore irrelevant input.

The first group of inputs includes phrases (and associated situations) with labels like those in (2):

(2) Look at the bird.
That damn cat's chasing the bird again.
Can you say bird?
Isn't that a pretty bird?
Of all the bird-brained stupid idiots!
Pet the nice birdie.
Ooh that's a big one, isn't it?

At the second stage, the input phrases will correspond to the labels in (3):

(3) The cookies are all gone.
No, you can't have any more ice cream.
Look at that cat run!
Uncle John is coming tomorrow.
Play with your car now.
Those cookies are hot.

At this stage the input phrases will generally include forms for actions (run, play) or descriptions (all gone, more, hot) in addition to names for objects.

III. The FOCUS rules.[1]

> Rule 1. Ignore all of the input string-of-sounds except groups consisting of a single stressed syllable and groups consisting of a single stressed syllable followed by a single unstressed syllable.
>
> Rule 2. Ignore all of the input in the string-of-sounds except the last group (as defined by Rule 1).
>
> Rule 3. Ignore all except those items labeled OBJECT in the message-communicated.
>
> Rule 4. Ignore all of the input in the string-of-sounds except the last group (as defined by Rule 1) or a group which corresponds to the form of a word in DICT.
>
> Rule 5. Ignore all except those items labeled OBJECT, ACTION, or ATTRIBUTE in the message-communicated.

A. The phonological component. Since this test is primarily concerned with the acquisition of a simple syntax, it ignores the problems associated with phonology. It does not include a phonological component.

B. Perceptual limitations. The perception of sounds is assumed to develop from an initial ability at stage one to retain only a single stressed syllable or a single stressed syllable followed by one unstressed syllable.

This FOCUS rule (Rule 1) removes all other units from the input. When applied to a string-of-sounds as in (4), it produces the output form in (5):

(4) (knyu: si: dhe. br.::d)

(5) (knyu: si: si:dhe. br.::d)[2]

C. Perceptual strategies. The FOCUS rules describing perceptual strategies are a developmental system. At the earliest stage, only the last perceived unit in the string-of-sounds is examined (Rule 2). FOCUS rules will also modify the message-communicated portion of the situation. At the first or naming stage, only those items labeled OBJECT in the message-communicated will be passed on (Rule 3).

If (6) is the input (1) after the application of the FOCUS rule limiting perception (Rule 1), the perceptual strategies at this stage (Rules 2 and 3) would create an output as in (7):

```
(6) ( ( (Can you see the bird?)
        (knyu: si: si:dhe. br.::d) )

      ( (mommy)
        (bobby-me)
        ( (sitting in the park) )
        ( (COMMAND
            (ACTION observe
              (OBJECT animate small feathered
                gray&white flyer water-splashing
                crumb-eating))) ) ).

(7) ( ( (Can you see the bird?)
        (br.::d) )

      ( (mommy)
        (bobby-me)
        ( (sitting in the park) )
        ( (OBJECT animate small feathered
            gray&white flyer water-splashing
            crumb-eating) ) ).
```

A later FOCUS rule (Rule 4) will allow for the recognition of forms wherever they occur in the string-of-sounds if those forms are in the DICT.

If the word (br.::d (+ object) (+ animate) (+ flyer) ...) has been learned (that is, it is in the DICT), an input like (8) is modified to produce (9) by Rules 1-4.

```
(8) ( ( (The bird is eating the cat!)
        (dhe. br.::d  iz  i:d  ing  dhe.  kaet) )

      ( (mommy)
        (bobby-me)
        ( (playing in the living room) )
        ( (EXCLAMATION
            (ACTION devour
              (OBJECT animate small feathered
                green&yellow flyer crumb-eating)
              (OBJECT animate small furry feline
                clawed tiger-striped))) ) )

(9) ( ( (The bird is eating the cat!)
        ( br.::d   kaet) )

      ( (mommy)
        (bobby-me)
        ( (playing in the living room) )
        ( (OBJECT animate small feathered
            green&yellow flyer crumb-eating)
          (OBJECT animate small furry feline
               clawed tiger-striped) ) ) )
```

At a later stage, items labeled ACTION or ATTRIBUTE in the message-communicated will be passed on, along with those labeled OBJECT (Rule 5).

```
(10) ( ( (That stove is hot!)
         ( dhaet  sto:v  iz  'hat ) )

       ( (mommy)
         (bobby-me)
         ( (visiting a hill-billy)
         ( (ATTRIBUTE unpleasantly-warm
             (OBJECT inanimate pot-bellied metal
               cooker) ) ) )
```

```
(11) ( ( (That stove is hot!)
         ( 'hat ) )

       ( (mommy)
         (bobby-me)
         ( (visiting a hill-billy)
         ( (ATTRIBUTE unpleasantly-warm)
           (OBJECT inanimate pot-bellied metal
             cooker) ) )
```

Note that the Rules 3 and 5 do not maintain or impose hierarchies on the items they pass on. Such hierarchies are imposed by the ranking function within the LEARNER.

IV. The ASTR Rules. The ASTR contains no pragmatic rules. As stated above, the morphological clue analyzer has been disabled.

The ASTR's only active function in this test is to simplify parsing. Item (12) is a phrase after parsing (that is, after potential parses have been examined by CHKR). The phrase at this point consists of three parts:

(label string-of-sounds parses).

Each member of parses includes a status code as its car, followed by the parse.

```
(12) ( (That stove is hot!)
       (sto:v  'hat)
       ( (ok ((sto:v) 'hat) ) ) )
```

The ASTR will examine the entries for the parse's words in the DICT. As a result of LEARNER's classification, the lexical entries look something like this:

```
(13) (sto:v (+ OBJECT) (+ metal) (+ pot-bellied) ...)

(14)  ('hat (+ ATTRIBUTE) (+ warm) ...)
```

The phrase structure rule writer will then write phrase structure

rules (as in (15) and (16)) for generating similar parses, using the information in the lexical DICT entry concerning the categories of the words. (Given lexicase restrictions on structural descriptions, this is quite an easy task.)

(15) ((AttributeP) (ObjectP) (Attribute))
AttributeP --> ObjectP Attribute
Attribute = a lexical item marked [+ATTRIBUTE]

(16) ((ObjectP) (Object))
ObjectP --> Object
Object = a lexical item marked [+OBJECT]

V. The GUIDE Rules.

Rule 1. (ASTR) Phrase structure rules refer only to major categories OBJECT, ATTRIBUTE, ACTION, pivot or dependent.

Rule 2. (LEARNER) Associate features from a single unit in message-communicated with a single group in the string-of-sounds.

Rule 3. (LEARNER) Eliminate from the string-of-sounds any form found in DICT. Also eliminate its meaning from the message-communicated.

Rule 4. (LEARNER) If a possible new word has the same form as a word in DICT, do not add the new word to DICT. Instead, replace the word in DICT with a word whose feature matrix is the intersection of the matrices of the two words.

Rule 5. (LEARNER) Label the head of a phrase as [+pivot]. Label its dependents [+dependent].

Rule 6. (RECORDER) Store all final (after LEARNER or PARSER) versions of the string-of-sounds in the hot memory. Store all final outputs (phrase plus situation) in the cold memory.

A. Rules pertaining to input. None.

B. Rules for FOCUS. None.

C. Rules for ASTR. There are no GUIDE rules for either the pragmatic or morphological clue portions of ASTR.

Guide Rule 1 limits the phrase structure rule writer so that its rules refer only to major categories: OBJECT, ATTRIBUTE, ACTION, pivot and dependent. Without this limitation, ASTR's phrase structure rule writer could write rules covering (12) through (14) as rules like (17) or (18):

```
(17) ((WarmP) (MetalP) (Warm))
       WarmP --> MetalP Warm
       Warm = a lexical item marked [+warm].

(18) ((XP) (AP) (X))
       XP --> AP X
       X = a lexical item marked [+ATTRIBUTE, +warm, ...]
```

D. Rules for LEARNER. There are four GUIDE rules for LEARNER. Rule 2 states that classification associates information in the message-communicated of the situation (after FOCUS) with a form in the string-of-sounds (after FOCUS). It applies to inputs like (7), repeated here, to create lexical entries like (19) to be stored in the DICT.

```
(7) ( ( (Can you see the bird?)
        (br.::d) )

      ( (mommy)
        (bobby-me)
        ( (sitting in the park) )
        ( (OBJECT animate small feathered
            gray&white flyer water-splashing
            crumb-eating) ) ).

(19) (br.::d (+ OBJECT) (+ animate) (+ small)
       (+ feathered) (+ gray&white) (+ flyer)
       (+ water-splashing) (+ crumb-eating) )
```

Rule 3 adds an additional learning ability to LEARNER, the ability to eliminate known units from the input in order to isolate other units. For example, if the word (br.::d ...) is already in the DICT, Rule 3 allows the LEARNER to ignore in (9) both its form in string-of-sounds and its meaning in message-communicated. The LEARNER can then process the simpler (20).

```
(9) ( ( (The bird is eating the cat!)
        ( br.::d   kaet ) )

      ( (mommy)
        (bobby-me)
        ( (playing in the living room) )
        ( (OBJECT animate small feathered
            green&yellow flyer crumb-eating)
          (OBJECT animate small furry feline
               clawed tiger-striped) ) ) )

(20) ( ( (The bird is eating the cat!)
         ( kaet ) )

      ( (mommy)
        (bobby-me)
        ( (playing in the living room) )
        ( (OBJECT animate small furry feline
            clawed tiger-striped) ) ) )
```

Rule 4 provides for a type of generalization. It states that two lexical entries with the same form may be combined to form a single entry for which the feature matrix is the intersection of the matrices for the two entries. From the inputs (7) and (9), the DICT could contain two entries with the same form, as (21) and (22).

```
(21) (br.::d (+ OBJECT) (+ animate) (+ small)
       (+ feathered) (+ gray&white) (+ flyer)
       (+ water-splashing) (+ crumb-eating) )

(22) (br.::d (+ OBJECT) (+ animate) (+ small)
       (+ feathered) (+ green&yellow) (+ flyer)
       (+ crumb-eating) )
```

In such cases, Rule 4 results in the replacement of the earlier entry in the DICT with a single new entry like (23).

```
(23) (br.::d (+ OBJECT) (+ animate) (+ small)
       (+ feathered) (+ flyer) (+ crumb-eating) )
```

Rule 5 simply provides labels for the words in parsed sentences. This rule would add the feature (+ pivot) to the entry ('hat ...) in the DICT and the feature (+ dependent) to the entry (sto:v ...) when processing the parse in (12).

```
(12) ( (That stove is hot!)
       (sto:v  'hat)
       ( (ok ((sto:v) 'hat) ) ) )
```

E. Rules for RECORDER. The string-of-sounds from all previous inputs must be available to the LEARNER so that ranking may take place. The version stored in the hot memory is the string-of-sounds after modification due to FOCUS rules.

Ranking works like this: If the string-of-sounds includes two or more items, the more frequently occurring item will be labeled the head with the other(s) labeled dependent(s). For example, if (12) is sent to the LEARNER, the ranking function will compare the frequency of the two items br.::d and kaet in the hot memory. If br.::d is more frequent, it will be labeled the head (and [+pivot] by Rule 5) and kaet will be labeled the dependent (and [+dependent], again by Rule 5).

```
(12) ( (That stove is hot!)
       (sto:v  'hat)
       ( (ok ((sto:v) 'hat) ) ) )
```

VI. The RECORDER organization. The current grammar is stored in DICT. It is a list of fully specified lexical items.

As for experience, the current memory contains only the currently modified state of the original input; earlier states are lost. The hot memory contains only successful parses which must be available for the LEARNER's ranking function. All outputs are stored in the cold memory to facilitate tracing by the user.

VII. THE RESULTS. What a mess! In short, this test failed. One reason for its failure was a lack of precision in the rules describing the model. A major advantage of using a computer program such as FLRN for the examination of hypotheses is that it forces explicit formulations of the claims made by a model.

The grammar developed in this test does not at all resemble the pivot grammar described in Braine 1963. There were a large number of inputs provided during the naming stage. The excessive number of such inputs resulted in the labeling of words for OBJECTS as heads (+ pivot) by the ranking function, rather than the functional words called pivots by Braine. A clearer distinction between the naming stage and later stages, even to the extent of clearing the hot memory at the end of the naming stage, might solve this problem. However, it is also quite possible that the number of different OBJECTS is just as limited as the number of ACTIONS and ATTRIBUTES in the human learner's input. In that case, if FLRN's inputs are the same as a human's, even the erasing of the hot memory at the end of the naming stage will not lead to the predicted classification of pivots and dependents.

TEST TWO: HOLISTIC DRIFT AND THE DEVELOPMENT OF PREPOSITIONS IN CHINESE. In this test, I will examine a rather extensive theory of language change. Then I will choose a very small section of it to model. In this theory, holistic typology, especially as influenced by accent type, is used to explain the direction of language change.

Sapir (1921) noted that separated languages may develop in similar directions as if they were guided by some inner plan. Venneman 1975 and Donegan and Stampe 1983 are two attempts to use holistic typology as an explanation for this drift.

In their article, Donegan and Stampe discuss the genetically related but typologically opposite Munda and Mon-Khmer language groups. Although the Munda type is similar to that of other Indian languages and the Mon-Khmer type similar to that of other mainland Southeast Asian languages, the authors minimize the effect of diffusion. Sapir (1921, ch. 9) has made the claim that languages are incapable of directly borrowing complex structural features. Donegan and Stampe support this opinion with their claim that "comparative studies have established a breath-taking degree of independence in the evolutions of the individual Munda and Mon-Khmer languages" (1983:338). From this starting point, they seek an explanation for the independent drift that has resulted in similar typologies within each group--and such opposite directions of drift for the two separate groups.

In an examination of word order, they find that new information is typically provided by the modifier (or operator, as opposed to the operand, using the terms they borrow from Vennemann 1975). They state, following Bolinger 1958, that new information takes the phrasal accent. They then link phrasal accents with word accents, since the two coincide in phrases consisting of a single word.

At this point, they have provided a reason for the tendency of a language to drift in a particular direction. The interaction of accent and word order results in pressure to develop one consistent order of heads and modifiers. This accounts for the convergence of the Munda languages (with the Indian typology) in spite of little evidence of direct borrowing of inflectional morphemes.

Donegan and Stampe explain other characteristics of the two primary types of languages in terms of rhythm. By associating prominence in accent with prominence in time (that is, greater duration), they are able to characterize two basic rhythm types: one in which the beat is counted in terms of syllables or morae and one in which the beat is counted in terms of words (actually stressed syllables in words). These two rhythm patterns, the first associated with operator-operand (OV) word order and the second with operand-operator (VO) order, provide explanations for the fact that OV languages have certain phonological characteristics (typically monophthongal vowels and simple consonant clusters) while VO languages have others (typically diphthongal and reduced vowels with non-geminate consonant clusters).

The two language types (as exemplified by Munda and Mon-Khmer languages) have the characteristics outlined in Table 4.1 (Donegan and Stampe 1983:337).

	OV (Munda) (falling accent)	VO (Mon-Khmer) (rising accent)
Phrase Accent:	Initial	Final
Word Order:	Variable	Rigid
Syntax:	Case, verb agreement	Analytic
Morphology:	Agglutinative Suffixing Polysynthetic	Fusional Prefixing or isolating
Timing:	Isosyllabic or isomoric	Isoaccentual
Consonatism:	Stable Geminate clusters	Shifting Non-geminate clusters
Vocalism:	Stable Monophthongal Harmonic	Shifting Diphthongal Reductive
Tone:	Level	Contour

Table 4.1. Characteristics of "Falling" and "Rising" Typologies.

Donegan and Stampe claim that accent pervades all levels of language structure and that as a result it provides a logical explanation for holistic drift. They make no claims as to the initial triggering factor in language change. It is not at all clear what the first step would be in the change from a relatively stable proto-language of one type to a later language of a different type. How is a different accentual pattern brought into a language?

In the test which follows, I will be examining drift in the direction from an OV to a VO pattern. I will assume that words with new accent patterns are brought into the language. These words influence the accent pattern of native words. The new accent brings about phonological changes that result in the loss of information bearing forms. The language develops new ways of communicating this information.

This test will not examine all of Donegan and Stampe's theory of drift. Rather, the test will look at only one small part of their hypothesis.

THE PROBLEM. The Chinese languages form most (if not all) the members of the Sinitic branch of the Sino-Tibetan language family. Chinese is generally of the rising accent typology. Chinese direct objects usually follow the verb (although most other dependents typically precede their heads). The languages are prepositional. Chinese is significantly different from the languages of the Tibeto-Burman family, the other main branch of Sino-Tibetan. With the exception of Karen, all of those languages are OV in type.

The typology of proto-Sino-Tibetan is thought to correspond to that of the Tibeto-Burman languages. It is assumed to have been OV (Benedict 1972:4) with postpositions (DeLancey 1990:78).

Chinese seems to have adjusted very neatly along the lines outlined by Donegan and Stampe. The phonological features of contour tones, word timed accent, and monosyllablic words are well documented for the family. As early as Archaic Chinese, the vowels

had developed an unusual degree of diphthongization (Benedict 1972:179). The word order of the modern dialects, while not typical of VO languages, does show the basic verb-object order and includes the use of prepositions rather than postpositions.

At some point in its history, Chinese made the switch from postpositions to prepositions. This change can be quite simply described in terms of drift toward a holistic VO type: In rising accented languages, elements which follow the phrasal accent (the position of postpositions in such languages) are reduced or lost. With nothing to carry the functional load of the lost postposition, the language adjusts by adding new prepositions. For Chinese, there is no mystery about the source of the new propositions. They are generally identical in form and similar in meaning to verbs of the language; in fact, they are often labeled co-verbs rather than prepositions by Sinologists.

Using FLRN, I will attempt to model the change from postpositions to prepositions in Chinese. In this model, I assume that the switch was relatively quick. I assume a generation (G-1) with postpositions immediately followed by a generation (G-2) with prepositions. I do not allow for an intermediate stage in which both postpositions and prepositions with the same meanings are available; in other words, I presume there is no stage at which a single phrase might have both a preposition and a postposition.

In this sample test, I will define functional load as the need to express specific meanings. The specific meanings are those

associated with prepositions, meanings which can be neatly described in terms of localistic features (Starosta 1982 is such a description for Mandarin).

TEST TWO: THE REPORT

I. Modifications to FLRN. The basic FLRN is used. The morphological clue analyzer portion of ASTR has been turned off.

II. The input.

A. The input form. The input structure is the usual paired phrase and situation.

```
( ( label string-of-sounds)
  ( speaker hearer context message-communicated) ).
```

In this test, a typical input looks like (24):

```
(24) (
       ( (from -I come from the store)
         (ngoh poutau -from leih) )

       ( (ngoh)
         (neih)
         ()
         ( ((1) (2) 3)
            (1 (ngoh (+ N) (+ PAT) (+ actr) ...))
            (2 (poutau (+ N) (+ LOC) (+ sorc) ...))
            (3 (leih (+ V) (1 ((+ PAT))) (2 ((+ LOC))))) )
       ).
```

The input phrases consist of Cantonese sentences in which the current prepositions are replaced by postpositions (which not accidentally look like English prepositions). The label consists of the postpositional meaning (if there is one) and an English gloss of the phrase. The string-of-sounds is divided into words. Details of phonology are ignored; in fact, not even adequate phonemic detail is included since tones are not adequately marked.

The input situations include fairly detailed semantic structures in the message-communicated. The format used for these semantic structures is taken from lexicase discussions of semantic interpretation (Pagotto 1985, Lindsey 1985, and Lee 1989). Associated with this approach is an assumption that an interpreted, semantic structure is simply a somewhat augmented version of the syntactic structure, with semantic relationships corresponding to syntactic dependency relationships.

B. The input sequence. The inputs are ordered according to the number of words in the string-of-sounds. These inputs vary in length from two word strings to much longer strings. Item (25) provides some typical examples with English glosses:

(25) keuih leng
-she is pretty

ngohdeih mhleng
-we're not good-looking

ngohdeih dou gou
-we're all tall

ngohdeih choh foche
-we ride a train

keuih hohkhaauh -from leih
-he comes from school

ngo ho mhhoyih nisyu -from syuhn -by heui gwongjau a
-can I go to Canton from here by boat?

III. The FOCUS rules.

Rule 1. Delete all postpositions from the string-of-sounds.

A. The phonological component. Although this test examines the results of phonological erosion, the erosion is assumed

to have had a specific effect (as stated in Rule 1). There is no phonological component.

B. Perceptual limitations. The only FOCUS rule concerns the loss of postpositions. This loss is phonologically conditioned, but the current model is not concerned with such details. Rule 1 simply states to delete (ignore) postpositions (which are marked by an initial -) in the string-of-sounds. Its effects are obvious.

C. Perceptual strategies. It is not clear whether Rule 1 involves a perceptual limitation or a perceptual strategy. I assume that question is not important for this model.

IV. The ASTR Rules. The ASTR contains no pragmatic rules to assist in parsing. In addition, given the isolating nature of the Chinese language, the morphological clue analyzer is considered of very limited use and is disabled.

V. The GUIDE Rules.

> Rule 1. (ASTR) Phrase structure rules refer only to major categories noun, adjective, determiner, verb, adverb, coordinating conjunction, and postposition or preposition.
>
> Rule 2. (LEARNER) Assume a parse in which dependency relationships are the same as those in the message-communicated.
>
> Rule 3. (LEARNER) If a possible new word has the same form as a word of the same major category in DICT, do not add the new word to DICT. Instead, replace the word in DICT with a word whose feature matrix is the intersection of the matrices of the two words.
>
> Rule 4. (LEARNER) All localistic meanings in the message-communicated must be associated with a word in the string-of-sounds. If there is no such word in the

string-of-sounds, search the DICT for a word to carry such meaning. Make a new entry for that word, replacing its major category with [+P].

Rule 5. (RECORDER) Only the most recent version of the input is stored in the current memory. Nothing is stored in the hot memory. All outputs are stored in the cold memory.

A. Rules pertaining to input. None.

B. Rules for FOCUS. None.

C. Rules for ASTR. There are no GUIDE rules for either the pragmatic or morphological clue portions of ASTR. Rule 1 limits the phrase structure rule writer so that its rules refer only to a few major categories: (+ Noun), (+ Verb), (+ Adj), (+ Adv), (+ Det), (+ P), (+ Ccnj). This follows from a regular assumption within the lexicase theory that there are a limited number of basic categories of words.

D. Rules for LEARNER. The grammar is provided quite directly by imposing the same dependency relationships on the phrase as those found in the message-communicated (Rule 2). A number of unproven assumptions about semantic form and language learning are hidden in this approach.

The LEARNER must create a means of marking localistic semantic features (from the message-communicated of the situation), even if the input phrase does not provide a clear indication of what that means might be.

Rule 3 eliminates homonyms of the same major category (noun, verb, etc.) from the lexicon. This rule is necessary in order to

prevent a DICT containing 20 or 30 homonyms, each carrying a different localistic meaning. It is probably worded too strongly; it seems likely that humans can handle at least a little more homonymy than this.

When the input phrase (as modified by the FOCUS rules) does not include a word with the meanings normally carried by a postposition, Rule 4 starts a search for some way of carrying the functional load. The search is limited to the words already in the DICT. (Of course, it is possible to imagine other possibilities: this search could result in the creation of new forms or in the borrowing of forms from other languages.)

Rule 4 further states that the new form used to carry the meanings associated with the lost postpositions will be of the major category [+P] (that is, a preposition or a postposition).

E. Rules for RECORDER. All outputs of LEARNER are stored in the cold memory; previous inputs are not available to the LEARNER (Rule 5).

VI. The RECORDER organization. The current grammar is stored in DICT. It is a list of fully specified lexical items.

As for experience, the current memory contains only the currently modified state of the original input; earlier states are lost. The hot memory is empty; the LEARNER has no access to any earlier inputs. All outputs are stored in the cold memory to facilitate tracing by the user.

VII. THE RESULTS. There were no surprises--which is not surprising given the amount of control included in the GUIDE rules. The G-2 generation was able to successfully replace the "lost" postpositions with prepositions. The new prepositions were derived from verbs with motion or positional meanings. These new [+P] words were prepositions rather than postpositions because of ordering features for the verbs from which they were derived. When no verb (or other word) with the appropriate semantic features was found, no preposition was derived for that meaning.

The DICT did include a few examples of nouns with inherent localistic features. This resulted from cases where the particular noun in question occurred only once in the inputs. Some additional requirement for generalization seems necessary.

There was also a problem concerning the mutual intelligibility of the G-1 and G-2 languages. Sentences produced using the G-2 grammar were not understood by a parser using the G-1 grammar if the parsing was done for complete sentences. If the complete sentences were broken into shorter units (corresponding to phrases), the G-1 parser was able to effectively parse and interpret the sentences. For example, a sentence with a preposition like item (26) was problematic when considered as one sentence. However, if the parser was allowed to divide it into parts as in (27), the sentence could be parsed and interpreted (although the details of the interpretation would be slightly different from those in a G-2 parser's interpretation, as indicated by the glosses).

(26) keuih choh feigei leih
-he by plane came

(27) keuih choh feigei
-he took plane

leih
-he came

An unanticipated problem was the inability of the G-2 parser (with the LEARNER turned off) to successfully parse and interpret input produced using the G-1 grammar, whether or not the FOCUS rule deleting postpositions was implemented.

I had assumed that the G-1 and G-2 dialects to be mutually intelligible. The inability of the G-2 generation to parse G-1 input makes this a questionable assumption.

NOTES FOR CHAPTER 4

1. During the development of FLRN, I have vacillated between a PROLOG type rule applier and a computationally simpler system using LISP conditionals. The exact form of these rules depends on which of those two systems is used. The English versions given here can be easily translated for use with either system. (They are also easier to read.)

2. I have simplified the FOCUS rules somewhat for expository purposes. The actual Rule 1 picks out both the stressed syllable and the stressed syllable plus unstressed syllable, with the latter listed after the former in the output.

CHAPTER 5

CONCLUSION

It is with some trepidation that I submit this report as a contribution to the field of linguistics. What it describes is, after all, a language processor less efficient even than the transition network parsers described by Woods (1970) some 20 years ago. Moreover, the learning portion of the program is only an outline of a system, far less substantive than the language learning models proposed by Berwick (1985) and Selfridge (1986).

Yet I feel little need to apologize. The work in computational linguistics over the past two decades has generally ignored issues of linguistic theory. With such great rewards awaiting the creator of an efficient natural language processor, it is not at all surprising that theoretical issues should be set aside if they are not immediately relevant to the current engineering project.[1]

This dissertation, on the other hand, has had as its primary focus the examination of theoretical issues. The computational tool it describes is specifically designed to examine theoretical claims.

The development of the FLRN PARSER required a close examination of the lexicase syntactic theory which it uses, an examination that resulted in some suggestions for the improvement of the lexicase grammatical theory. Other theoretical frameworks would benefit from similar attempts to codify their basic assumptions.

FLRN appears to be a versatile tool useful in testing a variety of types of hypotheses. It is flexible enough to test pragmatic,

semantic, and syntactic systems together in a single integrated model--or it can focus on much smaller claims about language and language change such as those shown in the examples of chapter 4.

I sincerely hope that the program will be a useful addition to the collection of tools linguists can employ when examining claims about the nature of language and learning.

NOTES FOR CHAPTER 5

1. I do not intend to belittle the contribution made by people who utilize linguistic frameworks in the design of their successful programs. But their contributions develop out of different goals from mine. If the purpose of the computational system is not to examine a theory, it is all too often easier to list a small group of exceptions rather than implement an explanatory theory or delve into the reasons for problems. The goal in too many of the natural language processing programs remains efficiency rather than the scientific examination of hypotheses.

APPENDIX A

LXBR SOURCE CODE AND DOCUMENTATION

```
;     LXBR (The Lexicon Builder). The initial input is a lexicon of
;minimally specified lexical entries. All of the rules are applied
;to each member of the initial lexicon. The output of LXBR is a
;list of fully specified lexical entries. Within FLRN, this list is
;referred to as DICT.

(defvar dict)
(defvar status)

(defun lxbr ()
  (bx-dict
    (do ((ll (ax-mlex) (cdr ll)) (rs nil))
      ((null ll) rs)
      (setq rs (append (wdbr (car ll)) rs)))))


;     WDBR (The Word Builder). The processes described under LXBR
;above are accomplished, lexical item by lexical item, in WDBR.

(defun wdbr (word)
  (append
    (wdbr-3 (wdbr-2 (list (wdbr-1 word))))
    (wdbr-5 (wdbr-4 word))))


;     WDBR-1 applies RRs.

(defun wdbr-1 (word)
  (p-rr (ax-rr) word))


;     WDBR-2 applies SRs.

(defun wdbr-2 (words)
  (p-sr (ax-sr) words))
```

```
;     WDBR-3 applies MRs and IRRs.

(defun wdbr-3 (words)
  (do ((ww words (cdr ww)) (rs nil))
    ((null ww) rs)
    (setq rs
      (cons (p-irr (ax-irr) (p-mr (ax-mr) (car ww))) rs))))


;     WDBR-4.  This subfunction applies DRs.  It applies all of the

;DRs as many times as indicated by the variable cycles.

;     It applies only DRs with a certain minimal productivity which

;is given as the last item (a number representing per cent) in the

;P-DR function call.  [Note that if the SHOW option for LXBR is

;activated, the application of WDBR-4 will cause some RR results to

;show an extra time.]

(defun wdbr-4 (word)
  (do
    ((cycles 3 (- cycles 1))
      (old (list word))
      (new (list word)))
    ((or (null new) (= cycles 0)) old)
    (setq new (p-dr (ax-dr) new 70))
    (setq old (append new old))
    (do ((nn new (cdr nn)) (rs nil))
      ((null nn) rs)
      (setq rs (cons (wdbr-1 (car nn)) rs)))))


;     WDBR-5 processes derived lexical items.

(defun wdbr-5 (words)
  (do ((ww words (cdr ww)) (rs nil))
    ((null ww) rs)
    (setq rs
      (append
        (wdbr-3 (wdbr-2 (list wdbr-1 (car ww)))))
        rs)))
```

```
;     P-RR (Apply Redundancy Rules).  The lambda description is the

;same as for P-MR and P-IRR.

(defun p-rr (rules word)
  (do ((rr rules (cdr rr)))
    ((null rr) word)
    (setq word (p-redun (car rr) word))))


;     P-SR (Apply Subcategorization Rules).  Note that P-SR takes a
;list of words as its input.

(defun p-sr (rules words)
  (do ((rr rules (cdr rr)))
    ((null rr) words)
    (setq words (p-sr-1 (car rr) words))))

(defun p-sr-1 (rule words)
  (do ((ww words (cdr ww)) (rs nil))
    ((null ww) rs)
    (setq rs (append (p-categ rule (list (car ww))) rs))))


;     P-MR (Apply Morphological Rules).

(defun p-mr (rules word)
  (do ((rr rules (cdr rr)))
    ((null rr) word)
    (setq word (p-redun (car rr) word))))


;     P-IRR (Apply Inflectional Redundancy Rules).

(defun p-irr (rules word)
  (do ((rr rules (cdr rr)))
    ((null rr) word)
    (setq word (p-redun (car rr) word))))


;     P-DR (Apply Derivational Rules).  The output of individual DRs

;may not be fed directly into other DRs.  Rules with a productivity

;lower than prod are not applied.

(defun p-dr (rules words prod)
  (do ((ww words (cdr ww)) (rs nil))
    ((null ww) rs)
    (setq rs (append (p-dr-a rules (car ww) prod) rs))))
```

```
(defun p-dr-a (rules word prod)
  (do ((rr rules (cdr rr)) (tp nil) (rs nil))
    ((null rr) rs)
    (setq tp (p-deriv (car rr) word prod)
    (if tp
      (setq rs (cons tp rs))
      nil))))


;     REDUN RULES.  Each redun rule is a list:

;          (rule-number left-matrix right-matrix
;              morphological-change)

;Rule-number may be any LISP symbol.  Both left-matrix and right-

;matrix must be matrices.  Morphological-change (which is optional)

;is a list of these items:

;          (old new enviroment).

;Morphological-change may be a list of morphological-changes.


;     P-REDUN (Apply a Single Redun Rule).  A redun rule applies to

a word if the rule's left-matrix is a submatrix (SUBM) of the word's

;matrix.  If the rule applies, the features in the rule's right-

;matrix may be added to those in the word's matrix, provided they

;meet the ;conditions described under AD-FITR below.

;     If all of the features from the rule's right-matrix are

;successfully added to the word's matrix, the phonological form of

;the word may be changed.  This change is handled by P-CHG below.

;     The input to P-REDUN is a word; the output is that word as

;modified by the rule.
```

```
(defun p-redun (rule word)
  (let
    ((left (car (cdr rule)))
     (right (car (cddr rule)))
     (change (cadr (cddr rule)))
     (form (car word))
     (matrix (cdr word))
     (temp nil))
    (cond
      ((subm left matrix)
        (cond
          ((null right)
            (setq status 0)
            (setq word (cons (p-chg change form) matrix)))
          (t
            (setq temp (ad-fitr right matrix))
            (cond
              ((null temp) (setq status 0))
              (t
                (setq status 2)
                (if (s-set-f temp right)
                  (setq word
                    (cons (p-chg change form)
                      (append temp matrix)))
                  (setq word
                    (cons form
                      (append temp matrix)))))))))
      (t (setq status 1)))
    (show (car rule) word status)
    word))


;     CATEG RULES.  SRs are processed as categ rules.   A categ rule

;is a list with the following members:

;          (rule-number left-matrix category-list).

;The rule-number may be any LISP symbol.  The left-matrix is a

;matrix.  The category-list is a list of two member lists:

;          ((right-matrix1 morphological-change1)
;           (right-matrix2 morphological-change2)
;                        ...)
```

```
;Each right-matrix is a matrix.  Each morphological-change is a list

;with the following members:

;          (old new environment).

;Each morphological-change can also be a list of such

;morphological-changes.


;     P-CATEG (Apply a Single Categ Rule).  A categ applies to a

;word if the rule's left-matrix is a submatrix of the word's matrix.

(defun p-categ (rule words)
  (do ((ww words (cdr ww)) (rs nil))
    ((null ww) rs)
    (cond
      ((subm (cadr rule) (cdr (car ww)))
        (setq rs (append (p-cat-1 rule (car ww)) rs)))
      (t
        (setq status 1)
        (show (car rule) (car ww) status)
        (setq rs (cons (car ww) rs))))))

(defun p-cat-1 (rule word)
  (do ((rr (p-cat-2 rule) (cdr rr)) (rs nil) (tp nil))
    ((null rr) rs)
    (setq tp (p-redun (car rr) word))
    (cond
      ((member tp rs 'equal)
        (setq status 0)
        (show (car (car rr)) word status))
      (t
        (setq rs (cons tp rs))
        (setq status 5)
        (show (car (car rr)) tp status)))))

(defun p-cat-2 (rule)
  (do
    ((left (list (car rule) (cadr rule)))
     (cats (car (cddr rule)) (cdr cats))
     (ruls nil))
    ((null cats) ruls)
    (setq ruls (cons (append left (car cats)) ruls))))
```

```
;     DERIV RULES.  Lexicase derivational rules are written as deriv
;rules.  Each deriv rule is a list:
;        (rule-number productivity old-matrix
;          new-matrix morphological-change)
;Rule-number may be any LISP symbol.  Productivity is a measure of
;how frequently the rule should apply, given as a number.

;    P-DERIV (Apply a Single Deriv Rule).  A deriv rule applies to a
;word if the old-matrix is a submatrix of the word's matrix and the
;productivity of the rule is greater than or equal to the prod in
;the P-DERIV call.
;    If a rule applies, it replaces the old-matrix features in the
;word's matrix with those in new-matrix and modifies the word's form
;as indicated by morphological-change.
;     If the rule applies, the output will be the new word.  If it
;does not apply, the output will be nil.
(defun p-deriv (rule word prod)
  (cond
    ((subm (car (cddr rule)) (cdr word))
      (cond
        ((< (cadr rule) prod)
          (setq status 7)
          (show (car rule) nil status)
          nil)
        (t (p-der-1 rule word))))
    (t
      (setq status 1)
      (show (car rule) nil status)
      nil)))
```

```
(defun p-der-1 (rule word)
  (setq word
    (cons
      (p-chg (car (cddr (cddr rule) (car word))
      (mapcar #'ad-new
        (cadr (cddr rule))
        (mapcar #'remove
          (car (cddr rule)) (cdr word)) s-fitr)
       's-fitr))))
  (setq status 6)
  (show (car rule) word status)
  word)


;     SUBM (Submatrix Predicate).  This function is the submatrix

;function used to determine whether a rule should apply.  It tests

;whether all features in matrix one match features in matrix two.

;[Note that SUBM is not strictly a subset predicate since the key

;test is whether or not the features match (as per M-FITR) rather

;than whether they are the same.

(defun subm (one two)
  (do ()
    ((null one) t)
    (cond
      ((member (car one) two 's-fitr))
      ((subm-1 (car one) two))
      (t (return nil)))
    (setq one (cdr one))))

(defun subm-1 (fitr matrix)
  (do ()
    ((null matrix) nil)
    (cond
      ((equal 1 (m-fitr fitr (car matrix)))
        (return t))
      (t (setq matrix (cdr matrix))))))
```

```
;     AD-FITR (Add Features). This function returns a list of

;features from the matrix new which could be added to the matrix

;old. In other words, it returns a list of those features in new

;which (1) are not already in old, and (2) do not conflict with

;those in old. (See M-FITR and M-VAL below for more detail

;concerning which features conflict.)

(defun ad-fitr (new old)
  (do ((nn new (cdr nn)) (rs nil))
    ((null nn) rs)
    (cond
      ((member (car nn) old 's-fitr))
      ((member (car nn) old 'm-fitr)
        (setq rs (cons (car nn) rs))
        (setq status 2))
      (t nil))))


;     S-FITR (Same Feature Predicate). Are one and two the same

;lexicase feature? S-FITR returns t if yes, nil if no.

(defun s-fitr (one two)
  (cond
    ((s-feat (cdr one) (cdr two))
      (cond
        ((equal (car one) (car two)) t)
        (t nil)))
    (t nil)))


;     S-FEAT (Same Feat Predicate). S-FEAT returns t if one and two

;are the same FLRN feat (n.b. not feature!); otherwise, it returns

;nil. Since a feat is the cdr of a feature, it will always be a

;list. S-FEAT recognizes the following types of feats:

;     (1) non-contextual, a list containing exactly one atom,
;     (2) simple contextual, a list containing exactly one matrix,
;     (3) compound contextual or ordering contextual, a list of
;         matrices with or without a position-holding marker __
```

```
(defun s-feat (one two)
  (cond
    ((null one) (null two))
    ((and (equal (car one) '__) (equal (car two) '__))
      (s-feat (cdr one) (cdr two)))
    ((not (listp (car one)))
      (equal (car one) (car two)))
    (t
      (if (s-matrix (car one) (car two))
        (s-feat (cdr one) (cdr two))
        nil))))
```

; M-FITR (Matching Feature Predicate). Does feature one match ;feature two? If one and two have the same value and feat, M-FITR returns a value of 1. If the features have conflicting values for the same feat,it returns nil. Otherwise, it returns a value of 2.

```
(defun m-fitr (one two)
  (cond
    ((s-feat (cdr one) (cdr two))
      (m-val (car one) (car two)))
    (t 2)))
```

; M-VAL (Matching Value Predicate). M-VAL returns nil (0) for ;conflicting values, 1 for matching values, and 2 in all irrelevant ;comparisons. M-VAL is a symmetrical function (i.e., if X matches ;Y, then Y matches X) except for cases involving ? and >.

```
(defun m-val (left right)
  (cond
  ((equal left '+) (m-val-1 right))
  ((equal left '-) (m-val-2 right))
  ((equal left '*) (m-val-3 right))
  ((equal left '+/-) (m-val-4 right))
  ((equal left right) 1)
  (t 2)))

(defun m-val-1 (val)
  (cond
    ((equal val '-) nil)
    ((equal val '+) 1)
    ((equal val '+/-) 1)
    (t 2)))
```

```
(defun m-val-2 (val)
  (cond
    ((equal val '+) nil)
    ((equal val '*) nil)
    ((equal val '-) 1)
    ((equal val '+/-) 1)
    ((t 2))))

(defun m-val-3 (val)
  (cond
    ((equal val'-) nil)
    ((equal val '*) 1)
    (t 2)))

(defun m-val-4 (val)
  (cond
    ((equal val '+) 1)
    ((equal val '-) 1)
    ((equal val '+/- 1)
    (t 2))))


;     S-SET-F (Same Set of Features Predicate).  This function

;returns t if sets one and two contain exactly the same features;

;otherwise, it returns nil.

(defun s-set-f (one two)
  (setq one (remove-duplicates one 's-fitr))
  (setq two (remove-duplicates two 's-fitr))
  (cond
    ((= (length one) (length two))
      (subsetp one two 's-fitr))
    (t nil)))


;     S-MATRIX (Same Matrix Predicate).  Exactly the same function

;as S-SET-F, included for mnemonic reasons.

(defun s-matrix (one two)
  (s-set-f one two))


;     S-SET (Same Set Predicate).  Do sets one and two contain

;exactly the same members?  S-SET returns t if yes, nil if no.
```

```
(defun s-set (one two)
  (setq one (remove-duplicates one 'equal))
  (setq two (remove-duplicates two 'equal))
  (cond
    ((= (length one) (length two))
      (subsetp one two 'equal))
    (t nil)))


;     AD-NEW (Add a New Item to a List).  This function adds an item

;to a list if the item is not already in the list.  If the item is

;already in the list, it returns the orginal list; otherwise, it

;returns the list with the new item added.

(defun ad-new (item list test)
  (cond
    ((member item list test) list)
    (t (cons item list))))


;     P-CHG (Apply Phonological/Morphological Change).  This

;function will vary depending on the string handling abilities of

;the LISP used and the specific phonological notation function.

(defun p-chg (change form)
  (if (null change)
    form
    (pack 'ch- form)))


;     ACCESS FUNCTIONS.  Input of various types is normally accessed

;via access functions (with AX- prefixes).  This allows the user to

;write inputs in a form different from that used within FLRN.

;     AX-MLEX [Access the Minimally Specified Lexicon].
;     AX-RR [Access the Lexical Redundancy Rules].
;     AX-SR [Access the Subcategorization Rules].
;     AX-MR [Access the Morphological Rules].
;     AX-IRR [Access the Inflectional Redundancy Rules].
;     AX-DR [Access the Derivational Rules].
```

```
;      STORAGE FUNCTIONS.  Output will normally be stored via storage
;functions (with BX- prefixes).

;      BX-DICT [Store the Dictionary].

;     SHOW [Show Steps in a Derivation].  This function is included
;to allow the user to trace the building of the lexicon.
(defun show () )
```

APPENDIX B

PROGRAMMER'S NOTES

FLRN includes a number of parts for which I cannot take complete credit. The chart parser within ASTR is from Gazdar and Mellish (1989), with only minor modifications to allow it to work with the lexicase DICT. In addition, one of the two systems I have been employing for the reading of FOCUS and GUIDE rules is Winston and Horn's (1984) system for forward chaining, with only a few minimal adjustments.

While writing FLRN, I lived in three different countries. During that time I had access to very different versions of LISP, The resultant code is rather strange. The version of FLRN that I currently use is written in MuLISP for IBM PC compatibles, but it includes a number of vestigial oddities from earlier versions. By the end of 1992, I expect to present a version of FLRN written in standard Common LISP to the Linguistics Department at the University of Hawaii, available for use by other scholars.

I welcome comments or inquiries. My address as of July 1992 is

Francis Lynn Lindsey, Jr.
Faculty of the Humanities
Ramkhamhaeng University
Bangkok 10240, THAILAND.

BIBLIOGRAPHY

Benedict, Paul K. 1972. Sino-Tibetan: a conspectus. Cambridge: Cambridge University Press.

Berwick, R.C. 1985. The acquisition of syntactic knowledge. Cambridge, Massachusetts: MIT Press.

Bloomfield, Leonard. 1933. Language. New York: Holt, Rinehart and Winston.

Bolinger, Dwight. 1958. Stress and information. American Speech 33:5-20.

Braine, Martin D.S. 1963. The ontogeny of English phrase structure: the first phase. Language 39:1-14.

Chomsky, Noam. 1965. Aspects of the theory of syntax. Cambridge, Massachusetts: MIT Press.

DeLancey, Scott. 1990. Sino-Tibetan languages. In Bernard Comrie (ed) The major languages of East and South-East Asia. London: Routledge.

Donegan, Patricia J. and David L. Stampe. 1983. Rhythm and the holistic organization of language structure. In J.F. Richardson, M. Marks, and A. Chukerman (eds) Papers from the parasession on the interplay of phonology, morphology and syntax. Chicago Linguistic Society.

Gazdar, Gerald, Alex Franz, Karen Osborne, and Roger Evans. 1987. Natural language processing in the 1980s: a bibliography. Stanford, California: Center for the Study of Language and Information.

Gazdar, Gerald and Chris Mellish. 1989. Natural language processing in LISP: an introduction to computational linguistics. Reading, Massachusetts: Addison-Wesley.

Ginsburg, Herbert and Sylvia Opper. 1969. Piaget's theory of intellectual development: an introduction. Englewood Cliffs, New Jersy: Prentice-Hall.

Greenberg, Joseph H. 1963. Some universals of grammar with particular reference to the order of meaningful elements. In J.H. Greenberg (ed) Universals of Language. Cambridge, Massachusetts: MIT Press.

Grice, H.P. 1975. Logic and conversation. In Peter Cole and Jerry L. Morgan (eds) Syntax and semantics, vol. 3: Speech acts. New York: Academic Press.

Grosz, B.J. 1986. The representation and use of focus in a system for understanding dialogs. In B.J. Grosz, K.Sparck-Jones and B.L Webber (eds) Readings in natural language processing. Los Altos, California: Morgan Kaufmann.

Grosz, B.J. and C.L. Sidner. 1986. Attentions, intentions, and the structure of discourse. Computational Linguistics 12:175-204.

Klein, Sheldon. 1966. Historical change in language using Monte Carlo techniques. Mechanical Translation 9:67-82.

Khamisi, Abdu M. 1985. Syntactic derivation of Swahili verbs. Ph.D. dissertation, University of Hawaii.

Langacker, Ronald W. 1987. Nouns and verbs. Language 63:53-94.

Lee, Nagiko I. 1989. Complementation in Japanese: a lexicase analysis. Ph.D. dissertation, University of Hawaii.

Lindsey, Francis L., Jr. 1985. Pronoun interpretation: an interpretive component for the lexicase theory. University of Hawaii Working Papers in Linguistics 17:71-91.

____. 1987. FLX: a lexicase parser. Technical report, Natural Language Processing, Hawaii International Center for High Technology Research, University of Hawaii.

O'Grady, William. 1986. Principles of grammar and learning. Chicago: University of Chicago Press.

____. 1991. Categories and case: the sentence structure of Korean. Philadelphia: John Benjamins.

Pagotto, Louise. 1985. Missing nominals in wh-questions and infinitives: a lexicalist view. University of Hawaii Working Papers in Linguistics 17:23-75.

____ and Stanley Starosta. 1986. Second annual report: lexicase grammatical theory and its application to English and Japanese grammar. Tokyo: NTT Research Laboratory.

Pinker, S. 1984. Language learnability and language development. Cambridge, Massachusetts: Harvard University Press.

Sapir, Edward. 1921. Language. New York: Harcourt, Brace and World.

Selfridge, Mallory G. 1986. A computer model of child language learning. Artificial Intelligence 29:171-216.

Slobin, Dan I. 1967. A field manual for cross-cultural study of the acquisition of communicator competence. Berkeley, California: University of California Press.

____. (ed) 1971. The ontogenesis of language. New York: Academic Press.

Springer, Hisami Konishi. 1992. Japanese topicalization: a lexicalist dependency analysis. Paper presented at the Pan-Asiatic Linguistics Conference, January 1992. To appear in Vol. 3 of the proceedings.

Starosta, Stanley. 1982. Mandarin case marking: a localistic lexicase analysis. University of Hawaii Working Papers in Linguistics 14:43-96.

____. 1988. The case for lexicase. London: Pinter.

____. 1989. Introduction to lexicase. Notes from a lecture given April 3, 1989, in Stuttgart, Germany.

____. 1990. Grammar from the lexicon. Notes from a lecture given June 11, 1990, at Chulalongkorn University, Bangkok, Thailand.

____ and Hirosato Nomura. 1986. Lexicase parsing: a lexicon driven approach to syntactic analysis. In COLING-86:127-132.

Vennemann, T. 1975. An explanation of drift. In C.N. Li (ed) Word order and word order change. Austin, Texas: University of Texas Press.

Winston, P.H. and B.K.P. Horn. 1984. LISP. Reading, Massachusetts: Addison-Wesley.

Woods, W.A. 1970. Transition network grammars for natural language analysis. In B.J. Grosz, K. Sparck-Jones and B.L Webber (eds). 1986. Readings in natural language processing. Los Altos, California: Morgan Kaufmann.