

UNCLASSIFIED

AD NUMBER

ADB029370

LIMITATION CHANGES

TO:

Approved for public release; distribution is unlimited. Document partially illegible.

FROM:

Distribution authorized to U.S. Gov't. agencies only; Contractor Performance Evaluation; JUN 1978. Other requests shall be referred to Air Force Rome Air Development Center, IRDT, Griffiss AFB, NY 13441. Document partially illegible.

AUTHORITY

RADC, USAF ltr, 26 Sep 1980

THIS PAGE IS UNCLASSIFIED

THIS REPORT HAS BEEN DELIMITED  
AND CLEARED FOR PUBLIC RELEASE  
UNDER DOD DIRECTIVE 5200.20 AND  
NO RESTRICTIONS ARE IMPOSED UPON  
ITS USE AND DISCLOSURE.

DISTRIBUTION STATEMENT A

APPROVED FOR PUBLIC RELEASE;  
DISTRIBUTION UNLIMITED.

L

2

LEVEL

Z

ADBO29370

RADC-TR-78-153  
Final Technical Report  
June 1978



QUINCE SYSTEM; STATE-OF-THE-ART REVIEW

William S-Y Wang  
Chiu-Chung Liao  
Robert Gaskins  
Mary S. Wang, et. al.

University of California

Distribution limited to U.S. Government agencies only;  
Contractor Performance Evaluation; June 1978. Other requests for  
this document must be referred to RADC (IRDT) Griffiss  
AFB NY 13441.

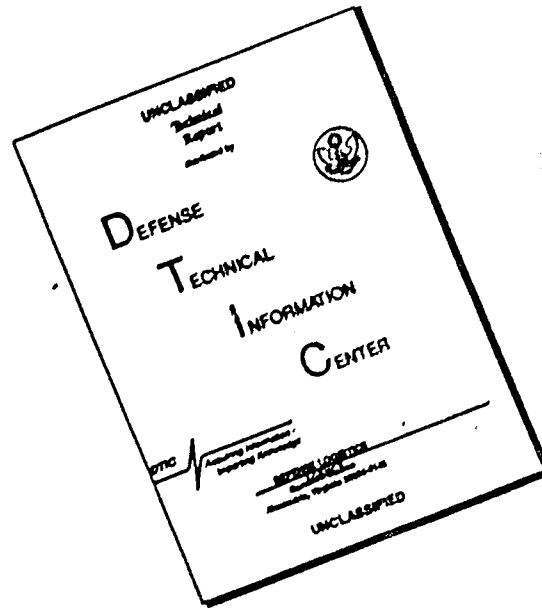
DDC FILE COPY

ROME AIR DEVELOPMENT CENTER  
Air Force Systems Command  
Griffiss Air Force Base, New York 13441

DDC  
RECEIVED  
AUG 23 1978  
B

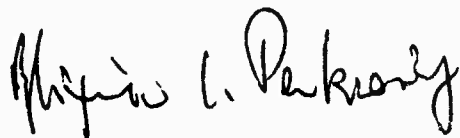
08 21 116

# DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

RADC-TR-78-153 has been reviewed and is approved for publication.



APPROVED:


ZBIGNIEW L. PANKOWICZ  
Project Engineer



APPROVED:

HOWARD DAVIS  
Technical Director  
Intelligence and Reconnaissance Division

FOR THE COMMANDER:



JOHN P. HUSS  
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (IRDT) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19 REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM	
1. REPORT NUMBER RADC-TR-78-153 ✓	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle) QUINCE SYSTEM; STATE-OF-THE-ART REVIEW. ✓		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report. 1 May 77 - 31 Jan 78	
7. AUTHOR(s) William S-Y/Wang, Robert/Gaskins Chiu-Chung/Liao, Mary S. Wang et al ✓		6. PERFORMING ORG. REPORT NUMBER N/A	
		8. CONTRACT OR GRANT NUMBER(s) F30602-77-C-0098/	
9. PERFORMING ORGANIZATION NAME AND ADDRESS The University of California at Berkeley Department of Linguistics Berkeley CA 94720		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62702F 45940830	
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (IRDT) Griffiss AFB NY 13441		12. REPORT DATE June 1978 ✓	
		13. NUMBER OF PAGES 217	
14. MONITORING AGENCY NAME & ADDRESS, if different from Controlling Office Same		15. SECURITY CLASS. (of this report) UNCLASSIFIED	
		15. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
16. DISTRIBUTION STATEMENT (of this Report) Distribution limited to U.S. Government agencies only; Contractor Performance Evaluation; June 1978. Other requests for this document must be referred to RADC (IRDT) Griffiss AFB NY 13441.			
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Same			
18. SUPPLEMENTARY NOTES RADC Project Engineer: Zbigniew L. Pankowicz, (IRDT)			
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Linguistic Theory Computational Linguistics Chinese-English Machine Translation Computer Programming			
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The Report documents a 9-month effort consisting of programming documentation for QUINCE System; inventory of R&D holdings (software, hardware, data files), and state-of-the-art assessment of linguistic and data processing requirements for IOC in Chinese-English machine translation. Background information is provided in Section I (INTRODUCTION). Section II contains a textual description of application software and utilities, including pertinent flowcharts, diagrams and tables. Programming documentation supplied under contract consists of a (Cont'd)			

D D C  
RECEIVED  
AUG 23 1978  
RECEIVED  
B

409 019

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

V

four-volume set (13 supplements). Inventory of hardware, software and data files is included in Appendix. Section III provides a critical review of the current state-of-the-art in computational syntactic description of natural languages from the viewpoint of its relevancy to machine translation R&D. Rules for a fragment of Chinese grammar are presented in the Appendix. Section IV describes requisites to a further consolidation of the linguistic data base in terms of semantic feature set, interlingual transfer component, contrastive lexical/syntactic studies and contextual analysis involving language-specific problems in Chinese (elided subjects; number, tense and aspect, definite vs. indefinite reference). Extensive bibliographies are provided in Section III (pp-173-187) and Section IV. (pp-205-207).

↑

UNCLASSIFIED	CONFIDENTIAL
SECRET	TOP SECRET
RESTRICTED	CONFIDENTIAL
CONFIDENTIAL	CONFIDENTIAL
CONFIDENTIAL/AMERICAN BOOKS	
SEE AVAIL and/or SPECIAL	
B	

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

## TABLE OF CONTENTS

	<u>Page</u>
<u>Summary</u>	1
I. <u>Introduction</u>	2
II. <u>Description and Documentation of the Quince Machine-Translation System</u>	5
1. Introduction	5
2. External Data Bases	6
2.1 Dictionary	8
2.2 Grammar	8
2.3 Telecode Substitution Table	8
2.4 Text	9
2.5 Formats	9
2.6 Additional Data Bases	10
3. Internal Data Storage	11
4. Interface Files	15
4.1 Canonized Text File	15
4.2 Segmented Text File	15
4.3 Vestigand File	15
4.4 Selected Dictionary File	16
4.5 Sentence Dictionary File	16
4.6 Format	16
4.7 Random Dictionary Module	17
5. Common Blocks	19
5.1 Interface File Definitions	19
5.2 Module Storage Tables	19
5.3 Table Names	20
5.4 Format, Block Data	22
5.5 Field Function Tables	22
6. Utilities	24
7. Additional Documentation	25



References	59
Appendix: Hardware, Software and Data Inventory	60
<b>III. <u>The State of the Art in Computational Syntactic Description of Natural Languages</u></b>	<b>65</b>
1. Introduction	65
2. Context-Free Grammars	67
2.1 Advantages of Context-Free Grammars	67
2.2 Disadvantages of Context-Free Grammars	68
2.3 Why Context-Free Grammars Grow Large	69
2.4 Attempts to Reduce the Size of Context-Free Grammars	73
3. A Model for Context-Free Grammars with Structured Vocabulary	75
3.1 van Wijngaarden Grammars	76
3.2 Notation for van Wijngaarden Grammars	79
3.3 Chomsky's $\bar{X}$ Convention as a van Wijngaarden Grammar	86
3.4 van Wijngaarden Grammars for Non-Context-Free Languages	93
3.5 van Wijngaarden Grammars with Hyper-Symbols as Predicates	98
4. Prior Linguistic Uses of Grammars with Structured Vocabulary	108
4.1 Pre-Chomskian Uses of Structured Vocabulary	110
4.2 Structured Vocabulary in Transformations, and "Extended Phrase Structure Grammars"	115
4.3 Post-Aspects Uses of Complex Symbol Vocabulary	122
4.4 Computational Linguists and Structured Vocabulary in Grammars	125
5. Restrictions on Grammars with Structured Vocabulary	133
5.1 A Restricted van Wijngaarden Grammar	134
5.2 Koster's Affix Grammars	141
5.3 Knuth's Attribute Grammars	147
5.4 An Experiment with Three Notations	151
5.5 The Parsing Problem for Restricted Grammars with Structured Vocabulary	164
6. The Quince System and Grammars with Structured Vocabulary	166
6.1 Previous Uses of Structured Vocabulary at POLA	167
6.2 Research Areas for Future Study	170

References	173
Appendix: Rules for a Fragment of Chinese Grammar	188

<b>IV. <u>Further Consolidation of the Linguistic Data Base: Lexical Features and Interlingual Transfer Rules</u></b>	193
1. Introduction	193
2. Lexical Features	194
2.1 Nature of Semantic Features	194
2.2 Functions of Semantic Features	194
2.3 Building up the Semantic Feature Set	195
2.4 The Other Types of Lexical Features	196
2.5 Types of Lexical Information	197
3. Interlingual Transfer Rules	197
3.1 Interlingual Component in Artificial Intelligence Approaches to Machine Translation	198
3.2 Nature, Functions, and Types of and Formalism for the Interlingual Transfer Rules	199
3.3 Contrastive Lexical and Syntactical Studies	200
3.3.1 Contrastive Lexical Studies	200
3.3.2 Contrastive Syntactic Studies	201
3.4 Contextual Analysis	202
3.4.1 Elided Subjects	202
3.4.2 Number, Tense, and Aspect	203
3.4.3 Definite vs. Indefinite Reference	204
References	205

**List of Figures:**

1. Quince modules and external data bases	7
2. Quince modules and interface files	13
3. Segment-by-segment processing and binary files	14
4. Random dictionary module	18
5. Overview of the Quince program writing system	21
6. Fields used in the field function tables, as organized by module storage table:	

A. IVCORE	26
B. SBCORE	27
C. LUCORE	28
D. LUCORE (cont.)	29
E. AGCORE	30
F. CSCORE	31
G. PRCORE	32
H. PROCRE (cont.)	33
I. ARCONI	34
J. ARCORE (cont.)	35
K. DTCORE	36
L. DTCORE (cont.)	37
M. SGDATA	38
N. SGTABS	39
O. STCORE	40
7. Data structures	
7A. IVCORE: segment-in-sentence data structure	41
7B. SBCORE: lattice structure during telecode substitution	42
7C. AGCORE: queue tree for LOOKUP winnowing	43
7D. ARCORE: linked lattice structure (successors only)	44
7E. ARCORE: linked lattice structure (predecessors only)	45
7F. ARCORE: lattice structure during parse-time winnowing	46
7G. PRCORE: unsorted and sorted linear order	47
7H. DTCORE: balanced tree structure	48
7I. SGTABS: linked segment table	49

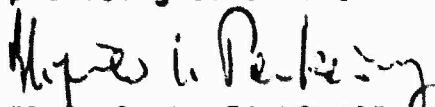
List of Tables:

1. Module Storage Tables Resident for Each of the Quince Modules	50
2. Field Function Tables Located in Each of the Module Storage Tables	51
3. Names of the Module Storage Tables, Field Function Tables, and Interface File Definitions	53
4. Block Data Subprograms for Each of the C-Block Blocks	55
5. Utility Routines Classified by Function	56

## EVALUATION

The Report constitutes a detailed state-of-the-art assessment of the QUINCE System for Chinese-English machine translation of S&T literature. The current version of the System consists of nine major modules controlling the translation process and a repertory of utilities controlling I/O operations, format conversion, debugging and optimization. The System's programming documentation is provided separately in Vols 1 - 9 (application software) and 10 - 13 (utilities). The System includes a variety of data bases, an extensive software package for implementation of the translation process, and a set of linguistic devices implicitly incorporated in linguistic and programming components to optimize the analysis of Chinese and English. A detailed description of the System is provided in Section II.

Judging from the viewpoint of scholastic merit, Sections III and IV constitute the most significant contributions to this Report. Section III contains an exceptionally comprehensive and thoroughly researched critique-in-depth of the current state of the art in computational syntactic description of natural languages, including a statement of its implications for a further development of the QUINCE System. It is concluded that grammars with structured vocabulary play an important role in all current language processing systems, including the QUINCE System in which this concept is elaborated in a more systematic manner than in other language processing systems. It also appears that a grammar notation based on Knuth's attribute grammars offers the most promising vista for a further development of the System. Section IV provides an exhaustive discussion of possibilities for a further consolidation of the linguistic data base in terms of the featurized lexicon and interlingual transfer rules. A continuing enhancement of the diversified feature subsystem and contrastive lexical/syntactic studies of Chinese and English, combined with contextual analysis of language-specific characteristics of Chinese are offered as the most promising solutions in this area.

  
ZBIGNIEW L. PANKOWICZ  
Technical Evaluator

## SUMMARY

This report presents the results of a nine-month period effort to document the Berkeley Chinese-English machine translation system (Quince system), to take inventory of all research materials, and to report on the current state of the art in linguistic theory, computational linguistics, and data processing techniques for advancement of the Quince system to the status of an initial operational capability in one sub-discipline of physics.

A detailed textual description of the Quince system modules plus a body of figures and tables are provided to assist the reader in conceptualizing the system and reading the program code listings appended in the Supplements to this report. An itemized inventory of both the hardware and software of the translation system is presented. We review the current state of the art in new syntactic descriptive methods with structured vocabulary, such as van Wijngaarden grammars, Koster's affix-grammars, and Knuth's attribute-grammars, which were developed for defining programming languages, but which are suitable for computational use in machine translation systems for natural languages. The existing linguistic data base of the system is reviewed in the light of current linguistic theory and of recent advances in artificial intelligence and computational linguistics. Suggestions for consolidating the linguistic data base and enhancing the parsing facility are made to advance the system to an initial operational capability.

## I. INTRODUCTION

The University of California Chinese-English machine translation research project was initiated in 1960 under a National Science Foundation grant. From 1967 to 1975 the Project was supported by the Department of the Air Force (Rome Air Development Center) under five contracts. These efforts have culminated in the development of the Syntactic Analysis System and the emerging of the Quince system for translation from Chinese to English.

However, owing to the abrupt termination of the previous contract in 1975 (F 30602-75-C-0059), the Quince system programs were left incomplete and their documentation has never had the chance of being adequately carried out. The present contract (F 30602-77-C-0098), covering a period of nine months, May 1977 to January 1978, provides the Project with an opportunity to fully document the Quince system programs as currently implemented and to reassess the whole translation system in the light of recent advances in linguistic theory, computational linguistics, artificial intelligence, and computer science. The documentation and inventory of the system and its reassessment will provide a smooth transition for any ensuing effort in Chinese-English machine translation research. The documentation and inventory of the system and its reassessment are the two major sections of this report. Chapter Two will be devoted to the first task, and Chapters Three and Four the second.

The Quince system, conceived as an integrated Chinese-English machine translation system, consists of two major components: the linguistic data base and system programs. Chapter Two presents a textual description of the Quince system software, keyed to the Supplements. It concentrates on those aspects least amenable to mechanical documentation: overviews of the system as a whole, the interface between major modules, and internal data structures. This chapter also includes a body of figures and tables to assist the reader in conceptualizing the translation

system and reading the Supplements.

There are thirteen Supplements altogether. Supplements One through Nine detail the contents of nine major modules of Quince system code. These are mechanically-produced documentations that explicitly extract the storage areas and the calling sequences of each sub-program. Each supplement contains three volumes: the Source Code Listing, and the Coding Internals Manual volumes I and II.

The remaining four supplements are mechanically-produced documentations of the utilities programs, and three types of storage documentation (loader storage, Common Block, and field function).

An itemized inventory of the hardware belonging to the U.S. government and the software related to the translation system is presented in the Appendix to Chapter Two.

Chapter Three reports on the state of the art in computational syntactic description of natural languages. It reviews context-free grammars and points out their inadequacies for handling natural languages and their clumsiness for human consumption. Van Wijngaarden's model for context-free grammars with structured vocabulary is presented to remedy the inadequacies of and avoid the clumsiness inherent in context-free grammars. The history of the use of structured vocabulary in linguistic descriptions is traced to some pre-Chomskyan structural linguists.

Restrictions on the generative capacity of van Wijngaarden grammars are discussed to arrive at a class of grammars easier to write and easier to parse. Related formalisms, i.e. Koster's affix-grammars and Knuth's attribute-grammars, are also explored and compared with other types of van Wijngaarden grammars. Finally, it is pointed out that the use of structured vocabulary in describing the Chinese grammar has been a topic of research at the Project since at least 1970. Future tasks and research areas for the Project are defined and strategies suggested.

In the last chapter, two components of the linguistic data base, the dictionary and interlingual transfer rules, are examined. To ensure better interactions between the two major sub-components of the grammar, the syntactic rules and lexicon, more lexical features are needed in the future grammar. Various kinds of the lexical features, their nature

and functions, and procedures to extract the lexical information from the existing grammar codes are discussed.

The status of the interlingual transfer rules in the translation cycle of the system is examined. Different types of the interlingual transfer rules and the formalisms to be used in the future are also discussed.

In addition, more contrastive lexical and syntactic studies between Chinese and English and contextual analysis are recommended in the future to strengthen the two components above of the linguistic data base. Strategies to achieve this goal are also briefly described. Areas where those studies will lead to the improvement of the linguistic data base are exemplified.



## II. DESCRIPTION AND DOCUMENTATION OF THE QUINCE MACHINE-TRANSLATION SYSTEM

### 1. Introduction

The Quince system is an integrated system for the machine translation of scientific texts from Chinese to English, developed at the Project on Linguistic Analysis, UC Berkeley. It includes several components: a large corpus of linguistic materials, such as texts, dictionary, and grammar; an extensive body of software to implement the translation; and a set of linguistic insights about how Chinese and English should best be analysed, which are implicitly incorporated within both the linguistic and programming materials.

The second component of the Quince system, the body of computer code written to perform the translation, is documented in this chapter and in the 13 Supplements appended to this report. It cannot, of course, be totally separated from its data base or theoretical approach. This component, however, has been under-documented in previous reports, and so it is presented here in isolation.

Supplements 1-9 detail the contents of 9 major modules of Quince system code. Each Supplement contains 3 volumes: the Source Code Listing, and the Coding Internals Manual in 2 volumes. These are mechanically-produced documentations that explicitly extract the storage areas and the calling sequences of each subprogram. These nine modules include the six 'main' modules, two 'supplemental' modules, and one module (the Parse Table Print Module) used for program debugging and linguistic research. Each module is written in Fortran.

The remaining four Supplements are mechanically-produced documentations of the utilities programs (Fortran as well as assembler), and three types of storage documentation (loader storage, Common Block, and field function).

This chapter presents a textual description of the Quince system software, keyed to the Supplements. It concentrates on those aspects least

amenable to mechanical documentation: overviews of the system as a whole, the interface between major modules, and internal data structures. This chapter also includes a body of appended tables to assist in reading the Supplements.

To large extent, the Quince system can be 'understood' by the flow and structures of the data at different stages of processing. The four external data files are the ultimate bases of the translation; of these, it can be said that the text is passed from module to module, sometimes in its entirety, sometimes segment-by-segment. This necessitates temporary interface files. Likewise, the grammar and its external/internal code conversion table reside in binary files. Within each module, segments of text are manipulated using data structures which are field function tables; these in turn reside in Common Blocks, that permit communication within and between modules. This chapter describes these data and storage details, as a documentation-by-effect of the Quince system.

The Quince system documented here is Version .8; this is to indicate that it is substantially complete, ready for documentation, but not finished. The translation from Chinese to English would be greatly improved if some of the data bases were enhanced, in particular by the addition of feature information to the dictionary and grammar; this would also be utilized by the transfer rules (see Chapter 4). The Quince system has prepared for these proposed changes in the data base, but until they are made available the system cannot be considered complete. Even so, the Quince system in its current form is a major research result in the field of machine translation.

## 2. External Data Bases

The Quince system has available four external data bases: a Chinese-English dictionary, a set of grammar rules for parsing Chinese, a set of Chinese telecode substitutions, and a raw text to be translated. (See Figure 1.)

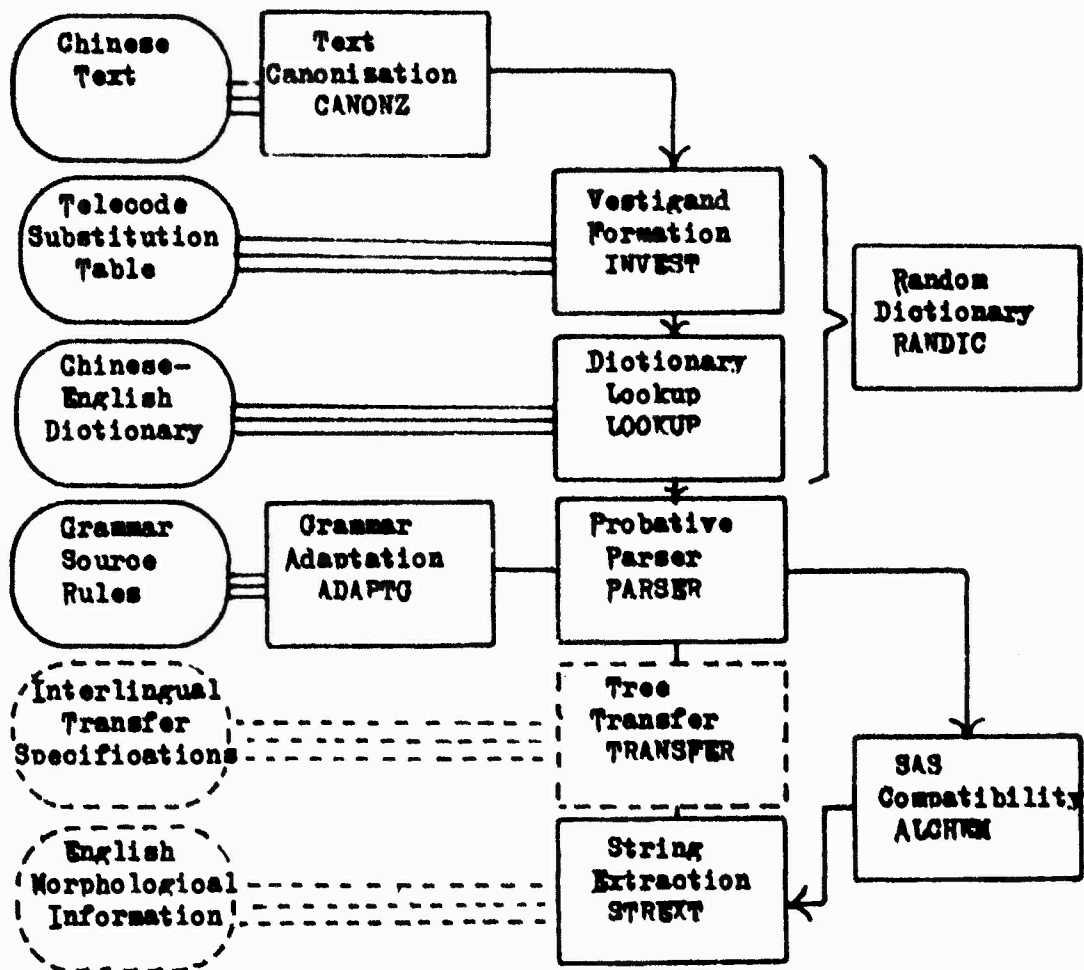


Figure 1. Quince Modules and External Data Bases. Overview of 4 external data bases, 2 additional data bases, 6 primary modules, 1 additional module, and 2 supplementary modules in the Quince system.

## 2.1 Dictionary

The Chinese-English dictionary exists in two bodies: the master CHIDIC (80,000 entries), and the smaller PHYDIC (40,000 entries). PHYDIC contains most general-purpose Chinese words, as well as technical terms in the fields of physics and mathematics. It is a subset of CHIDIC, which in practice is too cumbersome to maintain during research. Both dictionaries are in the same format. Each dictionary entry contains information on the grammatical category (terminal symbol) of the item in Chinese and its translation in English, keyed to telegraphic codes.

## 2.2 Grammar

The grammar is a set of context-free production rules, or source rules, which define the surface structure of Chinese; it is necessary to fully parse each segment of Chinese text before translation, as the structures of Chinese and English are so different. The grammar actually consists of 5 subgrammars, each of which handles a particular level in a parse-tree. These subgrammars are usually applied in sequential order. The size of each subgrammar is as follows:

Grammar 1	- -	125 rules
Grammar 2	- -	500 rules
Grammar 3	- -	2400 rules
Grammar 4	- -	340 rules
Grammar 5	- -	2750 rules

Many source rules are included in more than one subgrammar; in particular, Grammar 3 and Grammar 5 largely overlap.

## 2.3 Telecode Substitution Table

For every Chinese character there is a corresponding 4-digit telecode: this is the coding scheme used in the dictionary and in the text. There is, however, a set of characters that optionally substitute for one or more other characters. Accordingly, whenever potential substitution characters (i.e. telecodes) are encountered, their possible corresponding character(s) must also be made subject to dictionary lookup. These correspondences are found in the external telecode substitution table.

## 2.4 Text

The Chinese text to be translated includes 565,000 characters from physics and mathematics texts; it is broken down into subtexts for convenience. The text is not pre-edited: abbreviations are not expanded, telecode substitutions are not corrected for, different number-systems (e.g. classical, modern and Western) are left to co-exist, etc. In particular, the inconsistently-applied Chinese punctuation marks remain as in the original. The only additions are position-in-volume information for maintenance and identification purposes.

## 2.5 Formats

The four external data bases are maintained in an "external" format suitable for human maintenance: character strings, mnemonic category symbols, pronunciation information together with the telecodes, etc. Each file has its own set of software maintenance routines, peripheral to the Quince system and not here documented.

Because there have been no random-access facilities on the CDC 6400 at the University of California at Berkeley, these external files are fixed-field sequential files, stored on magnetic tape. During a previous contract period it seemed that such hardware facilities might become available; accordingly a random-access dictionary was designed, and a random dictionary software module was written to perform both telecode substitution and dictionary lookup. This module is one of two "supplemental" modules fully documented in this report. Were the random-access storage to become available, this random dictionary module would replace the present vestigand formation and dictionary lookup modules, and the present sequential dictionary format would become obsolete (see Figure 1).

As presently implemented, two of these external files are converted to an "internal" format before participating in the translation process: the table of telcode substitutions becomes an internal hash table in the module INVEST, and the grammar source rules are adapted into an automatically-allocated table by the module ADAPTG. The dictionary remains in external format; it is read through in one pass during the lookup process. The text first undergoes pre-editing, or canonization, and then is successively

broken down into smaller translation units; each unit is read in in external format, processed in internal format, and then saved on an intermediary interface file in external format.

## 2.6 Additional Data Bases

As originally designed, the Quince system included two more external data bases: a set of interlingual transfer specifications, and information on English morphology. These would, respectively, govern the transformation or transfer of a Chinese parse tree to an English tree, and tidy up the surface of the final string, by e.g. agreement in number and tense.

At present, these two data bases do not exist. The transfers have been approximated by reducing them to two tree operations: deletion and reversal of nodes. These are triggered by the application of specific grammar rules, which leave "transfer" codes on the node labels of the Chinese tree during parsing. In anticipation of a separate body of transfer specifications, the Quince system does not include a transfer module. There is instead a temporary interface module with the old SAS Syntactic Analysis System, which performs these limited transfers and also provides plotting capabilities for the resulting trees on the Calcomp plotter. The trees are then returned to the Quince system for string extraction. This SAS compatibility module is the second "supplemental" module documented in this report.

As presently implemented, each English word assumes its root dictionary form during string extraction, without morphological adjustment. The resulting translation is rough but reasonably comprehensible.

Chapter 4 outlines future plans for these two additional data bases. Once the interlingual transfer rules are available, an additional Quince module transfer will modify the Chinese parse tree prior to string extraction. The string extraction module will be expanded to include information from the morphological rules.

Figure 1 illustrates the relationship between the four external data bases and the six main Quince modules; it also includes the two supplemental modules, as well as two future data bases.

### 3. Internal Data Storage

During translation, the six Quince modules process the text in two modes: text-by-text (batch) and segment-by-segment. In batch mode, the entire body of text must pass through one module before entering the next; this is e.g. the case while preparing for dictionary lookup, since the entire text is looked up in the dictionary in one pass. In segment-by-segment processing, a single segment (translation unit) is passed through several modules. Obviously there is segment-by-segment processing within each of the six modules -- the distinction only becomes useful in discussing the interaction between modules.

The Quince modules pass information between each other in three possible forms: interface files, binary files, and module storage tables. These differ both in size and function, and are determined largely by the mode of processing.

During batch mode, the text is passed from module to module in the form of external interface files. These are of variable length (depending on the size of the text), and are written on tape as sequential character files. There are 5 interface files; each is written as output by one module, and rewound for use as input to the next module.

Binary files are used for two large internal bodies of data which cannot be accommodated in-core. One is the table of category (terminal) symbols, which relates the external strings coding these symbols to their internal hash-table codes; this table is used during both dictionary lookup and grammar adaptation, as both the dictionary and grammar source rules are external data bases. The second table is that of the adapted grammar itself -- this includes all five subgrammars, of which only one is in use during any one parse.

Once built within the translation run, these two tables are variously stored as binary files on tape, or as common files on system storage -- these storage allocations are performed automatically, depending on availability of space, to eliminate the repeated reconstruction of these tables and to minimize retrieval time.

The module storage tables are straightforward Common Blocks. They primarily provide for shared storage among the sub-programs within each of

the 6 modules; but they occasionally are shared by several modules, especially during segment-by-segment processing.

Figure 2 outlines the 5 interface files during batch processing, and their relation to the Quince modules. Figure 3 presents the flow of control in those modules that perform segment-by-segment processing; it includes the two binary files. Table 1 indicates which module storage tables (Common Blocks) are used by each of the Quince modules.



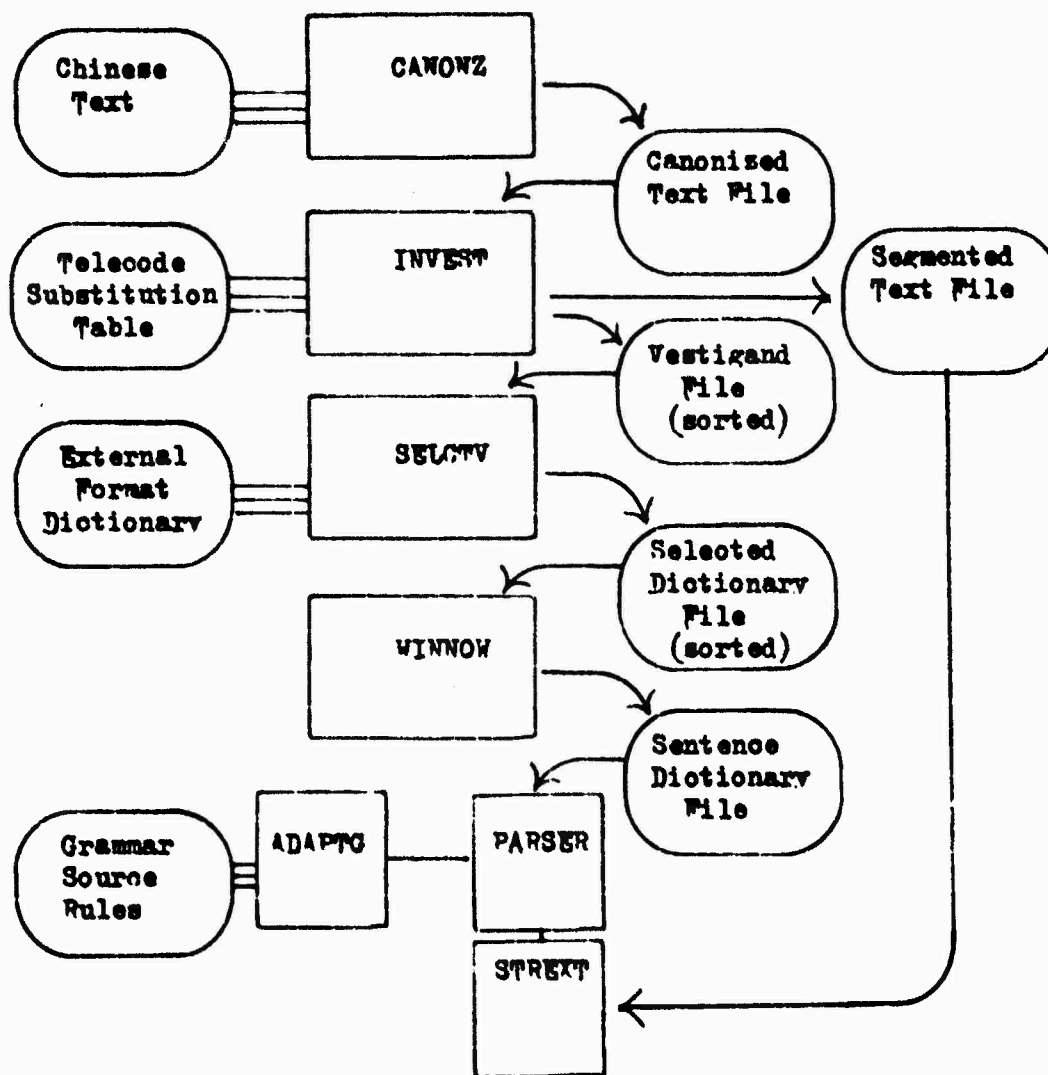


Figure 2. Quince Modules and Interface Files. Overview of 5 interface files, 4 external data bases, and 6 primary modules in the Quince system. (Note that the LOOKUP primary module is divided into its submodules SELECTV and WINNOW for clarity.)

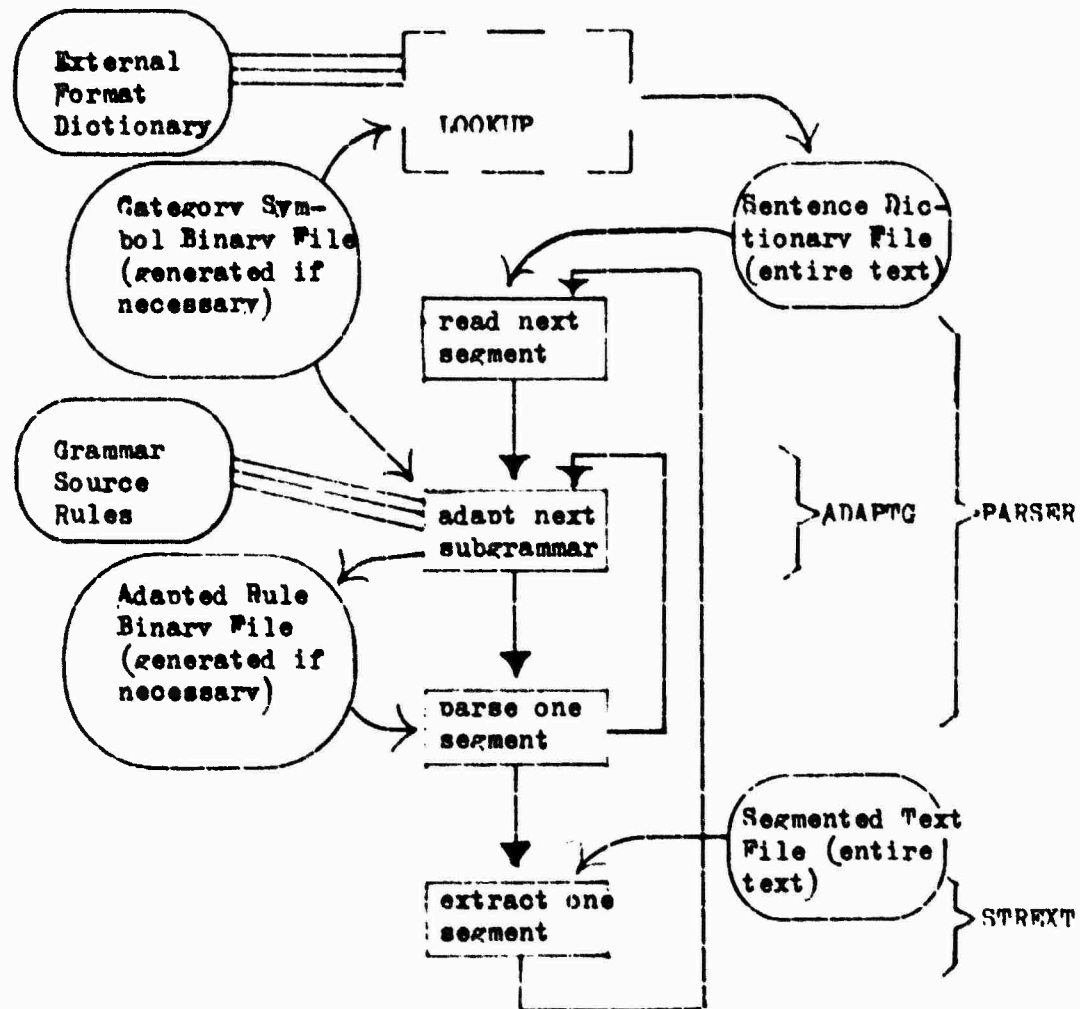


Figure 3. Segment-by-segment Processing and Binary Files. Segment-by-segment processing in the 3 end modules of the Quince system (ADAPTC, PARSER and STREXT), and their use of 2 binary files.

#### 4. Interface Files

##### 4.1 Canonized Text File

Before the raw text can be translated, each of its sub-texts must be pre-edited into smaller units, called sentences; these correspond roughly to English "sentences". Because of differences between English and Chinese punctuation practices, each Chinese "sentence" typically contains several of these smaller sentence units. In addition, special telecodes and characters such as textual identifiers, parentheses, footnotes, etc. must be analysed and related to the sentence. All these processes use an internal table of special telecodes in the Quince module CANONZ; this module outputs a canonized text file, in which the string of telecodes is broken up into sentences within the text.

##### 4.2 Segmented Text File

After the preliminary editing during canonization, each sentence is divided still further into segments; in practice, these segments will correspond to parse-units during parsing. In the dictionary each lexical entry may consist of 1-7 telecodes (including e.g. idioms or compounds); one characteristic of a segment is that a lexical entry will not extend past the segment boundaries. These boundaries include punctuation marks (period, parentheses, commas), as well as special syntax-marking Chinese characters.

The module INVEST outputs a segmented text file, this is a reformulation of the canonized text file in terms of segments rather than sentences. This file does not participate in the dictionary lookup or parsing of that segment: it is kept around in text order until STREXT, for research purposes, so that the English and Chinese strings may be manually compared.

##### 4.3 Vestigand File

As each segment is determined, it is necessary to make a list of all its lexical items; these are subsequently subject to dictionary lookup. Determining word boundaries in Chinese is, however, not a trivial task, as each dictionary entry consists of a variable number of telecodes. Thus from each segment are calculated all the vestiganda: these are strings of from 1 to 7 telecodes in length, any of which might be a valid lexical item.

Thus the module INVEST also outputs an interface vestigand file. This is the segmented text file formulated in vestigands (rather than in telecodes) for each segment.

#### 4.4 Selected Dictionary File

The vestigand file is first sorted into dictionary order (using the file/sort utility); and then the sub-module SELECTV (first half of the LOOKUP process) searches for each vestigand in the external dictionary. For each vestigand found (now a lexical item), it records all the dictionary information except the romanization (pronunciation), in addition to the information from the inputted vestigand file, onto the selected dictionary file. The selected dictionary file thus includes fewer records than the vestigand file, since many vestigands were not found in the dictionary; but each existing record includes more information. This file is then sorted back into text order.

#### 4.5 Sentence Dictionary File

Since many of the postulated vestigands have been rejected during dictionary lookup in SELECTV, it now becomes necessary to reconstitute each segment in terms of valid lexical items. This occurs in the sub-module WINNOW (second half of LOOKUP). During winnowing, the shortest complete paths are found which connect the beginning and end of each segment; for each rejected vestigand, every path which originally included it must be discarded -- and this in turn may eliminate some occurrences of otherwise acceptable lexical items, since they no longer "occur" in the segment. The sub-module WINNOW thus outputs the sentence dictionary file: each record contains all the information of the selected dictionary file, in nearly identical format, but there are fewer records. Also, the grammatical code for each lexical item is additionally expressed in terms of its category symbol, its internal code which will key it to an internal hash table during parsing.

#### 4.6 Format

The exact record format for each of the interface files is contained and documented in the interface file definitions; see Section 5.1 and

## Supplement 12.

### 4.7 Random Dictionary Module

It should be noted that there would no longer be any interface files if (as originally designed) the random dictionary module, with its associated random-access dictionary, were to replace the modules INVEST and LOOKUP. This is because batch processing would no longer be necessary: after the raw text had been pre-edited by CANONZ, each vestigand could be extracted and made subject to immediate dictionary lookup, and each segment parsed as soon as it was formed. This is illustrated in Figure 4. It should be remembered, however, that the module random dictionary has not been able to be implemented, so that as presently documented its input and output files are not yet defined in the code.

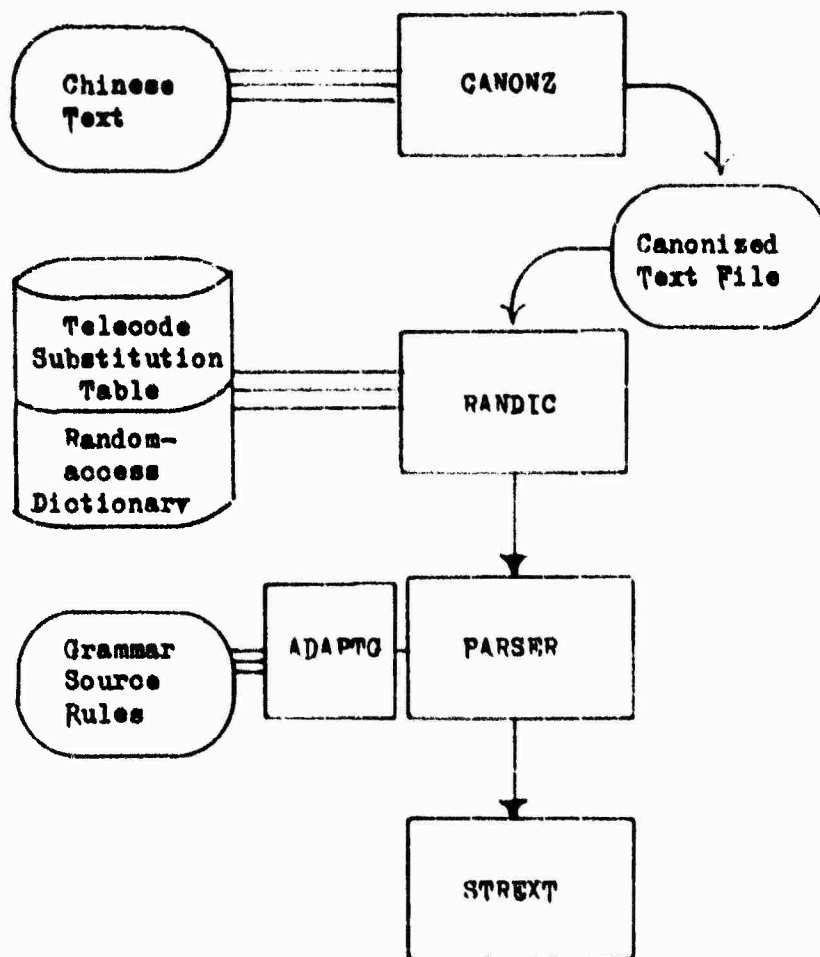


Figure 4. Random Dictionary Module. With RANDIC only the raw Chinese text would be processed in batch mode (CANONZ); all subsequent processing would be segment-by-segment.

## 5. Common Blocks

The six main Quince modules plus the two supplemental modules together use 23 common blocks -- shared storage areas. These blocks are functionally of two kinds: interface file definitions and module storage tables.

### 5.1 Interface File Definitions

These six areas are used for the five interface files plus the external-format dictionary. Each area includes buffer space for one file record, plus constants defining each record and variable names referencing each field in the record. The areas do not include any "working space" for processing the files: they simply "define" the file, and hence these interface file definitions are used by both the outputting and inputting module on either side of the interface file.

Because the interface files are sequential character files, there are no fields of less than one character in length, and no hash tables or other special allocation involved -- thus the interface definition tables contain no tables accessed through field functions.

### 5.2 Module Storage Tables

The remaining 17 common blocks are used primarily for shared constants and working space within the various subprograms that make up each of the 8 Quince modules, although a few blocks are shared by several main modules. This is illustrated in Table 1.

Two of these common blocks (LBCORE and MNCORE) are always resident. They are part of the debugging and optimizing capability of the Quince Program-Writing System, and are not considered part of the Quince translation system in the following discussion.

Most of the remaining 15 common blocks include tables with special storage requirements: fields of less than 1 character (bit-level), hash tables with numeric or character keys, floating tables in ECS, dynamically allocated tables, etc. These are the tables accessed through the so-called field functions. The 28 field function tables are located in the 15 module storage tables as shown in Table 2.

### 5.3 Table Names

In the Quince system source listings, and in the detailed program documentation which supplements this report, these various tables have different names in different contexts. This is a consequence partly of the Quince Program Writing System (outlined in Figure 5), and partly of attempts to write system-independent code, include e.g. separate names for Fortran arrays and Common Blocks. These names are related to each other in a reasonably systematic way, as detailed in Table 3. In the present chapter we will always use the "name-1" in Table 3 -- the input to the CBA Common Block Allocator and the input to the FFN Field Function Writer -- when referring to the interface file definitions, module-storage tables, and tables accessed through field functions.



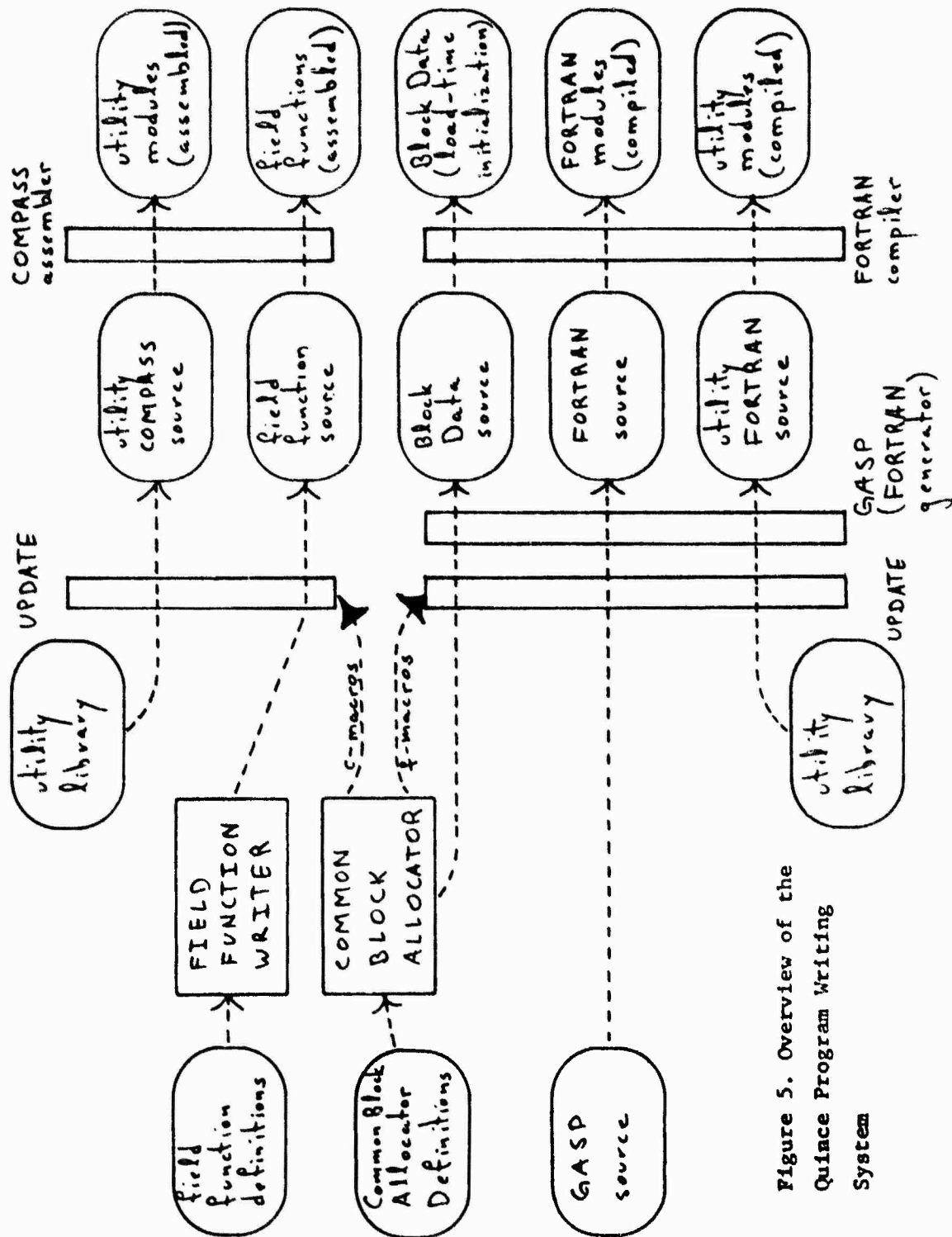


Figure 5. Overview of the Quince Program Writing System

#### 5.4 Format, Block Data

Both the interface file definitions and module storage tables are documented in the Common Block Allocator definitions in Supplement 12. The constants from each table are extracted to build Block Data subprograms, for load-time initializations; these are documented in Supplement 11. Table 4 lists the Block Data subprograms and their corresponding common blocks.

#### 5.5 Field Function Tables

The 28 field function tables are organized so that any field in any table, regardless of internal format, can be accessed by Fortran in a transparent and straightforward way. The variable names and constants associated with each table are documented in Supplement 13. The fields -- their names and size -- are illustrated in Figure 6A-60; these are grouped according to which storage module table they are located in.

These field function tables appear in the code as follows. Consider the fields IVPSG and IVNSG in field function table IVSFFN, as illustrated in Figure 6A. These are both pointers, one to the previous segment and one to the next segment. The following code would reverse two segments by interchanging the pointers:

```
DUMMY = IVPSG(I)
IVPSG(I) = IVNSG(I)
IVNSG(I) = DUMMY
```

This example is written in GASP; here is the FORTRAN code generated by the GASP Program Writer:

```
DUMMY = IVPSG(I)
CALL IVPSGØ (NULL, I, IVNSG(I))
CALL IVNSGØ (NULL, DUMMY)
```

The subroutine names IVPSGØ and IVNSGØ have been generated from the function names IVPSG and IVNSG to form a set/retrieve pair of field functions. In Supplements 1-9, all field functions are identified as "Calls Made to Routines Outside Module".

Within a single module storage area there may be several field function tables. These are often linked with each other by pointers and pointers-to-pointers, and processing consists primarily of moving these pointers around. It is generally the case, however, that each pointer always links with a certain type of node, i.e. with a certain other field function table, although the particular node in question may change within the table.

Figure 7 presents some of the more complex data structures used in the Quince system. The field names and field function table names are as in Figure 6A-60.

At almost every stage of the translation process, the text must be considered to have the data structure of a lattice, rather than a string. This is due to the linguistic nature of Chinese -- there are so many ambiguities in its analysis. Telecode substitution introduces alternate readings for each telecode, vestigands introduce alternate combinations of telecodes, multiple possible category symbols for each lexical item introduce alternate parse trees. For this reason, at almost every stage of translation, and within each module, data structures such as those in Figure 7 are used. Their manipulation takes up much of the program logic within the Quince modules; accordingly, the modules themselves will not be further documented in this chapter. The reader is referred to Supplements 1-9 for further details on the modules themselves.

## 6. Utilities

The Quince utility modules include those subprograms which are machine- or system-dependent, but which (generally) do not manipulate fields-within-a-word (this capability is provided by the field functions).

The utility source listings are presented in Supplement 10; they do not, however, contain as much internal documentation as do the other Quince modules, and so they are summarized in this section.

The utilities provide support in three general areas: input/output, format conversion, and debugging and optimization. The IO routines handle files on random-access storage, Extended Core Storage, and system files (coded and binary), as well as reading the system registers. They include the READS and WRITES routines, which replace the FORTRAN read and write statements for coded serial files.

The format conversion routines convert and shift among binary, decimal, integer, display character, and FORTRAN A1 format. They provide justification, and handle the only fixed-format field in the Quince system: packed machine-dependent telecodes.

The debugging and optimization routines are the most conspicuous in the Quince module source listings, as they appear in every routine to permit timing, tracebacks, and counts of entry-within-each-routine; they are specially implemented so as to catch fatal FORTRAN errors before they produce

a system crash, so that traceback can be completed. It is these routines that use the resident Common Blocks LOCORE and NMCORE. In a full production version of the Quince system, many of these modules would be removed completely; at present, they are controlled by switches on the system registers. Table 5 lists the utility routines by their function.

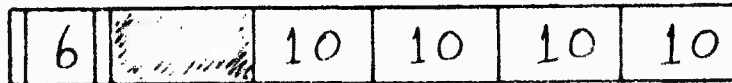
#### 7. Additional Documentation

The Quince system has also been documented in previous final reports. These tend toward providing a description of the processing in linguistic, rather than in computer, terms; however, much of the information is still current. In particular, (3) presents the coding conventions for identifying the special telecodes, and an outline of the procedures used in text preparation. (4) outlines the "steps" of machine translation.

The Quince Program Writer has a full description in (4), which also describes the plotting capabilities available through the SAS Compatibility module.

There are also several unpublished papers available from the Project. (1) is a manual for writing GASP, the structured programming language used as the source language for all the Quince modules; this source code is translated into Fortran by the GASP translator. (2) is a description of the theoretical approach used in the PARSER module.

IVCORE



IVSFFN

- IVFRS (10) - first telecode in segment
- IVIAS (10) - last telecode in segment
- IVPSG (10) - previous segment pointer
- IVNSG (10) - next segment pointer
- IVLRF (1) - last-seg-in-sentence flag
- IVPCI (6) - punctuation break class this seg  
(pos segs only)
- IVPOS (1) - punctuation only segment flag

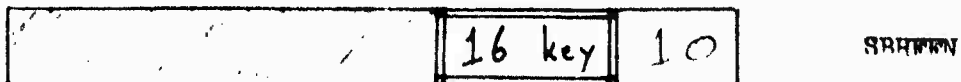


IVTFFN

- IVTCI (16) - packed telecode
- IVPSN (10) - previous telecode in sentence pointer
- IVNSN (10) - next telecode in sentence pointer

Figure 6A. Fields used in the Field Function Tables of Module Storage Table IVCORE

SBCORE



HSRHC (10) - substitution head for key telecode



SRNVI (12) - length of new telecode string

SRNWP (12) - head of new telecode string

SROLI (12) - length of old telecode string

SROLP (12) - head of old telecode string

SRNXS (12) - next substitution

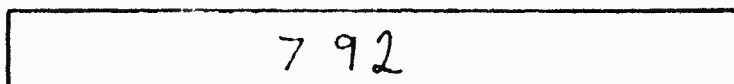


SRNXT (12) - next telecode node

SRHCI (16) - telecode

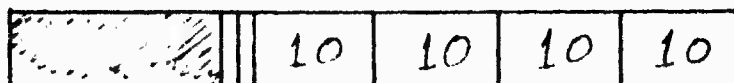
Figure 6B. Fields used in the Field Function Tables of Module Storage Table SBCORE

LUCORE



I,PTXT

I,PTXT (792) - 132-character text field




I,PTPRM

- I,PTPRM (10) - from nos for this type
- I,PTTO (10) - to nos for this type
- I,PTFRS (10) - first span for this type
- I,PTLAS (10) - last span for this type
- I,PTIN (1) - last-in-unit flag
- I,PTKP (1) - keep-this-type flag

Figure 6C. Fields used in the Field Function Tables of Module Store Table LUCORE



LUCORE (cont.)

12	13		13
----	----	---	----

I.Q.T.F.F.W

- I.P.A.T.R (13) - tree-organizing pointer
- I.W.Q.I.T.R (13) - queue-organizing pointer - next
- I.O.H.T.R (12) - I.O.H head represented

	10	10	10	10	10
---	----	----	----	----	----

I.S.P.F.F.W

- I.S.N.T.P (10) - head of list on which located
- I.S.T.N (10) - index (should be same) of WIS character text
- I.S.O.P.P (10) - next span on this list
- I.S.F.P.K (10) - from position
- I.S.T.P.K (10) - to position

Figure 6D. Fields used in the Field Function Tables of Module Storage Table LUCORE (cont.)

AGCORE

8	13	12	24
		12	12

ARI.FFN

- GRSID (24) - rightside of rule
- GLRCS (12) - LEFTCS — first part of rule rightside
- GRPCS (12) - RIGHTCS — second part of rule rightside
- GLPCS (12) - TOPCS — left side of rule
- QSRNW (13) - source rule ID number
- GIITR (8) - interlingual transfer code for this rule
- GNPFA (1) - more-flag — this rule has a non-unique right side (GRSID)
- GNPFL (1) - pseudo-flag — this rule creates a pseudo-left side

12	12		13
----	----	--	----

QTR.FFN

- WPATR (13) - tree-organizing pointer — father
- WNQUE (12) - queue-organizing pointer — next
- WCNST (12) - constitute type represented

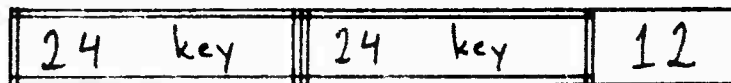
Figure 6E. Fields used in the Field Function Tables of Module Storage Table AGCORE

CSCORE



CSLFFW

CSMR (48) - category symbol print representation

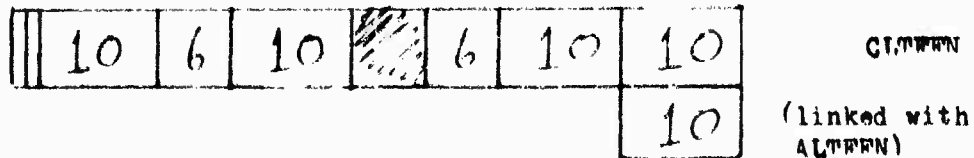


CATPTW

CATPT (12) - pointer into category symbol data table

Figure 6F. Fields used in the Field Function Tables of Module Storage Table CSCORE

PRCORR



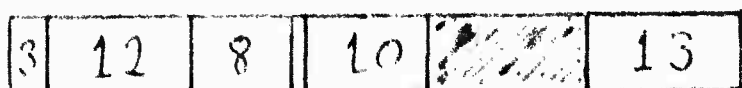
- DCNT<sup>m</sup> (10) - scratch copy of SCNT<sup>m</sup> (overwritten when LNNX is made)
- LNNX (10) - linear order next pointer
- UNLNRD (10) - un-linear order next queue-pointer
- SCNT<sup>m</sup> (6) - count of direct successors
- LNER (10) - linear order ordinal position (used as sort key)
- PCNT<sup>m</sup> (6) - count of direct predecessors
- LNPR (10) - linear order previous pointer
- SEFN (1) - seen flag (for transversal set-up)
- DUPL (1) - duplicate sonpos-type flag



- CSUSD (15) - index of active constitute with category symbol

Figure 6G. Fields used in the Field Function Tables of Module Storage Table PRCORR

PRCORE (cont.)



CRCPWR  
(linked with  
ARCPWR)

- KQI(13) - queue-link in sentence position queue
- KOI(10) - window — old lattice item represented
- KFF (1) - window — first of type flag
- KPF (8) - copy of POSWR field
- KCS (12) - copy of CAPSW field
- KTY (1) - copy of TYPWR field

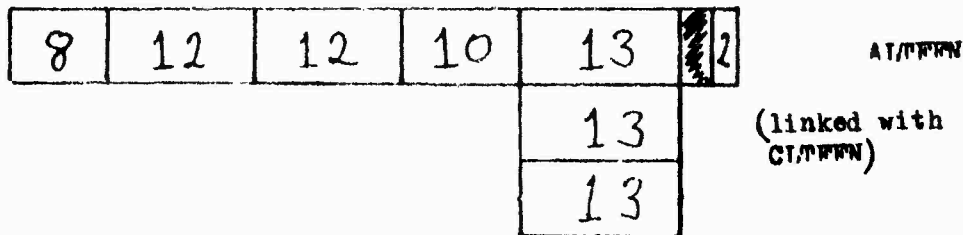


SQWR

- WRSPA (13) - active queue head
- LASPA (13) - active queue tail
- WRSPQ (13) - quiescent queue head
- LASTQ (13) - quiescent queue tail
- WRLAG (1) - last sensor in unit flag
- SQWLA (1) - flag — this sensor prepared for winnowing

Figure 6H. Fields used in the Field Function Tables of Module Storage Table PRCORE (cont.)

ARCORE



- TYPEL (2) - type field
- IATPT (13) - cover field for either of OI/CON/  
SYNDIC variants
- OI/CON (13) - constitute references (non-terminal)
- SYNDIC (13) - sentence dictionary pointer (terminals)
- INTRA (10) - pointer for linear traversal of  
lattice
- SCLST (12) - successor list
- PRIST (12) - predecessor list
- SPTYP (8) - sentence position type



IATPK (12) - lattice next pointer



IATLK (12) - lattice link pointer

Figure 6L. Fields used in the Field Function Tables of  
Module Storage Table ARCORE

ARCORE (cont.)

3	12	8	8	6	13	8
---	----	---	---	---	----	---

ARCORRN

(linked with  
CRCORRN)

- GRAMR (8) - grammar used to make
- PRDKS (13) - source rule number to make
- DSPNS (6) - parser invocation serial number
- POSFR (8) - sentence position from
- POSTO (8) - sentence position to
- QUIES (1) - quiescent flag
- ALFNT (1) - alternate flag
- CATSM (12) - category symbol pointer
- TYPEC (3) - type field

13	13	13	8	
----	----	----	---	--

- INLNG (8) - interlingual transformation code
- DICAD (13) - sentence dictionary pointer  
(terminals)
- RINDX (13) - right constituent constitute
- LINDX (13) - left constituent constitute

Figure 6J. Fields used in the Field Function Tables of  
Module Storage Table ARCORE (cont.)

DTCORE



DCWFFW

- RWINK (8) - link to right son in balanced tree
- LWINK (8) - link to left son in balanced tree
- LEXPT (8) - pointer to lexical heuristic information
- PSEQN (30) - permanent sequence number



- TC4DC (16) - 4th telecode
- TC3DC (16) - 3d telecode
- TC2DC (16) - 2nd telecode
- LPSNN (10) - last sense number assigned for this word
- BAL (2) - balance factor for balanced tree inspection

Figure 6K. Fields used in the Field Function Tables of Module Storage Table DTCORE



DTCORE (cont.)

	3	7	16	16	16	DCW <sup>REP</sup> (cont.)
--	---	---	----	----	----	----------------------------

- TC7DC (16) - 7th telecode
- TC6DC (16) - 6th telecode
- TC5DC (16) - 5th telecode
- SNSPT (7) - pointer to first word sense
- NTCS (3) - number of non-blank telecodes

8		16
---	--	----

- TC1DC (16) - 1st telecode
- UWLNK (8) - pointer up to father of node

Figure 6L. Fields used in the Field Function Tables of Module Storage Table DTCORE (cont.)

SGDATA



SGATPRN

- NXTAC (10) - pointer to next in list
- TXTRN (10) - pointer to telecode node



SGSPRN

- LASTSPN (5) - last span list element
- NXSPN (5) - next span list element
- BRKTC (16) - actual break telecode
- SPCLS (4) - class of this break



SGTRPRN

- LASTNK (10) - pointer to list of last active
- NEXTNK (10) - pointer to list of next active
- NXTXT (10) - next in input text order
- TCFLD (16) - telecode

Figure 6M. Fields used in the Field Function Tables of Module Storage Table SGDATA

SCOTABS



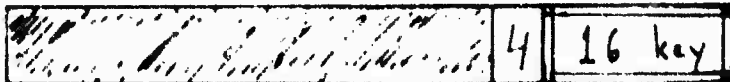
NEWTC

NEXTSA (12) - pointer to next subst/alt node  
 NEWTC (16) - subst/alt telecode



SBAIT

SBAIT (12) - pointer to first subst/alt

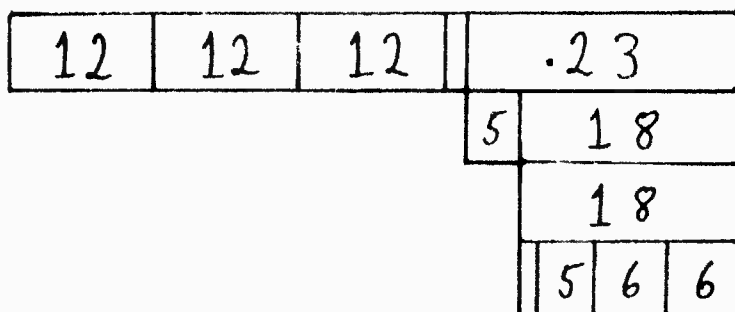


RKCLR

RKCLR (4) - class of segmentation break of key  
 telecode

Figure 6N. Fields used in the Field Function Tables of Module Storage Table SCOTABS

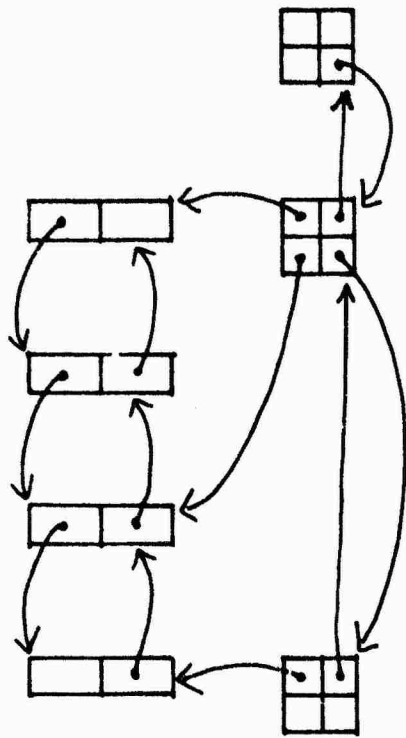
STCORE



STRWNN

- INFO (23) - info — type field and data field seen as one
- SONUM (18) - sonum - SDATA field seen as constitute number
- SDATA (18) - sdata field (multi-use — with sub-fields in STRWNN for full strings)
- SCURR (6) - scurr counter (full strings)
- SLAST (6) - slast counter (full strings)
- SFRST (5) - sfrst counter (full strings)
- SANYS (1) - any-summs-under-flag (full strings)
- TYPE (5) - type field
- TFLAG (1) - flag to condition presence of RIARI-RLINK/PLABI-THRED
- THRED (12) - thread link
- RLINK (12) - right link
- LLINK (12) - left link

Figure 60. Fields used in the Field Function Tables of Modules Storage Table STCORE



IVNSN	IVPSN
-------	-------

IVTFFN  
(ivcore)

IVFRS	IVPSG
IVLAS	IVNSG

IVSFFN  
(ivcore)

Figure 7A. Data Structures in IVCORE:  
segment-in-sentence data structure

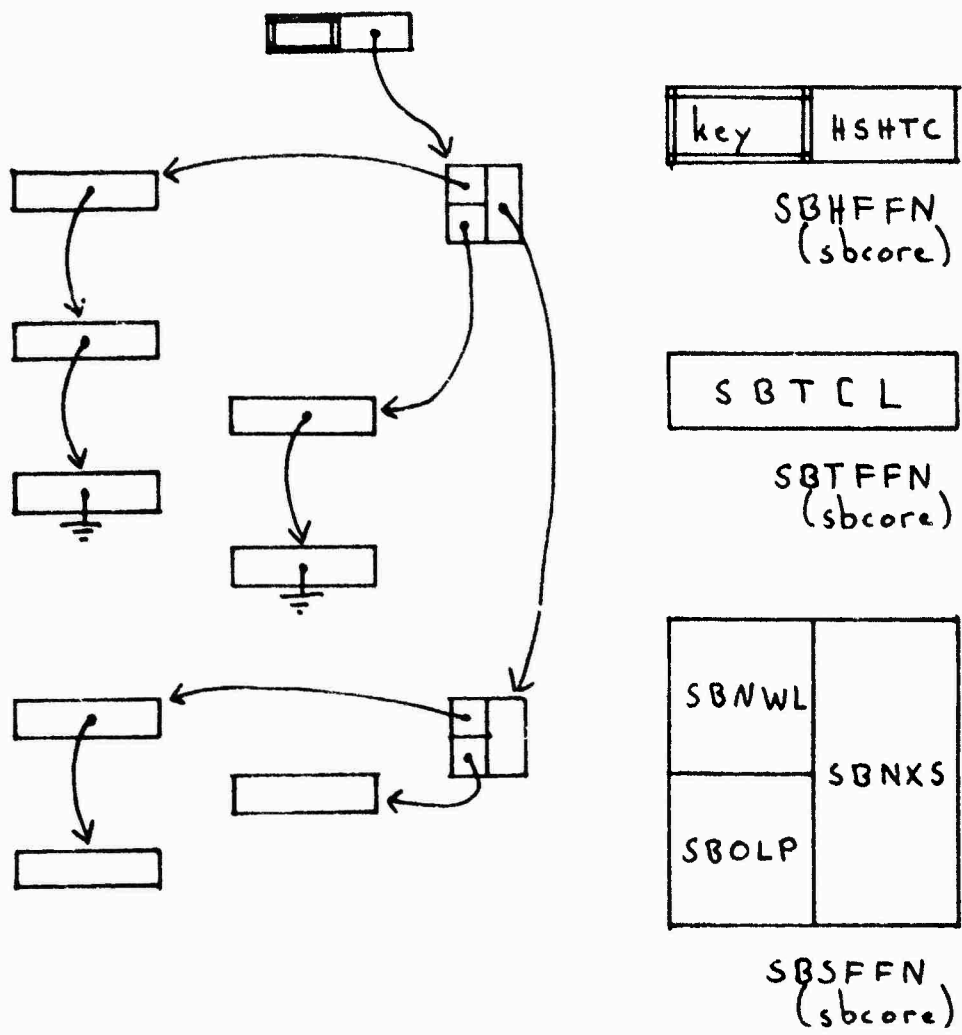


Figure 7B. Data Structures in SBCORE:  
lattice structure during telecode substitution

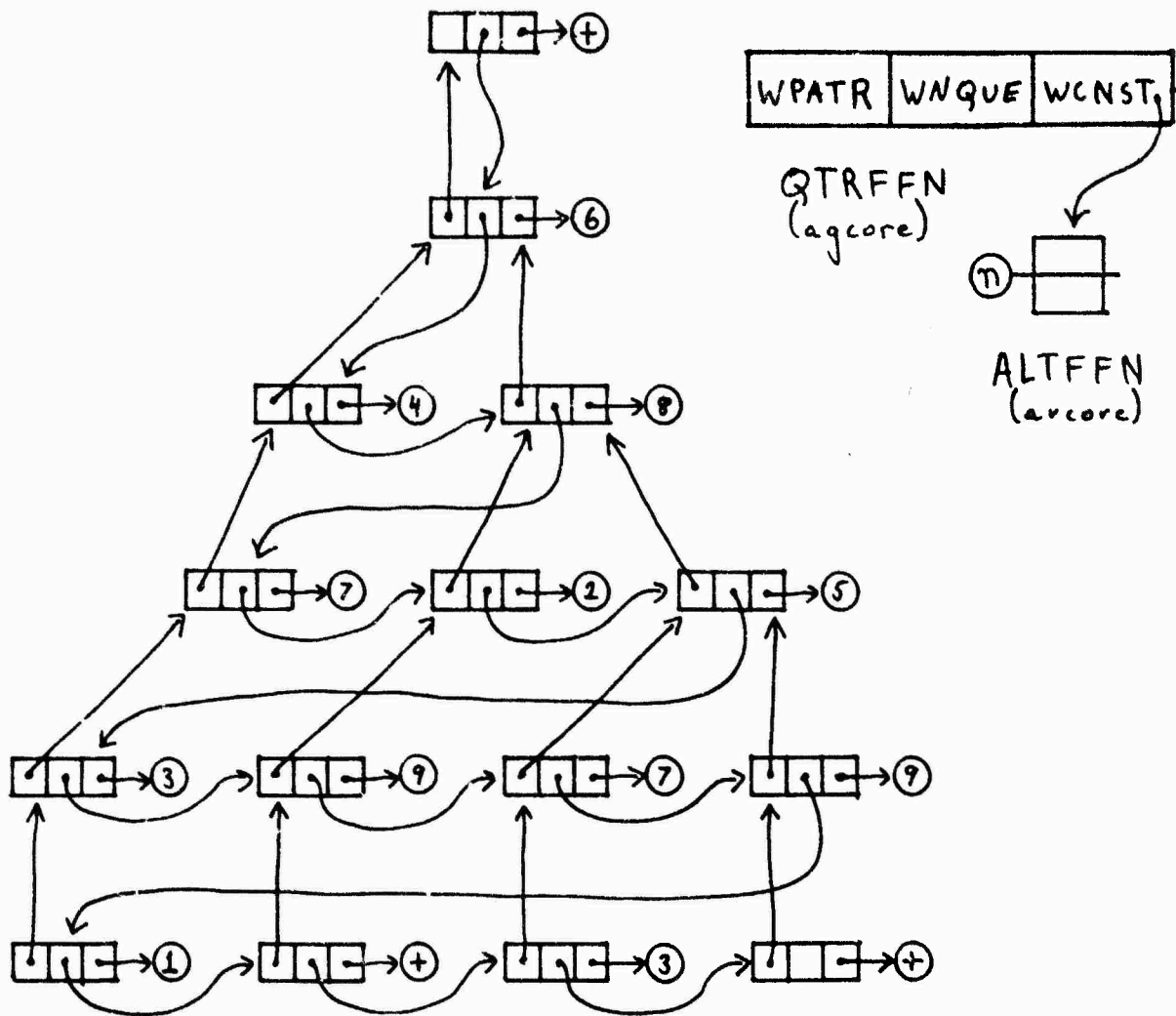


Figure 7C. Data Structures in AGCORE:  
queue tree during LOOKUP winnowing

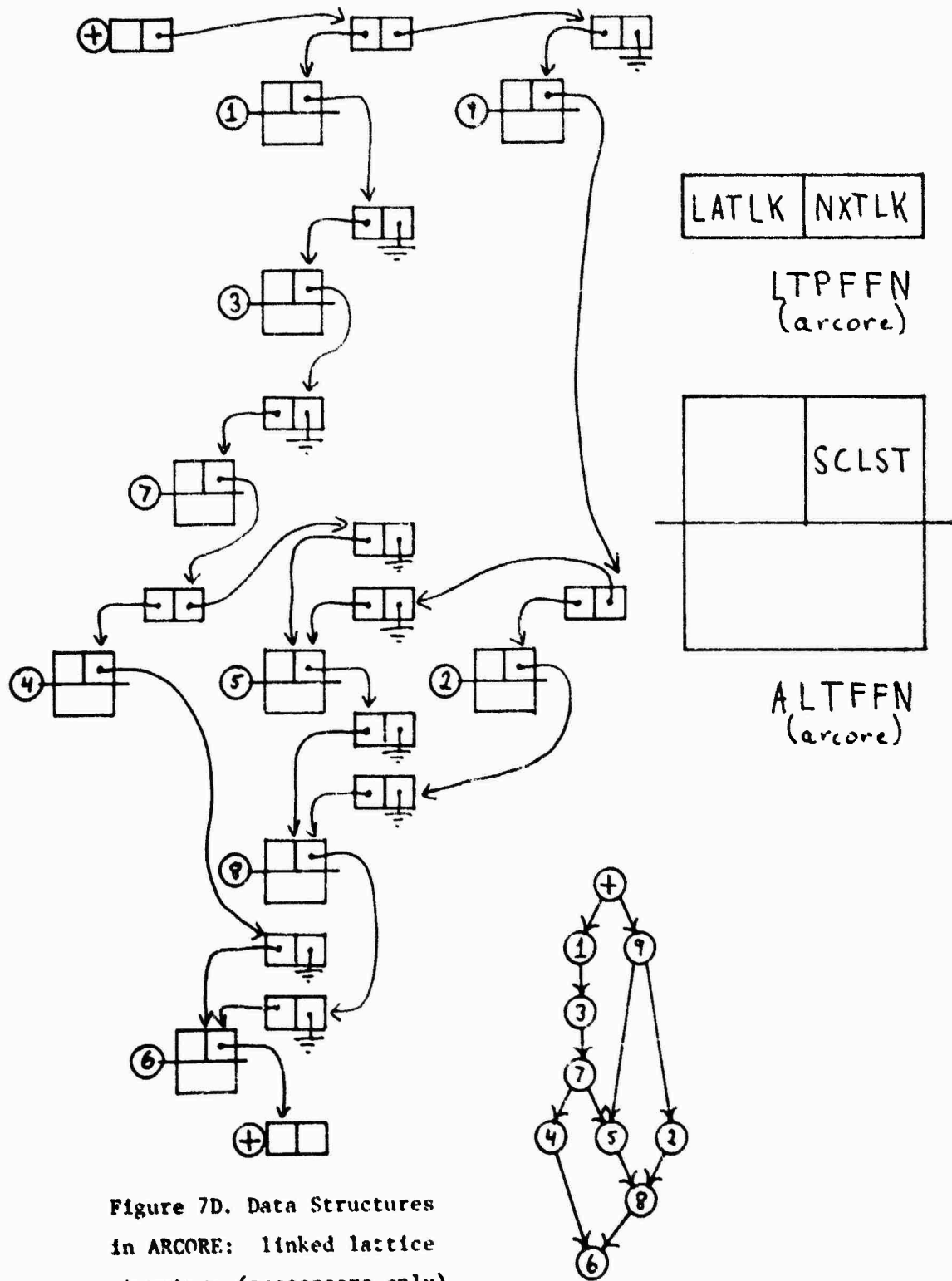
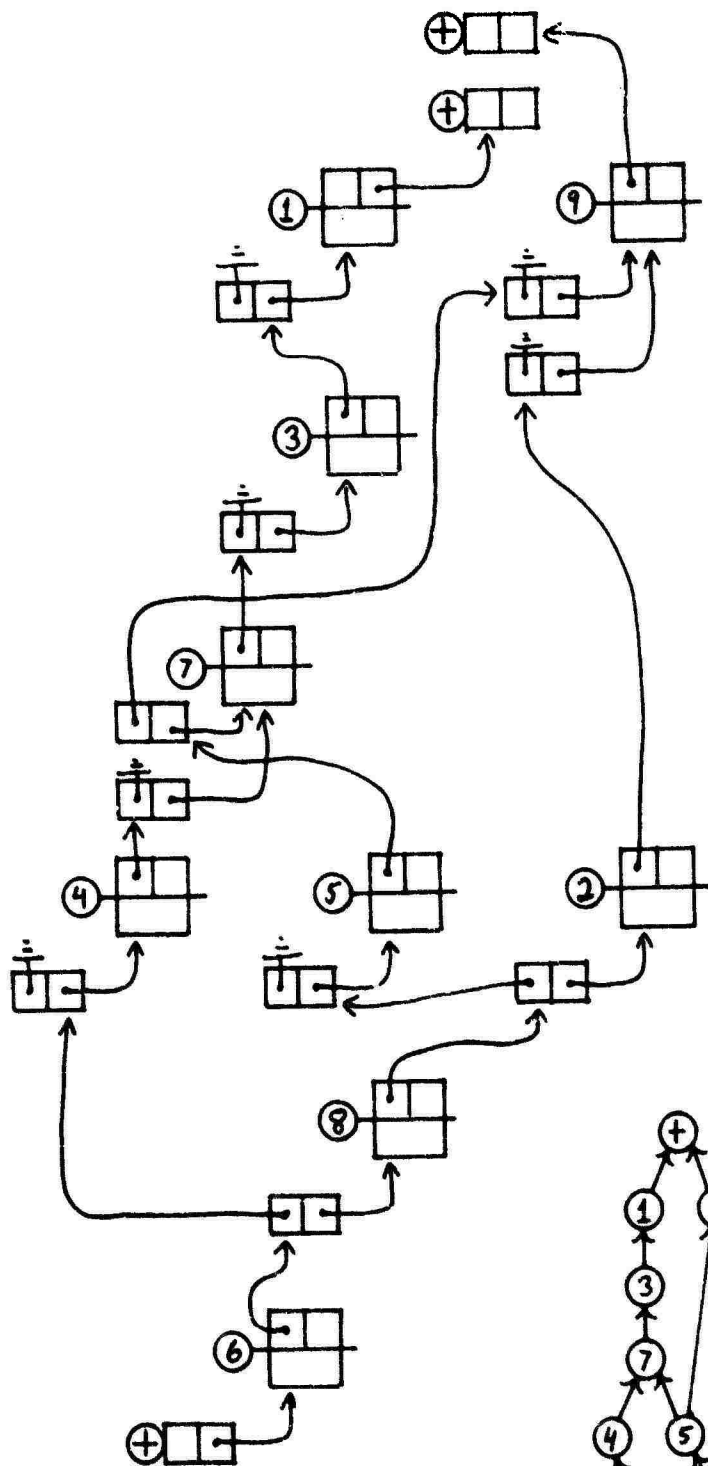


Figure 7D. Data Structures in ARCORE: linked lattice structure (successors only)





NXTLK	LATLK
-------	-------

LTPFFN  
(arcore)

PRLST	

ALTPFFN  
(arcore)

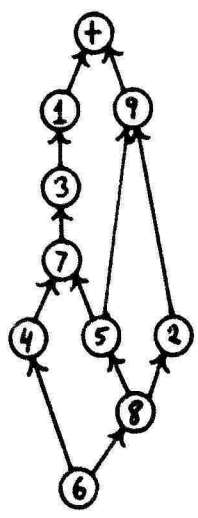


Figure 7E. Data Structures in ARCORE: linked lattice structure (predecessors only)

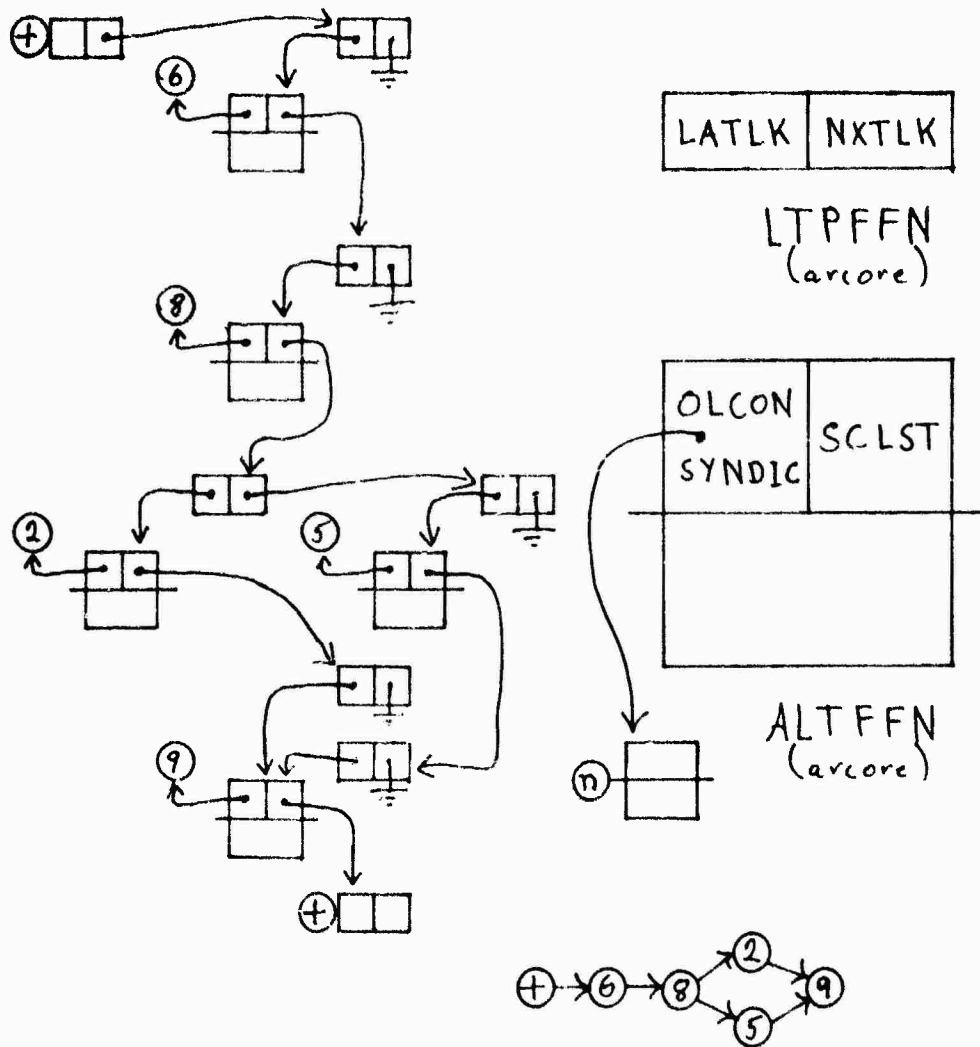


Figure 7F. Data Structures in ARCORE:  
lattice structure during parse-time winnowing

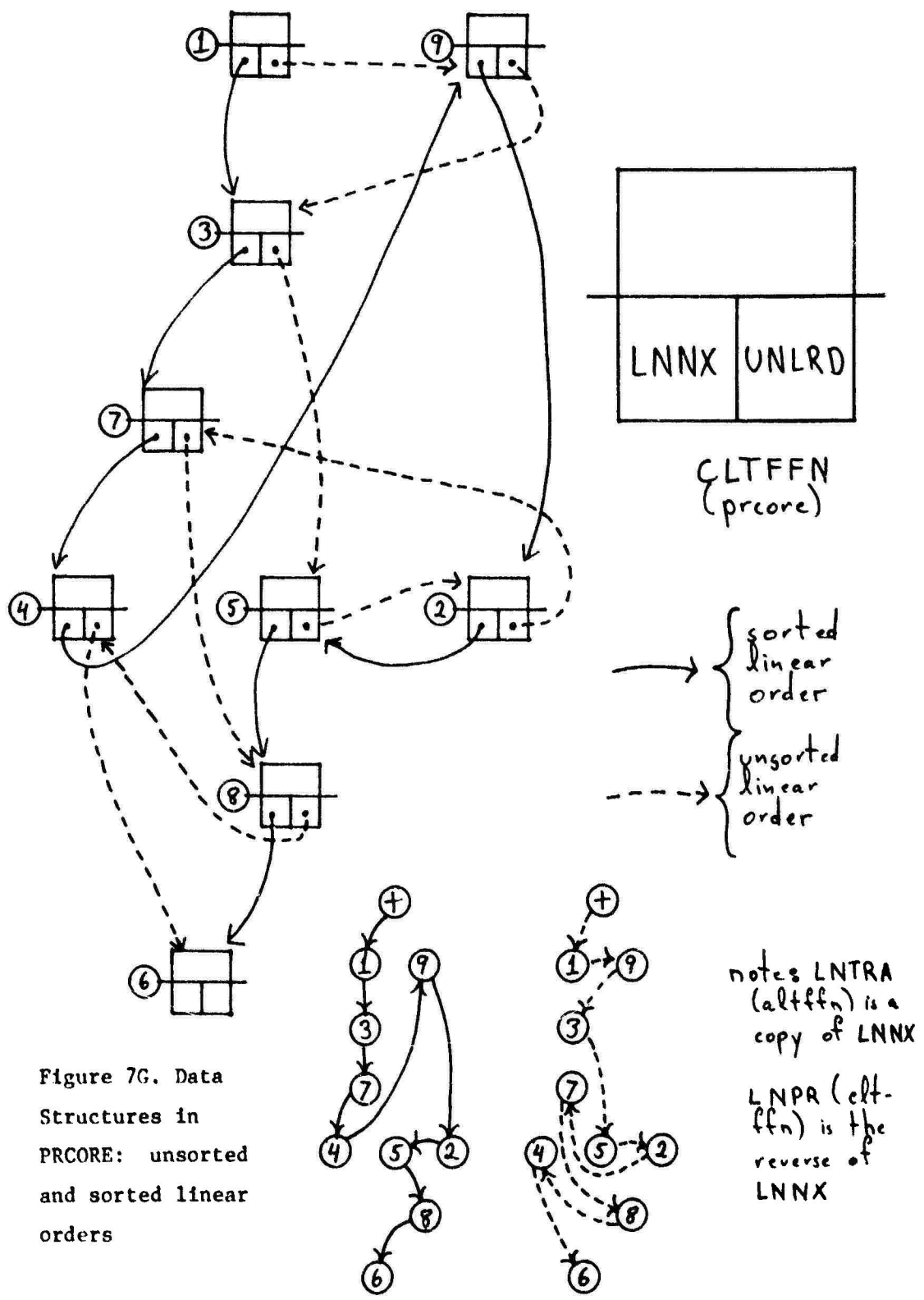


Figure 7G. Data Structures in PROCORE: unsorted and sorted linear orders

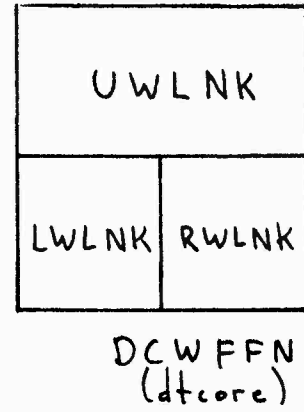
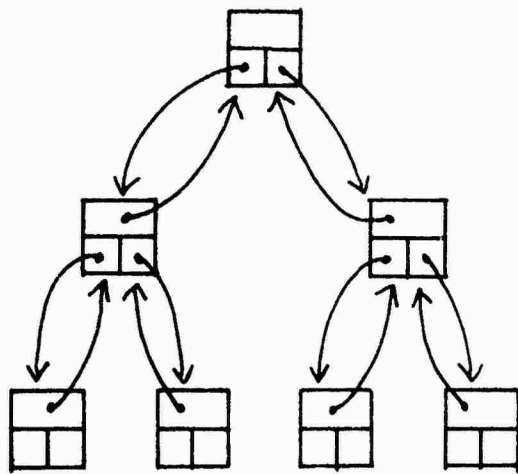


Figure 7H. Data Structures in DTCORE:  
balanced tree structure

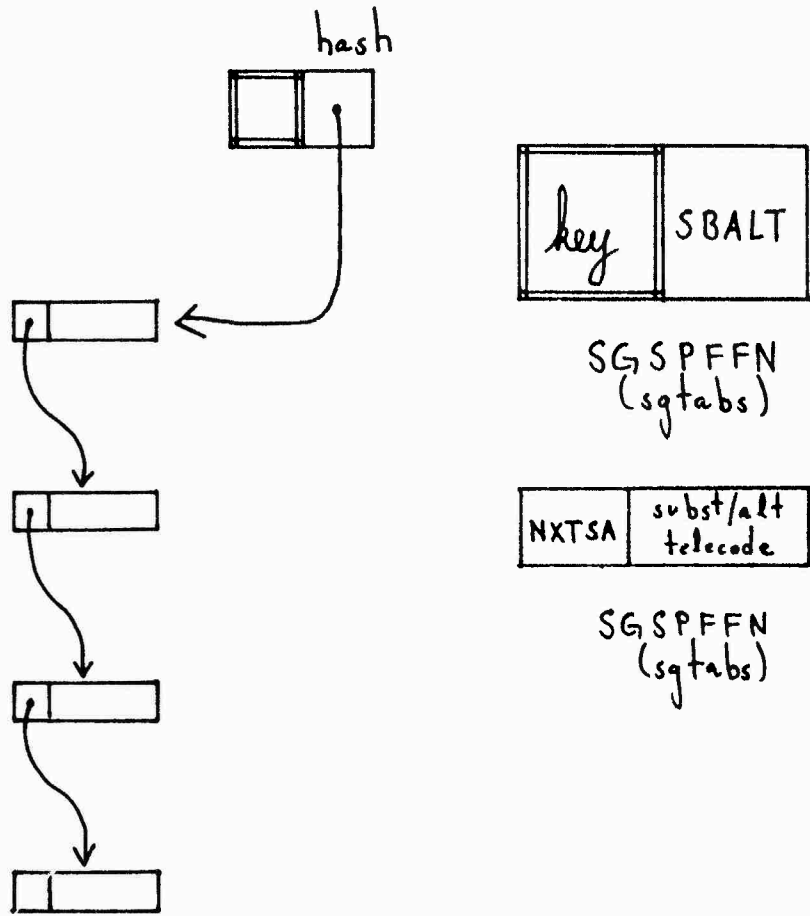


Figure 7I. Data Structures in SGTABS:  
linked segment table

CANONZ	CNCORE (canoni- zer)			LOCORE (low core resident)
INVEST	IVCORE (invest)	SBCORE (telecode substitu- tion)		
SELECTV	SLCORE (selectv)	LUCORE (lookup)	CSCORE (category symbols)	NMCORE (low core name table)
WINNOW				
ADAPTG	AGCORE (adapted grammar)	SRCORE source rules)		
PARSER		PRCORE (parse)	ARCORE (archive consti- tutes, lattices)	
STREXT	STCORE (string extract, uproot, transfer)			
RANDIC	DTCORE (diction- ary page)	SGDATA (segment data)	SGTABS (segment tables)	
ALCHEM	STRSAS (string extraction, temporary for SAS)			

Table 1. Module Storage Tables Resident  
for Each of the Quince Modules

CANONZ	CNCORE	
INVEST	IVCORE	IVSFFN - invest segment table IVTFFN - invest text table
	SBCORE	SBHFFN - telecode substitution hash table SBSFFN - telecode substitution table SBTFFN - telecode list table for substitutions
	LUCORE	LPTFFN - lookup spans table LQHFFN - lookup span queue table LQTFN - lookup queue tree table LSPFFN - lookup span table
	SLCORE	
	CSCORE	CSIFFN - category symbol data table CTHFFN - category symbol hash table
ADAPTG	SRCORE	
ADAPTG/ PARSER	AGCORE	ARLFFN - adapted rules table QTRFFN - winnowing queue tree table
	PRCORE	CLTFFN - lattice parse-time aux. table CRCFFN - constitute parse-time auxiliary table CSUFFN - category symbol parse-time data table SPQFFN - sentence position queue heads table
PARSER	ARCORE	ALTFFN - archive lattice table ARCFN - archive constitute table LTPFFN - lattice list next list pointers

Table 2. Field Function Tables Located in Each  
of the Module Storage Tables

STREXT	STCORE	STRFFN - master tree table
RANDIC	DTCORE	CDWFFN - word nodes for dictionary page
	SGDATA	SGATFFN - pointer to active telecodes
		SGSDFFN - telecode span list
		SGTTFFN - list of active telecodes
	SGTABS	SGSNFFN - substitutes/alternatives for telecodes
		SGSPFFN - telecodes for substitution/alternation
SGSTFFN - segmentation breaks table		
ALCHEM	STRSAS	

Table 2. (cont.)



Module Storage TableField Function Table

name-1	name-2	name-3	name-1	name-2	name-3
CNCORE	CNCOR	CNBLK			
IVCORE	IVCOR	IVBLK			
			IVSFFN	IVSFF	IVSGTB
			IVTFFN	IVTFF	IVTXTB
SBCORE	SBTCO	SBTBLK			
			SBHFFN	SBHSH	SBHTAB
			SBSFFN	SUBST	SBSTAB
			SBTFFN	SBTLC	SBTTAB
LUCORE	LUCOR	LUBLK			
			LPTFFN	LPTFF	LPTXTB
			LQHFFN	LQHFF	LQHDTB
			LQTFN	LQTF	LQTRTB
			LSPFFN	LSPFF	LSPNTB
SLCORE	SLCOR	SLBLK			
AGCORE	AGCOR	AGBLK			
			ARLFFN	ARLFF	ARLTAB
			QTRFFN	QTRFF	QTRTAB
CSCORE	CSCOR	CSBLK			
			CSIFFN	CSEQU	SCIBAS
			CTHFFN	CSEQU	CTHTAB
SRCORE	SRCOR	SRBLK			
PRCORE	PRCOR	PRBLK			
			CLTFFN	CLTFF	CLTCTB
			CRCFFN	CRCFF	CRCNST
			CSUFFN	CSUFF	CSUSTB
			SPQFFN	SPQFF	SPQHDS
ARCORE	ARCOR	ARBLK			
			ALTFFN	ALTFF	ALTCTB
			ARCFN	ARCF	ARCNST
			LTPFFN	LTNFF	LTCNXT
DTCORE	DTCM	DCTOR			
			DCWFFN	DWDEQ	DCTWDS
SGDATA	SGDCM	SGDCOR			
			SGATFFN	ACTEQ	ACTNDS
			SGSDFN	SPNEQ	SPNDS
			SGTFFN	TXTEQ	TXTDS
SGTABS	SGTCM	SGTCOR			
			SGSNFFN	SBPEQ	SBPTRS
			SGSPFFN	SBPEQ	SBPTRS
			SGSTFFN	BKSEQ	SEGBKS
STCORE	STCOR	STRES			
			STRFFN	STABL	STREE
STRSAS	STRSA	STRTEM			
LOCORE	LOCOR	RESNT			
NMCORE	NMCR	NMBLK			

Table 3. Names of the Module Storage Tables, Field Function Tables, and Interface File Definitions

Interface File Definitions

CZTDEF	CZTCR	CZTBLK
SGTDEF	SGTCR	SGTBLK
VSTDEF	VSTCR	VSTBLK
PDCDEF	PDCCR	PDCBLK
SLDDEF	SLDCR	SLDBLK
SDCDEF	SDCCR	SDCBLK

Table 3. (cont.)

Interface File Definitions

CZTDEF	BDCZT1
PDCDEF	BDPDC1
DSCDEF	BSDSC1
SGTDEF	BDSGT1
SLDDEF	BDSL1
VSTDEF	BDVST1

Module Storage Tables

AGCORE	BDAGC1
ARCORE	BDARC1
CNCORE	CDCNZ1
CSCORE	BDCS1
IVCORE	BDIVC1
LUCORE	BDLUC1
PRCORE	BDPRS1
SBCORE	BDSET1
SLCORE	BDSL1
SRCORE	BDSRL1
STCORE	BDSTR1
STRSAS	BDSTS1

Resident Tables

LOCORE	BDLOC1
NMCORE	BDNAME

Table 4. Block Data Subprograms  
for Each of the Common Blocks

## I. I/O Routines

### A. random-access disk I/O (COMPASS)

CLDISC

LRDISC

MTDISC

NMDISC

OPDISC

RDDISC

WRDISC

### B. ECS storage (COMPASS)

RE

WE

### C. System registers (COMPASS)

READRG

WRITRG

### D. system files

#### 1) binary files

OPBIN

CLBIN

REWIB

WREOFB

RDBIN

WRBIN

#### 2) coded files

OPCOD

CLCOD

REWINC

WREOFC

RDCOD1

WRCOD1

Table 5. Utility Routines Classified  
by Function

3) serial coded files

READS

WRITES

II. Format Conversion Routines (COMPASS)

A. packed machine-dependent telecodes

ARTOTC

TCTOAR

TCCOMP

B. binary/decimal/integer/display/Al

BTOD24

BTOO24

ITOC

CTOI

ARFORM

RAFORM

C. non-FORTRAN character

NFORTC

D. justification

LJUSTC

RJUSTC

LJUSTI

RJUSTI

III. Other Routines

A. collating sequences

CHCODE

CODECH

B. shift

LLS

LRS

Table 5. (cont.)

C. field insertion

INBUF

D. string equality

STREQ

IV. Debugging and Optimization Routines

A. Traceback

TRCBAK

DBOUT

ERROR

PRTBE

PRTBP

PRTBS

NARCS

RECOVR

B. Timing

TIMER

PRGRAF

C. Machine environment

CONFIG

Table 5. (cont.)

## References

1. Gaskins, Robert. Manual of GASP. Unpublished paper.
2. Gaskins, Robert. Probative Parsing. Unpublished paper.
3. Wang, William S-Y. and Stephan W. Chan. 1974. Development of Chinese-English Machine Translation System. University of California. RADC-TR-74-22, Final Report, February 1974.
4. Wang and Chan. 1975. Chinese-English Machine Translation System. University of California. RADC-TR-75-109, Final Report, April 1975.

## APPENDIX: HARDWARE, SOFTWARE AND DATA INVENTORY

### 1. Hardware Inventory

- 1.1 Teletype KSR-37 Terminal, with upper and lower case, 150 baud. At present it can be connected via modem with the CDC 6400 at the Lawrence Berkeley Radiation Laboratory, and also into the ARPANET.
- 1.2 Chinese Teleprinter Model 600D, 2 sets. Each set has a configuration consisting of a Chinese character keyboard, a printing unit for direct hard-copy output, a paper tape punch and a reader, and a slightly modified standard teletype with a standard English keyboard.
- 1.3 DEC DL-11E Asynchronous Serial Interface -- for character display on DEC PDP 11/20 - VT-11 display.

### 2. Software Inventory

#### 2.1 Quince System

The Quince system modules are contained in three libraries, each of which is stored on tape and maintained in both source and object form in a set of three cycles each.

1. DMLIB, the Data Management Library
  - a. field function definitions
  - b. field functions
  - c. common block allocator definitions
2. UTLIB, the Utilities Library -- all system- or machine-independent routines, both in assembler (COMPASS) and Fortran
3. QULIB, the Quince Library
  - a. GASP source of all system-independent subprograms
  - b. Block Data subprograms

#### 2.2 Program Writing System

The Program-writing system is a body of locally-written software aids for creating and maintaining large bodies of code. Each is stored on its own tape.

1. GASP Fortran Translator
2. Field Function Writer
3. Common Block Allocator



## References

1. Gaskins, Robert. Manual of GASP. Unpublished paper.
2. Gaskins, Robert. Probative Parsing. Unpublished paper.
3. Wang, William S-Y. and Stephan W. Chan. 1974. Development of Chinese-English Machine Translation System. University of California. RADC-TR-74-22, Final Report, February 1974.
4. Wang and Chan. 1975. Chinese-English Machine Translation System. University of California. RADC-TR-75-109, Final Report, April 1975.

## 2.3 Other Software

1. SAS -- Syntactic Analysis System -- predecessor to Quince system

2. Plot Routines

The plot routines permit the graphic display of trees and Chinese characters for research purposes; they are actually part of the previous Syntactic Analysis System, with data interface via the SAS Compatibility Module (ALCHEM) of the Quince system.

a. plotting subprograms

b. vector definitions of 7000 Chinese characters

## 3. Data Inventory

### 3.1 Chinese-English Dictionaries on Tape

1. CHIDIC (approximately 80,000 records)

2. PHYDIC (approximately 40,000 records)

3. McGraw-Hill Scientific Dictionary (partial) on 5 reels

4. DOD Chinese-English Scientific Dictionary (approximately 500,000 records) on 4 reels

5. Special sorts on CHIDIC

(i) one-telecode entries

(ii) long entries (more than 3 telecodes)

(iii) reverse telecode sort

(iv) grammar code sort

6. Special sort on PHYDIC

(i) grammar code sort

### 3.2 Chinese Grammars

The Chinese grammar consists of five levels. The total number of rules at each level is:

Grammar 1                    124 rules

Grammar 2                    506 rules

Grammar 3                    2408 rules

Grammar 4                    336 rules

Grammar 5                    2744 rules

There are three ways of arrangement of the rules:

1. five-level grammar by levels

2. five-level grammar by length

3. concordance of rules

### 3.3 Chinese Texts

#### 1. Physics Texts (papers numbered 1 to 37).

Total telecodes: 421,464

The Physics Texts were coded from the following books and articles:

- a. Yuanzineng jichu zhishi. Huadong Shifan Daxue. 1958. 114pp.  
原子能基礎知識. 華東師範大學
- b. Yuanzineng de yuanli he yingyong. Kexue Chubanshe. 1965.  
原子能的原理和應用. 科學出版社
- c. Yuanziheneng. Kexuejishu Chubanshe. 1957. 262pp.  
原子核能. 科學技術出版社
- d. Yuanzineng de jiben lilun yu yuanzineng de heping gongxian.  
(no publisher). 1966. 44pp.  
原子能的基本理論與原子能的和平貢獻
- e. Ci liuti lixue 磁流體力學
- f. Gaowen dengliziti donglixue 高溫等離子體動力學
- g. Gaowen dengliziti de fushe 高溫等離子體的輻射
- h. Gaowen dengliziti zhenduan fangfa 高溫等離子體診斷方法
- i. Tongweisu he shexian de yingyong (xia)  
同位素和射線的應用(下)
- j. Jige xinde ji jingguo gaizhuang de yanjiuxing rezhongzi fangying-  
dui. 几个新的及經過改裝的研究性熱中子反應堆
- k. PQT fanyingdui zhongzi tongliang de zengjia ji shiyan kenengxing  
de kuoda PQT反應堆中子通量的增加及實驗可能性的擴大
- l. Shiyanxing qingshui nongsuoyou fanyingdui (BBP-2) de gaizhuang  
試驗性輕水濃縮鈾反應堆(BBP-2)的改裝
- m. Zhongshui fanyingdui (TP) de gaizhuang  
重水反應堆(TP)的改裝
- n. Gongli wei 2000 wa de chenruxing shiyan fanyingdui (MPT)  
功率為2000瓦的沉入型試驗反應堆(MPT)
- o. Rezhongzi tongliang  $10^{14}$  zhongzi/limi<sup>2</sup>. miao de yanjiu fanyingdui  
(BBP-M) 熱中子通量 $10^{14}$ 中子/厘米<sup>2</sup>. 新的研究反應堆
- p. Fushe huaxue jianjiu zhuanyong fanyingdui (BBP-U)  
輻射化學研究專用反應堆

q. Yuanzi he yuanzineng. Jiaoyu Tupian Chubanshe. 1956. 121pp.

原子和原子能. 教育图片出版社

2. Biochemistry Texts (papers numbered 1 to 17).

Total telecodes: 59,320

3. "Tokuyama" Texts (sample excerpts from modern Chinese short stories ca. 1920-30, obtained on a cooperative project with Dr. Helen Tokuyama of the University of California at Irvine).

Total telecodes: 83,830

Total Machine-Readable Text Telecoded: 564,614

The Physics Texts c, e through q and the Tokuyama Texts exist both on magnetic tape and on the original Chinese Teleprinter paper tape. The rest of the Physics Texts and the Biochemistry Texts exist in 80 column cards.

#### 3.4 Chinese Character I/O Information

1. Kuno character vectors (7,000 records)

2. Telecode-Romanization Table (10,000 cards)

3. Chinese Teleprinter Keyboard to Telecode Table (4,800 cards)

4. Four-Corner System Romanization Equivalences (1,500 cards)

5. Original Cards for Chinese Character Indexes Volumes (14,000 cards)

6. Augmentation to Chinese Character Indexes, with romanization equivalents (14,000 cards)

## Supplements

- |  |          |
|--|----------|
| 1. Test Canonization Module            | (CANONZ) |
| 2. Vestigand Formation Module          | (INVEST) |
| 3. Dictionary Lookup Module            | (LOOKUP) |
| 4. Probative Parser Module             | (PARSER) |
| 5. Parse Table Print Module            | (PARPNT) |
| 6. String Extraction Module            | (STREXT) |
| 7. Grammar Adaptation Module           | (ADAPTG) |
| 8. Random Dictionary Module            | (RANDIC) |
| 9. SAS Compatibility Module            | (ALCHEM) |
| 10. Quince Utilities Module            | (QUTILS) |
| 11. Loader Storage Allocation Module   | (BLKDAT) |
| 12. Common Block Definitions           | (CBADEF) |
| 13. Field Function Definitions         | (FFNDEF) |
| 14. GASP System Language Pre-processor | (GSPSRC) |

### III. THE STATE OF THE ART IN COMPUTATIONAL SYNTACTIC DESCRIPTION OF NATURAL LANGUAGES

#### 1. Introduction

When the recent history of linguistics is viewed from the perspective of computational linguistics and machine translation, it may fairly be said that the most conspicuous event remains the introduction of the context-free phrase-structure grammar by Noam Chomsky in the middle 1950's. Despite the variety of alternative formalisms for the description of languages which have been introduced (by Chomsky and others) in the intervening twenty years, it is still the context-free grammar which dominates the thinking of computational linguists and dominates, also, the systems which they devise.

There are, however, certain technical difficulties with the use of context-free grammars which have led computational linguists to "augment" their grammars with "features", and with "conditions" or "actions" based on the features. This is true of the well-known systems of today (e.g., Woods' ATN grammars and Winograd's systemic grammars are of this kind), and has been true stretching back to the days of the COMIT programming system of Yngve. Most computational linguists believe that such augmented context-free grammars are sufficient to describe natural languages, at least in some rough practical way, although there is no real theory explaining how to use the augmentations, or why they are so helpful.

Such augmentation devices were also used (though much less formally and systematically, of course) by linguists of the pre-Chomskian American structuralist tradition to describe such phenomena as agreement and contextual restrictions; this is one aspect of their procedures which was never reconstructed satisfactorily in phrase-structure grammars. Furthermore, the lack of such augmentation devices has proved troublesome in current linguistic uses of context-free grammars, and they have now been re-introduced in the most recent work on the base component of Chomskian transformational grammars -- first with features of lexical items, and then with complex-

symbol representations for all nonterminals of the grammar.

In an entirely unrelated development, as a way of defining programming languages, the properties of the syntactic description method known as "van Wijngaarden grammars" or "the Algol 68 definition method" have recently become better understood. It now appears that this method of describing "context-free grammars with structured vocabulary" does reconstruct an important element common to structuralist linguists, recent Chomskian linguists, and computational "augmentations": that is, the use of significant abbreviatory conventions in context-free grammars by exploiting a systematically-structured vocabulary of symbols.

Not only does the van Wijngaarden syntactic description method appear to neatly cover a wide variety of extensions to context-free grammars and thus give insight into what important properties they share, but related formalisms (Koster's affix-grammars, Knuth's attribute-grammars) offer similar properties while also being naturally related to context-free grammars in the sense that the naturalness of interpretation and attractive parsing properties of context-free grammars are preserved. Hence, although these formalisms have all been developed in connection with programming languages, they appear to be of even greater interest and importance for the processing of natural languages.

In this chapter we will review the state of the art in defining grammars for natural languages which are suitable for computational use in machine translation systems, giving particular stress to the new methods just mentioned as models for a good deal of current unformalized practical knowledge. We will attempt to provide an overview of the progress in defining programming languages, and to relate the new features of this work to prior descriptive methods used by linguists. Finally, we indicate how this work is related to the parsing implemented in the Quince system at the Project on Linguistic Analysis, and what further research is needed over the next three to five years to incorporate these improved techniques.

## 2. Context-Free Grammars

We began with the observation that all computational systems currently used for research on natural language are based on context-free grammars, even if they also incorporate much additional machinery to interpret, or translate, or whatever. In this section we will review the advantages of the context-free grammar formalism which have led to this state of affairs, the extensions to the basic context-free grammar which are introduced to counter certain disadvantages, and the remaining difficulties.

### 2.1 Advantages of Context-Free Grammars

The reasons for the pre-eminent popularity of the context-free grammar formalism are many. First, perhaps, is the fact that a context-free grammar both defines a set of admissible strings, by giving a set of constraints on the ordering of elements, and also associates with each string in its language a hierarchical, tree-like structural description. It turns out that almost always one wishes both to separate valid strings from invalid, and also to assign structures to the valid ones; perhaps it is only so because the tool is at hand, but this has seemed a logical single task.

It is also true that in an amazingly wide range of applications the context-free grammar has seemed to be "a natural conceptual basis for definitions; the basis must correspond to the way we actually think about (what is being defined), otherwise the related formalisms are not likely to be fruitful" (Knuth 1971). In large part, context-free grammars have been such a "fruitful formalism" because of the declarative character of a grammar. Donald Knuth, again, says that "a grammar is 'declarative' rather than 'imperative'; it expresses the essential relationships between things without implying that these relationships have been deduced using any particular algorithm" (Knuth 1971). This notion of a grammar as a set of declarative "well-formedness conditions" is also familiar to linguists, from McCawley's discussion of the phrase-structure base component of a transformational grammar (McCawley 1968). (A frequent shortcoming of computational research on natural languages, especially that conducted by non-linguists under the name of "artificial intelligence", has been to extend context-free grammars in procedural ways, apparently out of a lack of appreciation for declarative



formalisms; see, e.g., Winograd 1971, 1975.)

Finally, not only is it true that relatively small context-free grammars are easy for human beings to devise, understand, and improve, but they are also easy for computers to manipulate. There have always existed algorithms for parsing with a context-free grammar, and in recent years extremely good algorithms have been described and refined in many variants appropriate for a wide range of purposes (Aho and Ullman 1973).

## 2.2 Disadvantages of Context-Free Grammars

There are, to be sure, some disadvantages of context-free grammars, and they spring to mind even more readily than do the advantages since they are a constant source of difficulty.

The theoretical difficulties may be dispensed with -- such things as the inability to have infinite branching from a single node (so as not to put an upper bound on the number of items in a coordinate structure), or the inability to deal with unbounded overlapping dependencies of the sort which are well-known to be a prominent feature of Mohawk (Postal 1964b). (Postal's criticism is sound, although just slightly askew; it is revised in Fidelholtz 1974.) These difficulties are true, but irrelevant. Such examples show that neither in terms of weak generative capacity (the sets of strings) nor in terms of strong generative capacity (the sets of structural descriptions) do context-free grammars provide a description of natural language surface structures; but as a practical matter they cause no particular trouble.

The fact that these are the wrong terms in which to discuss the adequacy of context-free grammars becomes clear from the observation that, if we simply restrict a natural language to sentences short enough to fit in six-point type between California and Alpha Centauri, then the restricted language will be finite and hence trivially a Chomsky type 3 (finite-state) language, and thus a fortiori context-free.

The point is that a grammar which is not clear enough to be invented and improved by human beings cannot be produced; and clarity, as Edsger Dijkstra observes, "has pronounced quantitative aspects" (Dijkstra 1972). The chief practical difficulty with context-free grammars for natural

languages has been their large size and their corresponding lack of transparency. Susumo Kuno (1963) reported on an English grammar containing 133 syntactic categories and over 2100 rules, which did not yet incorporate the obvious agreement restrictions of English. Grammars with upwards of 10,000 rules are known to exist.

As the number of rules grows into the thousands, and as it is realized that tens of thousands of rules would be only a beginning, all the practical advantages of context-free grammars disappear. Such grammars are no longer at all easy to understand, nor are they easy to manipulate for computer use.

A typical experience, repeated over and over throughout the 1960's and early 1970's, has been that a context-free grammar can be written readily to serve as an initial demonstration model over a limited range, but that replacing that context-free grammar with one adequate for actual natural language is, practically speaking, impossible. As Samuel Johnson wrote in the preface to his Dictionary of 1755, "a large work is difficult because it is large, even though all its parts might singly be performed with facility."

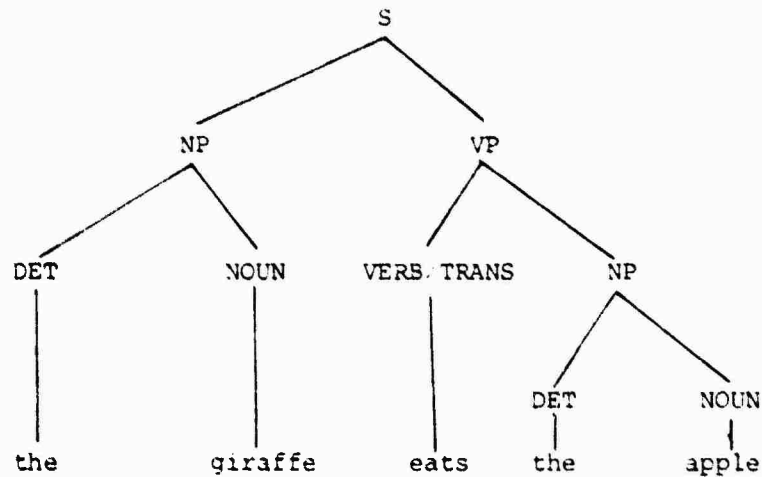
### 2.3 Why Context-Free Grammars Grow Large

Context-free grammars grow large beyond the effective power of humans to control them primarily because of the need to encode within them restrictions on contexts. This is, of course, not a contradiction or a paradox; the name "context-free" refers to the form of the rules in the grammar, not to any impossibility of utilizing context to restrict the language of the grammar. As every linguist should know by now, Peters and Ritchie (1973) contains a demonstration that every language which can be "analyzed" by testing putative structural descriptions using context-sensitive well-formedness rules is a context-free language -- that is, it also is generated by a context-free grammar, although a context-free grammar which may have many, many rules.

A small example of the way in which the need for context expands context-free grammars is given by Winograd (1971). He exhibits the grammar:

1.  $S \rightarrow NP VP$
2.  $NP \rightarrow DET NOUN$
3.  $VP \rightarrow VERB/INTRANS$
4.  $VP \rightarrow VERB/TRANS$
5.  $DET \rightarrow the$
6.  $NOUN \rightarrow giraffe$
7.  $NOUN \rightarrow apple$
8.  $VERB/INTRANS \rightarrow dreams$
9.  $VERB/TRANS \rightarrow eats$

which generates derivations such as:



Winograd points out, though, that to expand the grammar so as to include number agreement for subjects, thus giving:

The giraffes eat the apple.

The giraffe eats the apple.

but not:

\*The giraffes eats the apple.

\*The giraffe eat the apple.

requires (if we are to be strictly observant of the notion of a context-free grammar) that we introduce new category symbols to code the terminal

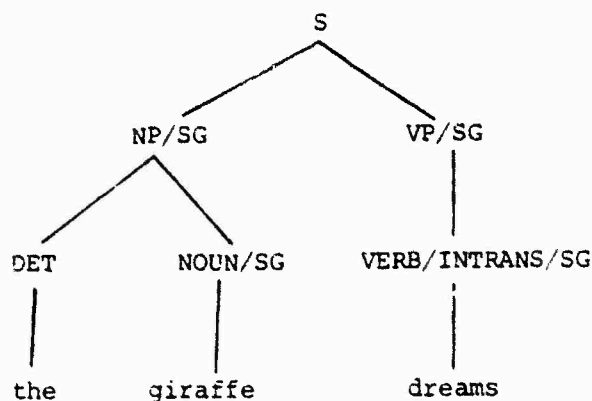
vocabulary. We must add rules:

- 6a. NOUN/SG → giraffe
- 6b. NOUN/PL → giraffes
- 8a. VERB/INTRANS/SG → dreams
- 8b. VERB/INTRANS/PL → dream
- 9a. VERB/TRANS/SG → eats
- 9b. VERB/TRANS/PL → eat

and we must also double the number of rules above the terminals, adding additional non-terminal vocabulary as necessary:

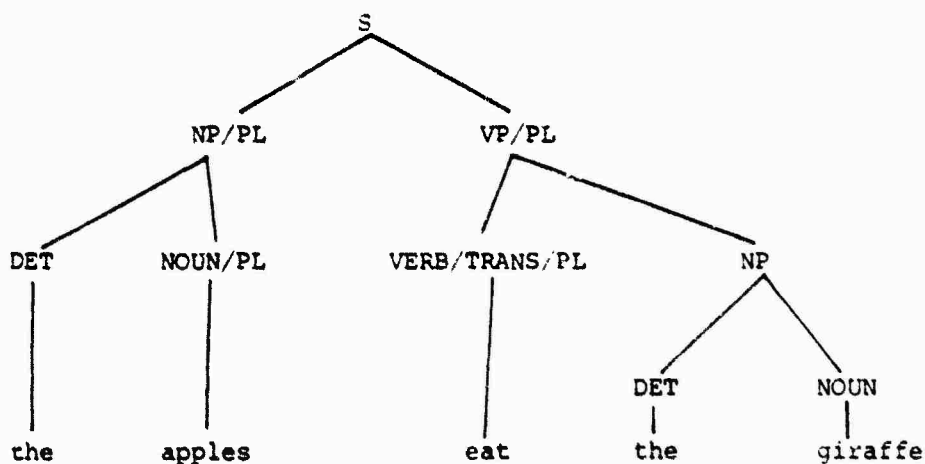
- 1a. S → NP/SG VP/SG
- 1b. S → NP/PL VP/PL
- 2a. NP/SG → DET NOUN/SG
- 2b. NP/PL → DET NOUN/PL
- 3a. VP/SG → VERB/INTRANS/SG
- 3b. VP/PL → VERB/INTRANS/PL
- 4a. VP/SG → VERB/TRANS/SG NP
- 4b. VP/PL → VERB/TRANS/PL NP

(Observe that two symbols in this grammar such as NP/SG and NP/PL are wholly distinct symbols from the standpoint of the definition. Their similarity in spelling is a help to the human reader in grasping the significance of the symbols in the grammar, but the grammar itself does not exploit the similarity.) We now have derivations such as:



This is straightforward, and it is clear that the way in which we can enforce number agreement in context is by duplicating the vocabulary of the grammar and the productions of the grammar all the way back from two items which must agree (such as NOUN/SG and VERB/INTRANS/SG) to their common parent (here, clear back to the start symbol S). If the agreement possibilities have three values (masculine, feminine, and neuter, say) then the symbols and the rules must be multiplied by three, and so forth.

It becomes discouraging to note that a similar multiplication will be required for every individual feature of context which must be coordinated -- next, for instance, we might notice that our grammar even with number agreement for subjects will derive:



(We can assume that the rule re-writing plain NP still exists in the grammar, because object number agreement is not necessary; alternatively we could double the VP rules again, to introduce freely both NP/SG and NP/PL as objects.) This would lead us to double once again to code the correct restrictions for "animate subject", leading to terminal vocabulary such as:

NOUN/SG/ANIM → giraffe  
NOUN/PL/ANIM → giraffes  
NOUN/SG/NONANIM → apple  
NOUN/PL/NONANIM → apples  
VERB/TRANS/SG/ANIMSUBJ → eats  
VERB/TRANS/PL/ANIMSUBJ → eat

and again we must double the rest of the productions, beginning with:

1a. S → NP/SG/ANIM VP/SG/ANIMSUBJ  
1b. S → NP/PL/ANIM VP/PL/ANIMSUBJ  
1c. S → NP/SG/NONANIM VP/SG/NONANIMSUBJ  
1d. S → NP/PL/NONANIM VP/PL/NONANIMSUBJ

Even though we have already passed the point of reasonableness, it is obvious that we must continue this process for a long time. As Winograd remarks, "this sort of duplication propagates multiplicatively through the grammar, and arises in all sorts of cases."

This then is the way in which the desire to encode context restrictions causes a context-free grammar to grow, by multiplying its sequences of rules over and over again.

#### 2.4 Attempts to Reduce the Size of Context-Free Grammars

The preceding analysis of how context-free grammars become unmanageable suggests that some method is needed to simplify grammars by indicating where all the parallel sets of rules occur. In practice, virtually every serious use of context-free grammars has introduced some

such mechanism (even if only informally), and such mechanisms will be the subject of following sections.

In addition to these methods, however, there have been at least two attempts to reduce the size of large grammars by a process of factoring them into smaller grammars, a sort of "divide and rule" strategy.

The first of these was Woods's notion of creating a "regular expression grammar" (Woods 1969). In principle, this consists of an algorithm which takes an arbitrary context-free grammar and factors it into a set of regular (type 3) grammars, plus the essentially context-free transitions between them. In practice it seems that all examples have been composed by hand in already factored form. In addition to the gain of breaking a larger context-free grammar into a number of smaller grammars, it was pointed out that the smaller grammars could be improved by using optimization techniques applicable to regular grammars. Such grammars are of some interest, and they need not be developed in the "procedural" AI terminology of transition networks as Woods has chosen to develop them. (See Lalonde 1977 for a development as "regular right-part grammars" leading to a theory and an implementation which appear more attractive than those of Woods.)

Whereas Woods broke up grammars "horizontally", the other attempt was to break up grammars "vertically" into smaller pieces applied sequentially; this was the "hierarchical sub-grammar" mechanism introduced in the Quince system of the Project on Linguistic Analysis (Wang and Chan 1975, Gaskins 1973). The POLA technique required human-separation of a large grammar into pieces, with the non-terminal symbols of some grammars serving as terminals of other grammars. In the development of such grammars it was thus possible to separate some concerns, because in applying them to parsing it was possible to utilize alternative grammars depending on the tree-tops developed by a preceding grammar application.

Both of these ways of making one large context-free grammar into several smaller ones are useful, though they address such different goals that it is impossible to compare their relative effectiveness (and they are not mutually exclusive). But they have in common that neither really does much about the multiplicative duplication of vocabulary symbols and rules previously described. Hence, it has been necessary to augment both

schemes with additional extensions to control multiplicative duplication, which would still be crippling if not contained.

### 3. A Model for Context-Free Grammars with Structured Vocabulary

Once we have identified the problem of multiplicative duplication of vocabulary and rules in a context-free grammar, it is tolerably obvious what to do about it: we must introduce a notational system which allows the process of duplication to be implied, rather than requiring that it be carried out at full length. In fact, this step is so obvious that it has been taken by almost everyone who ever wrote grammars to be read by human beings, but the variety of different notions and notations introduced has been so bewildering that the similarities have not generally been appreciated. For this same reason the device has not been developed as well as it could be for use in descriptions of natural languages.

There is now an elegant and comprehensive notation available for writing context-free grammars with systematically-structured vocabularies and productions, and this is the "two-level" grammar devised by Adrian van Wijngaarden for the definition of the programming language Algol 68. The van Wijngaarden grammars (in some intuitive sense) cover and include all the other proposals, and are the simplest technique available. This descriptive system is unfortunately not widely known, despite (perhaps because of) the publicity given to the language Algol 68.

In this section, accordingly, we will introduce the notion of a van Wijngaarden grammar. We will then relate it to a number of descriptive techniques used by linguists, to a number of extensions of context-free languages used by computational linguists, and also to related formalisms used by computer scientists to define formal languages and programming languages -- in particular, to the attribute grammars of Donald Knuth and the affix grammars of C. H. A. Koster.

It is not at all clear -- a reader should know in advance -- that it would be wise to adopt the van Wijngaarden grammar format as an actual encoding of rules to be used for computational analysis; but the van Wijngaarden grammar abstracts the essential problem so cleanly that it offers the indispensable framework of insight within which related formalisms



can be understood and compared with one another.

### 3.1 van Wijngaarden Grammars

The idea of a "two-level" grammar (or "van Wijngaarden grammar", sometimes also W-grammar or vW-grammar) was originated by Adriaan van Wijngaarden, for many years the director of the Mathematisch Centrum at Amsterdam, as a method for describing the programming language Algol 68 then under development by an international committee (van Wijngaarden 1965, van der Poel 1971). A van Wijngaarden description of Algol 68 appeared in 1969 (van Wijngaarden et al. 1969) which failed to exploit the potential of the method; a revised description appeared in 1975 (van Wijngaarden et al. 1975), which for the first time showed to advantage the two-level grammar idea, and interest in the method revived to some degree (see Cleaveland and Uzgalis 1977).

Unfortunately for the descriptive method, the Algol 68 language has not been well received (specimen reaction, attributed to P. Z. Ingerman: "This language fills a much-needed gap"), and the widespread distaste for the language Algol 68 has contributed to the unpopularity of the method specially devised to describe it. Worse still, the Algol 68 Report introduced a slightly different notation for a context-free grammar, and this impeded discussion further. (The Algol 68 Report notation for a van Wijngaarden grammar will not be used here, but a short specimen is included at the end of section 3.2) But the method of syntax description is really elegant and important, and can be divorced from Algol 68. It should be better known to computer scientists and linguists.

Let us begin with a couple of examples of van Wijngaarden grammars; after the examples the terminology will be reviewed at length and made more precise. In section 2.3 above, we considered a grammar from Winograd with rules such as

- 1a.  $s \rightarrow np/sq \quad vp/sq$
- 1b.  $s \rightarrow np/pl \quad vp/pl$
- 2a.  $np/sq \rightarrow det \quad noun/sq$
- 2b.  $np/pl \rightarrow det \quad noun/pl$

and so forth. (The use of lower-case letters to spell the symbols is a change, which will be explained presently, but obviously this is the same grammar as when upper-case letters were used.)

We could abbreviate these rules by taking advantage of the structure which is in the vocabulary of symbols -- i.e., the systematic relations between the pairs of symbols such as "np/pl" and "np/sg". We can introduce a cover term for either "sg" or "pl", and write the cover symbol as "NUM" (using upper-case letters for cover symbols, lower-case letters for the others). Using this abbreviation, the rules become

h1.  $\langle s \rangle \rightarrow \langle np/ NUM \rangle \langle vp/ NUM \rangle$

h2.  $\langle np/ NUM \rangle \rightarrow \langle det \rangle \langle noun/ NUM \rangle$

and so forth. (We will use the angle-brackets to surround the symbols in the grammar, since each one may consist of more than one piece such as "noun/" and "NUM", and the brackets aid in visual parsing of the rules by a human reader.) We must now understand each of these rules as a "rule schema", a pattern for generating the rules given before by plugging in various values for "NUM". In order to make this idea precise, we will specify the values that "NUM" can assume by a separate context-free grammar, a "meta-grammar":

m1.  $NUM ::= sg \mid pl$

It is very important also to stipulate that the same value of NUM must be inserted into each occurrence of NUM in a rule schema. For instance, there is not a rule such as "  $s \rightarrow np/sg \ vp/pl$  ", because that could only result from replacing NUM in rule h1. with 'sg' at its first occurrence, but with 'pl' at its second occurrence. This principle we will refer to as the Uniform Replacement Convention (URC).

These two sets of rules above constitute a "two-level" van Wijngaarden grammar corresponding to the original context-free grammar (the four rules from Winograd) given just previously. Those four original rules are represented in the van Wijngaarden grammar by (a) rule schemata which incorporate variables in context-free rules (like rules h1, h2 above), and (b) a second context-free grammar whose terminal strings are the permitted values of the variables (like rule m1 above). Terminal strings of this

grammar are substituted for variables in the rules of other grammar, subject to the Uniform Replacement Convention.

The name used for the variables is "meta-symbols". The second grammar, naturally enough, is called the "meta-grammar" because it defines the meta-symbols. The first grammar is called a "hyper-grammar", into which the substitutions are made. Accordingly, we have three kinds of rules:

(1) Meta-rules are context-free rules which define the possible values of meta-symbols:

m1. NUM ::  $\rightarrow$  sg | pl

(2) Hyper-rules are schemata for context-free rules, whose symbols may contain meta-symbols:

h1.  $\langle s \rangle$  :  $\rightarrow$   $\langle np/ NUM \rangle$   $\langle vp/ NUM \rangle$

h2.  $\langle np/ NUM \rangle$  :  $\rightarrow$   $\langle det \rangle$   $\langle noun/ NUM \rangle$

These two sets of rules together make up a "two-level" van Wijngaarden grammar. They define the language generated by their production rules:

(3) Production-rules are the context-free rules which can be produced from the schematic hyper-rules by systematic replacement of meta-symbols according to the Uniform Replacement Convention.

The meta-rules and hyper-rules of the van Wijngaarden grammar above specify the four production-rules:

p1.  $s \rightarrow np/sg \quad vp/sg$

p2.  $s \rightarrow np/pl \quad vp/pl$

p3.  $np/sg \rightarrow det \quad noun/sg$

p4.  $np/pl \rightarrow det \quad noun/pl$

In this case, as very frequently happens, the two sets of rules in the van Wijngaarden grammar (the meta-rules and the hyper-rules) specify a set of production-rules which could perfectly well be written down in full, as we have just done. But the van Wijngaarden format is shorter (not much here, but often very much shorter), and more importantly the van Wijngaarden grammar preserves the information that rules which

differ only in the number-agreement specified ("NUM") are really two instances of the same rule schema. This formal generalization corresponds to a linguistic claim that sentences with singular subjects do not have a different grammar from sentences with plural subjects; in fact the grammar is the same, but there must be number agreement between the verb and its subject. Also, in the van Wijngaarden format the rules for sentence formation could be modified by changing just the appropriate single rule schema (hyper-rule), leaving the definition of the meta-variable NUM in the meta-rules unchanged.

The preservation of this kind of structure in the vocabulary of symbols is a most important feature of van Wijngaarden grammars for natural languages, and it amounts to more than just a clever abbreviation for an ordinary context-free grammar. As we will see later, a two-level grammar can define languages which have no context-free grammar, and van Wijngaarden grammars are actually equivalent to Chomsky type-0 grammars, or unrestricted rewriting systems.

It will be apparent to every linguist that the use of meta-symbols in context-free grammars is an old custom in linguistic description (and we will examine some of those older uses below); the distinctive contributions of the van Wijngaarden grammar are some of the techniques for exploiting such meta-symbols, and the explicit use of (what else?) a second context-free grammar to derive the meta-symbols. The whole arrangement seems extremely obvious, but it turns out to have some very non-obvious properties.

### 3.2 Notation for van Wijngaarden Grammars

Since we now have three kinds of rules, three kinds of symbols, and so forth, it is best to have a very clear notation for keeping them separate. This section introduces the full nomenclature in a step-by-step fashion, and at the end of the section there is a reference summary of the notation which can be consulted while reading the remainder of this chapter.

We will use upper-case letters for meta-symbols, and lower-case letters for symbols such as the ones which can be derived from meta-symbols (we call these lower-case symbols proto-symbols, but their name

seldom comes up). The grammar of the meta-symbols is the meta-grammar. Each meta-rule in such a meta grammar will take the form of expanding one single meta-symbol on the left side of a meta-rule into a string of meta-symbols and proto-symbols. Thus, the meta-symbols are the non-terminals of the meta-grammar and the proto-symbols are its terminals; that is why meta-symbols are upper-case and proto-symbols are lower-case, as is customary in an ordinary context-free grammar written according to the usual Chomsky conventions. For example, a meta-grammar of three meta-rules is the following:

```
SUBJ ::→ ANIMATE
ANIMATE ::→ minus-animate | plus-animate HUMAN
HUMAN ::→ minus-human | plus-human
```

where SUBJ, ANIMATE, and HUMAN are the non-terminals (meta-symbols), and minus-animate, plus-animate, minus-human, and plus-human are the terminals (proto-symbols). (There is really no requirement to spell out "plus" or "minus", but the style customary in writing these grammars is to use long names for symbols -- probably a bad style.)

In the meta-rules, the symbol ::→ is used for the "re-write" symbol; a different re-write symbol is used in each type of grammar so that a single rule in isolation can always have its type identified. Any rule with the double-colon arrow is necessarily a meta-rule. The usual vertical bar is used to separate alternatives on the right side of rules, and the same bar is used in all grammars. Alternatives can always be written as additional rules at the option of the grammar writer, so the three meta-rules above are equivalent to the five (unabbreviated) meta-rules:

```
SUBJ ::→ ANIMATE
ANIMATE ::→ minus-animate
ANIMATE ::→ plus-animate HUMAN
HUMAN ::→ minus-human
HUMAN ::→ plus-human
```

It is necessary to stress that these meta-grammars are to be interpreted as utterly-ordinary context-free grammars. The only feature which is slightly unusual is that you are permitted to choose any non-terminal as the start-symbol for the grammar, and then the values of that non-terminal are the strings it derives in the meta-grammar. For example, in this meta-grammar if SUBJ is chosen as the start symbol, it gives three possible strings of terminal proto-symbols:

SUBJ derives:     minus-animate  
                  plus-animate minus-human  
                  plus-animate plus-human

So, in the rule schemata, wherever SUBJ appears it could be replaced with any of these three strings. But if HUMAN is treated as the start symbol of the meta-rules, then HUMAN only leads to two strings of terminal proto-symbols:

HUMAN derives:    minus-human  
                  plus-human

and so where HUMAN is used in the rule schemata it could be replaced only with one of these two strings.

Moving now to the other component of a van Wijngaarden grammar, we will call the grammar of the rule schemata the hyper-grammar. The symbols used in the hyper-grammar are hyper-symbols, which are strings of proto-symbols (little letters) and meta-symbols (big letters) enclosed in angle brackets. For example, np SUBJ is a single hyper-symbol, which contains within its angle brackets the proto-symbol 'np' and the meta-symbol 'SUBJ'. Each rule schema is called a hyper-rule, and takes the form of a context-free rule rewriting a single hyper-symbol on the left side as a string of hyper-symbols. For example, three hyper-rules are:

< s > :→ < np SUBJ >   < SUBJ vp >  
< np SUBJ > :→ < det terminal >   < noun SUBJ terminal >  
< SUBJ vp > :→ < verb SUBJ terminal >

Each has one hyper-symbol on the left side, and a string of hyper-symbols on the right side. In hyper-rules the rewrite symbol is  $:\rightarrow$ , so any rule with a single-colon arrow is a hyper-rule. Hyper-alternatives are separated by a vertical bar, just as with meta-alternatives. (There are no hyper-alternatives in the rules above.) We have already used up the distinction between upper-case non-terminals and lower-case terminals in the meta-grammar, and indeed both appear intermixed in the hyper-symbols. We need a new convention to express that distinction in hyper-grammars, so by convention all terminals in hyper-grammars will end with the proto-symbol "terminal". (In the hyper-grammar above, both the second and third rules expand to strings of terminals only.) Some additional mechanism is then required, such as a lexicon, to associate each terminal symbol with its representation, which is ordinary computational practice anyway. We will not be concerned here with the representation of terminals.

The three hyper-rules make use of the meta-symbol SUBJ which (as we saw before) derives three terminal strings in the meta-grammar: since any of these may be substituted (observing the Uniform Replacement Convention) in each hyper-rule, that makes the hyper-rules short for nine rules in total:

$\langle s \rangle \rightarrow \langle np \text{ minus-animate} \rangle \langle \text{minus-animate vp} \rangle$   
 $\langle s \rangle \rightarrow \langle np \text{ plus-animate minus-human} \rangle$   
 $\qquad \qquad \qquad \langle \text{plus-animate minus-human vp} \rangle$   
 $\langle s \rangle \rightarrow \langle np \text{ plus-animate plus-human} \rangle$   
 $\qquad \qquad \qquad \langle \text{plus-animate plus-human vp} \rangle$

$\langle np \text{ minus-animate} \rangle \rightarrow \langle \text{det terminal} \rangle \langle \text{noun minus-animate terminal} \rangle$   
 $\langle np \text{ plus-animate minus-human} \rangle \rightarrow \langle \text{det terminal} \rangle$   
 $\qquad \qquad \qquad \langle \text{noun plus-animate minus-human terminal} \rangle$   
 $\langle np \text{ plus-animate plus-human} \rangle \rightarrow \langle \text{det terminal} \rangle$   
 $\qquad \qquad \qquad \langle \text{noun plus-animate plus-human terminal} \rangle$

$\langle \text{minus-animate vp} \rangle \rightarrow \langle \text{verb minus-animate terminal} \rangle$   
 $\langle \text{plus-animate minus-human vp} \rangle \rightarrow \langle \text{verb plus-animate minus-human terminal} \rangle$   
 $\langle \text{plus-animate plus-human vp} \rangle \rightarrow \langle \text{verb plus-animate plus-human terminal} \rangle$

Rules such as these, generated by the hyper-rule schemata by replacing meta-symbols, we will call production-rules. Such production-rules have the plain arrow as the rewrite symbol (their mark as regular context-free grammar rules), and they rewrite a single production-symbol on the left side as a string of production symbols. The production-symbols are simply the concatenation of the proto-symbols within a pair of angle brackets after substitution has taken place; the brackets are ordinarily retained for ease in reading. Observe carefully that the separation between distinct proto-symbols is no longer present after replacement of meta-symbols has taken place; a production-symbol such as "s" and a production symbol such as "npplus-animateminus-human" are equally thought of as just single, unanalyzeable symbols -- exactly as they would be in an ordinary context-free grammar.

For reference, we now insert a summary of this section:

A van Wijngaarden grammar (vW-grammar) consists of two components: (1) a meta-grammar, and (2) a hyper-grammar.

The purpose of the meta-grammar is to define a structured vocabulary of symbols. It takes the form of a context-free grammar in which the non-terminals are meta-symbols (written in UPPER CASE LETTERS), and the terminals are proto-symbols (written in lower case letters). Each meta-rule consists of re-writing a single meta-symbol as a string of meta-symbols and proto-symbols. The re-write symbol used in meta-rules is :: →. The symbol used in meta-rules to separate meta-alternatives is |.

Example meta-grammar component of a vW-grammar:

SUBJ :: → ANIMATE

ANIMATE :: → minus-animate | plus-animate HUMAN

HUMAN :: → minus-human | plus-human

The purpose of the hyper-grammar is to serve as a set of rule schemata for the rules defining the language of the vW-grammar. It takes the form of a context-free grammar in which the non-terminals are hyper-symbols (strings of proto-symbols and meta-symbols enclosed in angle brackets < >) and the terminals are hyper-symbols ending with the proto-symbol 'terminal'. Each hyper-rule consists of re-writing a single hyper-symbol as a string of hyper-symbols. The re-write symbol used in hyper-rules is : →. The symbol used in hyper-rules to separate hyper-alternatives is |.

Example hyper-grammar component of a vW-grammar:



$\langle s \rangle \rightarrow \langle np \text{ SUBJ} \rangle \langle \text{SUBJ } vp \rangle$   
 $\langle np \text{ SUBJ} \rangle \rightarrow \langle \text{det terminal} \rangle \langle \text{noun SUBJ terminal} \rangle$   
 $\langle \text{SUBJ } vp \rangle \rightarrow \langle \text{verb SUBJ terminal} \rangle$

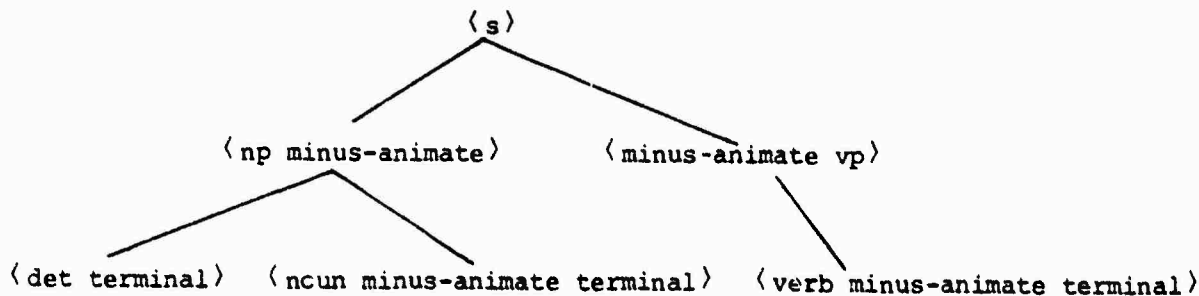
In a vW-grammar, the hyper-rules specify a grammar (the production-grammar) which is obtained by replacing in the hyper-rules all the meta-symbols with strings of proto-symbols which can be derived in the meta-grammar by treating the meta-symbol as the start-symbol. (Thus, the meta-grammar may actually be made up of several sets of meta-rules which do not interact.) This must be done in accordance with the Uniform Replacement Convention (URC), which says that all instances of the same meta-symbol in a single hyper-rule must be replaced with the same string of proto-symbols.

The purpose of the production-grammar obtained in this way is to specify the language of the vW-grammar. It takes the form of a context-free grammar (but possibly with an infinite number of rules) in which the non-terminals are concatenations of proto-symbols (strings of lower case letters), and the terminals are concatenations of proto-symbols ending in 'terminal'. (Some further mechanism, such as a lexicon, is then used to give the representation of each terminal symbol.) Each production-rule consists of re-writing a single production-symbol as a string of production-symbols. The re-write symbol used in production rules is  $\rightarrow$ . The symbol used in production-rules to separate production-alternatives is  $\perp$ . Angle brackets may optionally be retained around production-symbols to aid readability.

Example production-grammar (specified by the vW-grammar above):

$\langle s \rangle \rightarrow \langle np \text{ minus-animate} \rangle \langle \text{minus-animate } vp \rangle$   
 $\langle s \rangle \rightarrow \langle np \text{ plus-animate minus-human} \rangle$   
 $\quad \langle \text{plus-animate minus-human } vp \rangle$   
 $\langle s \rangle \rightarrow \langle np \text{ plus-animate plus-human} \rangle$   
 $\quad \langle \text{plus-animate plus human } vp \rangle$   
 $\langle np \text{ minus-animate} \rangle \rightarrow \langle \text{det terminal} \rangle \langle \text{noun minus-animate terminal} \rangle$   
 $\langle np \text{ plus-animate minus-human} \rangle \rightarrow \langle \text{det terminal} \rangle$   
 $\quad \langle \text{noun plus-animate minus-human terminal} \rangle$   
 $\langle np \text{ plus-animate plus-human} \rangle \rightarrow \langle \text{det terminal} \rangle$   
 $\quad \langle \text{noun plus-animate plus-human terminal} \rangle$   
 $\langle \text{minus-animate } vp \rangle \rightarrow \langle \text{verb minus-animate terminal} \rangle$   
 $\langle \text{plus-animate minus-human } vp \rangle \rightarrow \langle \text{verb plus-animate minus-human terminal} \rangle$   
 $\langle \text{plus-animate plus-human } vp \rangle \rightarrow \langle \text{verb plus-animate plus-human terminal} \rangle$

The language generated by this production-grammar (and thus by this vW grammar) consists of three strings of terminals, all with derivations essentially alike except for agreement of selectional restrictions:



(Important note. The terminology and notation introduced in this section (and used in all succeeding sections) were devised especially for this presentation, and do not correspond to those in the Algol 68 Report or Revised Report (van Wijngaarden et al. 1969, 1975); the present notation more resembles that used in various other, independent, studies (Baker 1972, Deussen 1975, Greibach 1974). But since the Algol 68 Report is the only sizeable example of van Wijngaarden grammar, it is nice to be able to read it; the recent introduction to van Wijngaarden grammars by Cleaveland and Uzgalis (1977) is an excellent practice field for the Algol 68 Report, and the present notation was chosen with an eye to making the transition as easy as possible.

(The Algol 68 Report, for example, uses the word "notion" in most of the places where "symbol" is used here; so a grammar consists of meta-notions, hyper-notions, proto-notions, and so forth. This use of "notion" is counter-intuitive. The notation of the Algol 68 Report differs in being less redundant; for instance, the sample van Wijngaarden grammar would be written:

(meta-rules:)

SUBJ:: ANIMATE.

ANIMATE:: minus-animate; plus-animate HUMAN.

HUMAN:: minus-human; plus-human.

(hyper-rules:)

s: np SUBJ, SUBJ np.

np SUBJ: det terminal, noun SUBJ terminal.

SUBJ vp: verb SUBJ terminal.

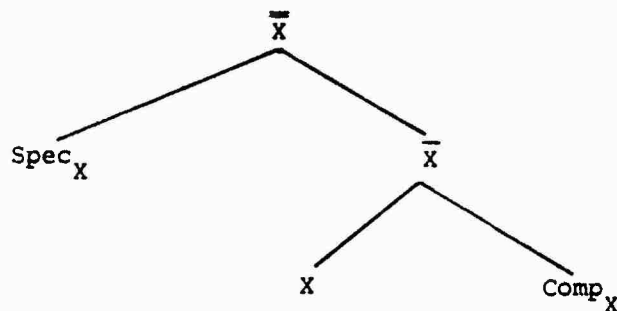
(No arrows, final periods, alternates separated by semicolons, hyper-symbols set apart by commas.)

(This notation is admirably suited to typewriters, and it is a pity that it is so hard to read (much harder, of course, in larger and more complicated grammars). But ten years of bitter experience have made it clear that for some reason people do not immediately find the Algol 68 Report notation helpful.)

### 3.3 Chomsky's Convention as a van Wijngaarden Grammar

As another example of the notation and the motivation for grammars with structured vocabulary, we might consider the " $\bar{X}$  Convention" as introduced by Chomsky (1970). (This presentation is now completely outdated, but it will serve as an example here since it is likely to be better known than more recent work in  $\bar{X}$  syntax.) The  $\bar{X}$  convention is introduced as part of a general reformulation of the base component so that instead of unanalyzeable non-terminal symbols, each non-terminal node will be characterized by a complex symbol. (This is in itself a way of introducing a structured vocabulary into a context-free grammar, a point to which we will return eventually.) Jackendoff (1974) has summarized the  $\bar{X}$  convention in this way: "The general nature of the claims made by the  $\bar{X}$  convention is now clear. The structural schema (28) (below), in which  $N$  represents any lexical category, is claimed to constitute a linguistically significant generalization of the structures associated with the major categories."

(28)



That is, we expect there to exist rules whose structural descriptions refer to a range of structures including more than one value of X."

The rules proposed to be included in the base component by Chomsky will employ "a variable standing for the lexical categories N, A, V" which is called X. "Then the base rules introducing N, A, and V will be replaced by a schema." Eventually Chomsky arrives at rules which are summarized by Jackendoff as:

$$\bar{X} \rightarrow [\text{Spec}, \bar{X}] - \bar{X}$$

$$\bar{X} \rightarrow X - \text{comp}$$

where 'comp' is an abbreviation for some sequence of nodes, but is not itself a constituent. Sample comps are "NP, S, NP S, NP PrepP, PrepP PrepP, etc." (Chomsky 1970), and in the  $\bar{X}$  schema above the "full range of structures that serve as complements" should appear."

As presented by Chomsky, this entails a complete redefinition of the base component in ways which are not made fully explicit. We can, however, identify at least four different devices at work: (1) use of X as a cover term for N, A, or V in rule schemata; (2) use of X,  $\bar{X}$ ,  $\bar{X}$  to indicate systematic relatedness of categories; (3) use of  $[\text{Spec}, \bar{X}]$  as a complex symbol which is analyzed differently depending on the value of X --  $[\text{Spec}, \bar{N}]$  as the determiner,  $[\text{Spec}, \bar{V}]$  as the auxiliary, and  $[\text{Spec}, \bar{A}]$  perhaps as the system of qualifying elements associated with adjective phrases. "Analyzed" has here a technical meaning, namely that there are rules

[ Spec,  $\bar{N}$  ]  $\rightarrow$  Det

[ Spec,  $\bar{V}$  ]  $\rightarrow$  Aux

and so forth; (4) use of 'comp' (Chomsky uses an ellipsis ... instead) to indicate expansions into various node sequences, although 'comp' is not a constituent.

Most of these devices can be captured adequately in a van Wijngaarden grammar, and so it will be an instructive exercise to recast the two rule schemata of Chomsky, and all the accompanying understandings about how they are to be interpreted, into van Wijngaarden form.

(1) It is easy to find a way to let X represent the category symbols N, A, or V in a rule, since that is a chief use of meta-variables. All we need is a meta-rule defining X:

$X ::= n \mid a \mid v$

(2) It is also easy to capture the relatedness of the categories X,  $\bar{X}$ ,  $\overline{\bar{X}}$  by defining a meta-symbol for bar and double bar (call them BAR and DOUBLE), and then using hyper-symbols in the hyper-rules which are composed of the meta-symbol for X and the meta-symbol for one of the bars -- such as  $\langle X \rangle$ ,  $\langle X \text{ BAR} \rangle$ ,  $\langle X \text{ DOUBLE} \rangle$  -- so that each hyper-symbol contains a use of the same meta-variable, X. The separate question of how the categories so related are rewritten in terms of one another is correctly captured in the Chomsky schemata, and so it transfers naturally to the hyper-rules which are also schemata. The first hyper-rule, then, will be:

$\langle X \text{ DOUBLE} \rangle \Rightarrow \langle \text{spec } X \text{ BAR} \rangle \langle X \text{ BAR} \rangle$

(3) Having introduced such production-symbols as  $\langle \text{spec } n \text{ bar} \rangle$  or  $\langle \text{spec } v \text{ bar} \rangle$  under the covering hyper-symbol of  $\langle \text{spec } X \text{ BAR} \rangle$ , it is perfectly easy to distinguish among them and to expand them differently through additional hyper-rules:

<spec n bar> :→ <det>  
 <spec v bar> :→ <aux> <OPTIONALADV>  
 <spec a bar> :→ <qual>

(4) The remaining question of how to write 'comp' and yet avoid having it be a constituent is the most difficult. Chomsky used the wild-card symbol of ellipsis because rule schemata do not provide a way to summarize rules with unboundedly different numbers of symbols on the right-hand side, and neither do van Wijngaarden hyper-rules.

It is possible, however, that the content of 'comp' should not have a recursive definition with the structure hidden, but rather should be defined in terms of a finite number of "slots", each of which can be "filled" by various nodes or in some cases by nothing; this style of description has often been used by linguists. For exposition, we will assume that a 'comp' is an optional NP or PP, followed by an NP or S. This will provide an exhibition of one way in which optional elements can be introduced into van Wijngaarden grammars.

The rule which we wish to write would be represented in a common notation for abbreviating context-free rules as:

$$\bar{x} \rightarrow x \quad \left( \left\{ \begin{array}{l} \text{NP} \\ \text{PP} \end{array} \right\} \right) \quad \left\{ \begin{array}{l} \text{NP} \\ \text{S} \end{array} \right\}$$

This abbreviates six rules, namely those with right-hand sides

X	NP	NP
X	PP	NP
X		NP
X	NP	S
X	PP	S
X		S

We will now explain an important additional convention which is needed in interpreting van Wijngaarden grammars for optional elements

and also for other purposes, and then we will apply that information to writing the rule above.

In a van Wijngaarden grammar, we can make use of optional elements by introducing a new meta-symbol and meta-rule:

EMPTY  $:: \rightarrow \lambda$

(EMPTY has as its expansion the null string, represented by the lower-case lambda [ $\lambda$ ] rather than the usual upper-case [ $\Lambda$ ] to preserve conventions.)

We can then write meta-rules utilizing EMPTY as one of the alternatives, for example:

OPTIONALADV  $:: \rightarrow \text{adv} \mid \text{EMPTY}$

and hyper-rules to use such meta-symbols, such as the one from the last sub-section:

$\langle \text{spec v bar} \rangle :: \rightarrow \langle \text{aux} \rangle \langle \text{OPTIONALADV} \rangle$

Such a hyper-rule will give rise to sub-trees in structural descriptions such as



(We will customarily show EMPTY in such trees rather than its terminal empty string in the meta-grammar, for ease of reading.)

The important convention is that we interpret a structural description such as the second one above to be exactly the same as another tree in which the EMPTY and all nodes which dominate EMPTY exclusively have been pruned away; here, we identify the second tree above with the pruned tree

< spec v bar >

|

< aux >

We will have more important uses for this convention in following sections, since it will permit the enhancement of using hyper-symbols as "predicates", which increases the naturalness of van Wijngaarden grammars.

Using this device for optionality, then, we can now write rules for the COMP elements:

OPTIONALCOMPA ::→ np | pp | EMPTY

COMPB ::→ np | s

EMPTY ::→ λ

< X BAR > ::→ < X > < OPTIONALCOMPA > < COMPB >

Here the COMP meta-symbols can be considered as the names of "slots" and their terminal meta-expansions as "fillers" for the slots; the slot names play no role in trees generated by the hyper-rule above, because they are replaced with actual node names in production-rules. (This interpretation is capable of further historical and practical development, since linguists have argued for years about how to incorporate such notions into context-free grammars.)

We have now completed, piecemeal, the construction of van Wijngaarden rules to describe Chomsky's two schemata and quite a number of attendant informal understandings. The result looks like this.

#### van Wijngaarden grammar for the X convention

Meta-rules:

m1. X ::→ n | a | v

m2. BAR ::→ bar

m3. DOUBLE ::→ BAR BAR



- m4. OPTIONALCOMPA ::→ np | pp | EMPTY  
 m5. COMPB ::→ s | np  
 m6. OPTIONALADV ::→ adv | EMPTY  
 m7. EMPTY ::→ λ

Hyper-rules:

- h1. < X DOUBLE > :→ < spec X BAR > < X BAR >  
 h2. < X BAR > :→ < X > < OPTIONALCOMPA > < COMPB >  
 h3. < spec n bar > :→ < det >  
 h4. < spec v bar > :→ < aux > < OPTIONALADV >  
 h5. < spec a bar > :→ < qual >

The production rules which can be derived from the hyper-rules (via the Uniform Replacement Convention) are:

- (from h1:) n bar bar → spec n bar n bar  
 a bar bar → spec a bar a bar  
 v bar bar → spec v bar v bar
- (from h2:) n bar → n np np  
 n bar → n pp np  
 n bar → n np s  
 n bar → n pp s  
 n bar → n s
- a bar → a np np  
 a bar → a pp np  
 a bar → a np  
 a bar → a np s  
 a bar → a pp s  
 a bar → a s
- v bar → v np np  
 v bar → v pp np  
 v bar → v np  
 v bar → v np s  
 v bar → v pp s  
 v bar → v s

(from h3:) spec n bar → det  
 (from h4:) spec v bar → aux λ  
                   spec v bar → aux adv  
 (from h5:) spec a bar → qual

It is indeed clear that several linguistically significant generalizations are not asserted in these production-rules, but they are asserted in the rules of the van Wijngaarden grammar (whether or not these generalizations are correct is not the point here; for further information about the X convention see (Malitsky 1975)). It is because these linguistically significant generalizations are captured, that the van Wijngaarden rules are much easier for a human being to understand also.

#### 3.4 van Wijngaarden Grammars for Non-Context-Free Languages

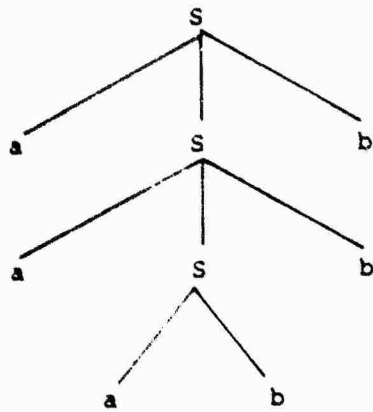
With the motivation provided by the foregoing linguistic examples, we will now introduce some brief and schematic examples of van Wijngaarden grammars for artificial languages so as to give a better idea of their possibilities. These examples are modeled after some in the literature (de Chastellier and Colmerauer 1969, Cleaveland and Uzgalis 1977), and illustrate techniques which are common in van Wijngaarden grammar writing but which are not readily apparent to one accustomed to single-level grammars.

Our first example will be a grammar for the set of strings  $\{ a^n b^n \mid n \geq 1 \}$  -- that is, the language of all strings of a's followed by an equal number of b's, or  $ab, aabb, aaabbb, \dots$

The usual context-free grammar for this language is given as

$$S \rightarrow a S b \mid a b$$

which produces derivations such as



There is, of course, no reason not to write such a grammar directly as a van Wijngaarden grammar with a null meta-grammar component; hence,

$$\langle s \rangle ::= \langle a \text{ terminal} \rangle \langle s \rangle \langle b \text{ terminal} \rangle \mid \langle a \text{ terminal} \rangle \langle b \text{ terminal} \rangle$$

In either notation, the number of a's and b's is controlled by how many times the first alternative of the rule is used, and the equality constraint is enforced because each rule application introduces exactly one terminal a and also exactly one terminal b.

But there is another way of getting the same language from a van Wijngaarden grammar, which is less straightforward but which generalizes as the grammar above does not. An alternative van Wijngaarden grammar for  $\{a^n b^n \mid n \geq 1\}$  is

Meta-rules:

$$m1. N ::= n \mid N n$$

$$m2. AB ::= a \mid b$$

Hyper-rules:

$$h1. \langle s \rangle ::= \langle N a \rangle \langle N b \rangle$$

$$h2. \langle n N AB \rangle ::= \langle AB \text{ terminal} \rangle \langle N AB \rangle$$

$$h3. \langle n AB \rangle ::= \langle AB \text{ terminal} \rangle$$

This will take a bit of study, but the idea is basically simple and is useful in many van Wijngaarden grammars. Notice first that meta-rule 1 (m1 above) specifies recursively any number of n's -- meta-symbol N derives as terminal strings of proto-symbols n, nn, nnn, nnnn, nnnnn, and so on. Thus, meta-symbol N used as a start symbol in the meta-grammar yields an infinite set of values; and that in turn means that when N is used in a hyper-rule such as h1 above, that the set of production-rules which can be made from the hyper-rule schema is also infinite, one rule for every possible value of N. The first few production rules manufactured from hyper-rule h1 above would be:

$$\begin{aligned} \langle s \rangle &\rightarrow \langle na \rangle \langle nb \rangle \\ \langle s \rangle &\rightarrow \langle nna \rangle \langle nnb \rangle \\ \langle s \rangle &\rightarrow \langle nnaa \rangle \langle nnnb \rangle \\ \langle s \rangle &\rightarrow \langle nnaaa \rangle \langle nnnnb \rangle \\ \langle \quad \rangle &\rightarrow \langle nnaaaa \rangle \langle nnnnab \rangle \\ &\vdots \end{aligned}$$

and so on. This immediately changes the theory of the rules that we are accustomed to, because now the language of the van Wijngaarden grammar is specified by an infinite number of production-rules (whereas it is a basic requirement of ordinary context-free grammars that the set of rules should be finite).

The second hyper-rule also used the meta-symbol N in an essential way:

$$h2. \langle n N AB \rangle \rightarrow \langle AB \text{ terminal} \rangle \langle N AB \rangle$$

This again abbreviates an infinite number of production rules, which provide (in general) that a symbol composed of a certain number of n's plus an a or b, can be rewritten as an a-or-b-terminal followed by a symbol containing one fewer N's than the left side of the rule. (The meta-symbol AB in the rule is just a cover term for a or b.) It is the Uniform Replacement Convention which makes this true; the URC assures that the same value for N (and for AB) is inserted into both

sides of the rule; since there is an extra explicit 'n' on the left, the production-rules generated by this hyper-rule will be of the form:

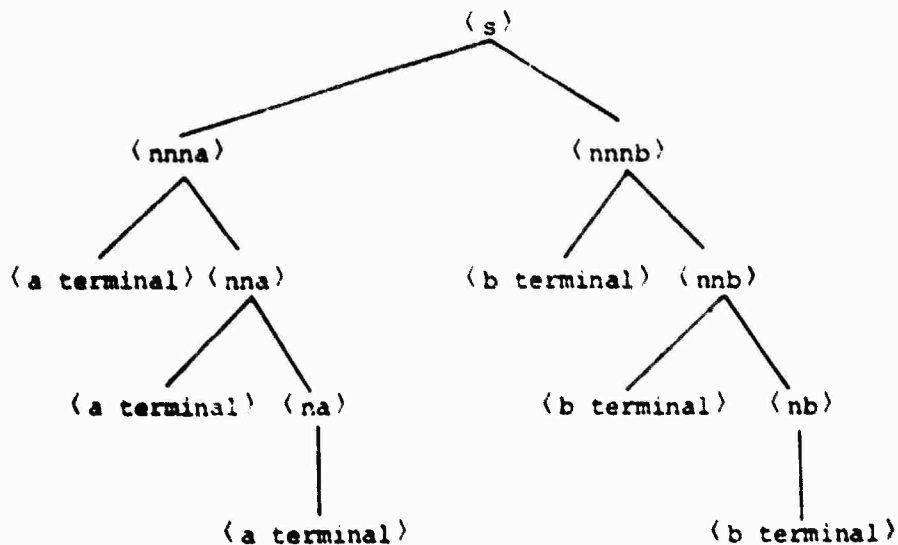
$\langle nna \rangle \rightarrow \langle a \text{ terminal} \rangle \langle na \rangle$   
 $\langle nnb \rangle \rightarrow \langle b \text{ terminal} \rangle \langle nb \rangle$   
 $\langle nnaa \rangle \rightarrow \langle a \text{ terminal} \rangle \langle nna \rangle$   
 $\langle nnnb \rangle \rightarrow \langle b \text{ terminal} \rangle \langle nnb \rangle$   
 $\langle nnaaa \rangle \rightarrow \langle a \text{ terminal} \rangle \langle nnaa \rangle$   
 $\langle nnnnb \rangle \rightarrow \langle b \text{ terminal} \rangle \langle nnnb \rangle$   
 $\vdots$

and so on for all the values of N. Each such rule in effect casts off a terminal and leaves a new non-terminal which can be re-written by a previous rule. The final hyper-rule, h3,

h3.  $\langle n AB \rangle \rightarrow \langle AB \text{ terminal} \rangle$

simply provides for handling the final case, the 'shortest' non-terminals produced by rules of hyper-rule h2.

So a sample derivation in this van Wijngaarden grammar will look like



In this grammar the number of a's and b's is determined in an entirely different way from the context-free grammar: here the number of a's and b's depends not on how many times a recursive rule is applied to rewrite  $S \rightarrow a S b$ , but rather on which of the infinite number of expansions of  $\langle s \rangle$  is initially chosen and applied once. The equality constraint is likewise enforced in an entirely different way: here the equality depends not on the fact that one a and one b are added on each recursive expansion, but rather is enforced by the Uniform Replacement Convention, which assures agreement between the two non-terminals introduced by the first rule application ("agreement" here meaning that they initiate parallel chains of rules in the grammar to generate the same number of terminals).

There would be no point in going through all of this for the  $\{a^n b^n \mid n \geq 1\}$  language, since that language has a simpler context-free grammar. But the approach of the second van Wijngaarden grammar generalizes in a way that the approach of the first one does not. If we now wish to have a language  $\{a^n b^n c^n \mid n \geq 1\}$  -- that is, any number of a's, followed by the same number of b's, and again an equal number of c's, or abc, aabbcc, aaabbbccc, aaaabbbbcccc, etc. -- the simpler scheme breaks down. This new language is one which is well-known to be not context-free, meaning that it is not generated by any context-free grammar. (It is not hard to see why this is so: there are only two sides to a center-embedded symbol, so a context-free grammar can coordinate only two strings at a time, and those must be mirror-images of each other.)

But there is a van Wijngaarden grammar for the language  $\{a^n b^n c^n \mid n \geq 1\}$  and it is only trivially different from the grammar for the preceding one. The only difference we need to introduce is to make a new meta-symbol ABC to be a cover symbol for a, or b, or c, and we also need to introduce the three items in the first hyper-rule instead of only two.

The resulting grammar is:

Meta-rules:

$N ::= n \mid N n$

$ABC ::= a \mid b \mid c$

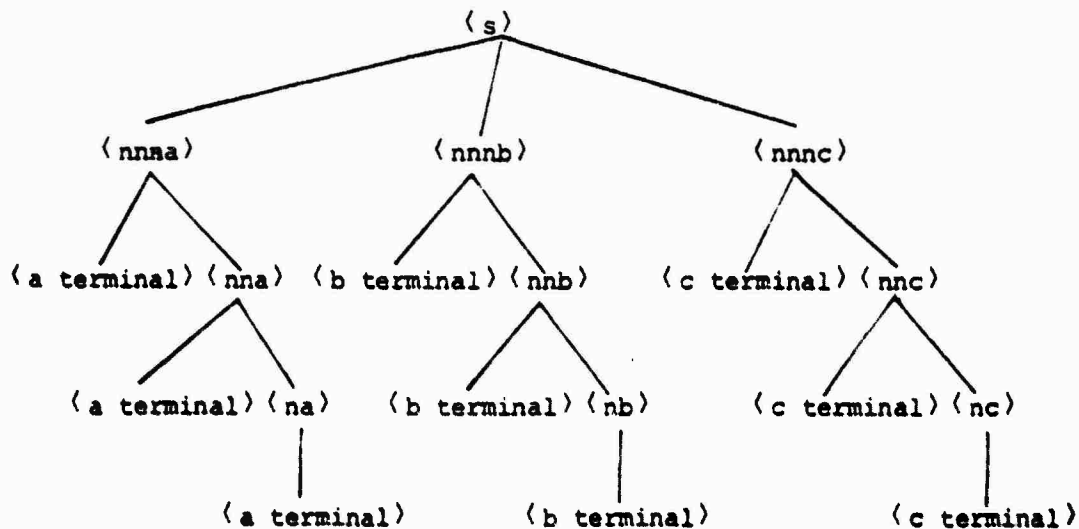
Hyper-rules:

$\langle s \rangle ::= \langle N a \rangle \mid \langle N b \rangle \mid \langle N c \rangle$

$\langle n N ABC \rangle ::= \langle ABC \text{ terminal} \rangle \mid \langle N ABC \rangle$

$\langle n ABC \rangle ::= \langle ABC \text{ terminal} \rangle$

with derivations such as:



While it is true that counting of this sort is not precisely a phenomenon of natural language, this example demonstrates that other matters of agreement which are not naturally handled by context-free rules may nevertheless be handled simply by a van Wijngaarden grammar.

### 3.5 van Wijngaarden Grammars with Hyper-symbols as Predicates

We will now consider a device which can be seen as merely a matter of style in writing van Wijngaarden grammars, but which opens up surprising

possibilities of gaining the effect of "tests" or conditions on properties of symbols while remaining wholly within the original syntactic framework. This possibility was not used in the original Algol 68 Report, but was incorporated for the first time in the Revised Report; this suggests, correctly, that the idea was not entirely obvious, even to A. van Wijngaarden himself. But it is very simple, and although it is in some sense a trick, it is a satisfyingly elegant trick. The basic notion involved is to introduce extra hyper-symbols into hyper-rules, and to arrange that the other hyper-rules should either derive EMPTY from the additional symbols, or else should fail to derive any terminal string, thus leading to a blocked derivation. But this explanation give no idea of how the idea is used, and we shall now develop that slowly.

Let us begin with an example of a van Wijngaarden grammar to generate all strings of double letters -- the (finite) language 'aa', 'bb', 'cc', ..., 'yy', 'zz'. A perfectly adequate van Wijngaarden grammar would be:

Meta-rule:

$$\text{ALPHA} ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | \\ q | r | s | t | u | v | w | x | y | z$$

Hyper-rule:

$$\langle s \rangle ::= \langle \text{ALPHA terminal} \rangle \langle \text{ALPHA terminal} \rangle$$

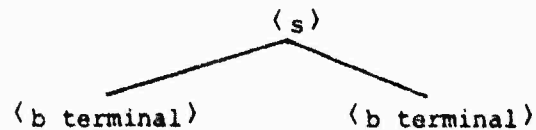
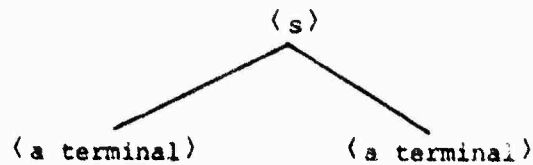
This grammar is correct because the Uniform Replacement Convention assures that the hyper-rule represents exactly 26 production rules such as

$$\langle s \rangle \rightarrow \langle a \text{ terminal} \rangle \langle a \text{ terminal} \rangle$$

$$\langle s \rangle \rightarrow \langle b \text{ terminal} \rangle \langle b \text{ terminal} \rangle$$

and so on. Only strings where both letters are the same are generated by the grammar, because only rules with both occurrences of ALPHA replaced by the same letter are available. We will have derivations such as





That grammar, as we said, is perfectly satisfactory, but now consider this longer approach to defining the same language of double letters:

**Meta-rules:**

- m1.  $\text{ALPHA} ::= a | b | c | \dots | x | y | z$
- m2.  $\text{ALPHA1} ::= \text{ALPHA}$
- m3.  $\text{ALPHA2} ::= \text{ALPHA}$
- m4.  $\text{EMPTY} ::= \lambda$

**Hyper-rules:**

- h1.  $\langle s \rangle ::= \langle \text{ALPHA1 terminal} \rangle \langle \text{ALPHA2 terminal} \rangle \langle \text{ALPHA1 ALPHA2} \rangle$
- h2.  $\langle \text{ALPHA ALPHA} \rangle ::= \langle \text{EMPTY} \rangle$

This grammar relies crucially on the way the Uniform Replacement Convention is understood to work. The URC says that in any one rule, all occurrences of the same meta-symbol must be replaced by identical terminal-strings generated in the meta-rules by the meta-symbol; but different meta-symbols may be replaced with different terminal strings. In particular, in hyper-rule 1 above ALPHA1 and ALPHA2 are different meta-symbols; they may be replaced independently with the same values

or with different values, but repeated instances of ALPHA1 or repeated instances of ALPHA2 must be replaced consistently with the same string in every repeated instance. (In the meta-rules above, obviously ALPHA1 and ALPHA2 have been introduced -- both directly deriving ALPHA -- for this very purpose, to have "synonyms" for ALPHA which are different to the URC. In future grammars, we will assume without explicit mention that all meta-symbols ending in single digits like these are introduced as synonyms for meta-symbols without digits, as in meta-rules 2 and 3 above.)

By the URC, then, from the first hyper-rule we will get production-rules like the following:

```

(a) → (a terminal) (a terminal) (a a)
(a) → (a terminal) (b terminal) (a b)
(a) → (a terminal) (c terminal) (a c)
.
.
.
(a) → (b terminal) (a terminal) (b a)
(a) → (b terminal) (b terminal) (b b)
(a) → (b terminal) (c terminal) (b c)
.
.
.
(a) → (c terminal) (a terminal) (c a)
(a) → (c terminal) (b terminal) (c b)
(a) → (c terminal) (c terminal) (c c)
.
.
.

```

The first column of terminals and the first letter in the pair at the end both come from replacing ALPHA1; the second column of terminals

and the second letter in the pairs at the end both come from replacing ALPHA2; so replacement values for the same meta-symbol always agree, but replacement values for the different meta-symbols are chosen independently.

We have dramatically increased the number of production-rules used to define the language. In the previous grammar of double letters, we had 26 production-rules in all; now in this grammar we have 26 times 26 production rules (676 rules) from the first hyper-rule alone, and all the pairs of letters that we do not want are being generated so far.

We correct for this, and discard all the pairs of letters that do not agree, with the second hyper-rule:

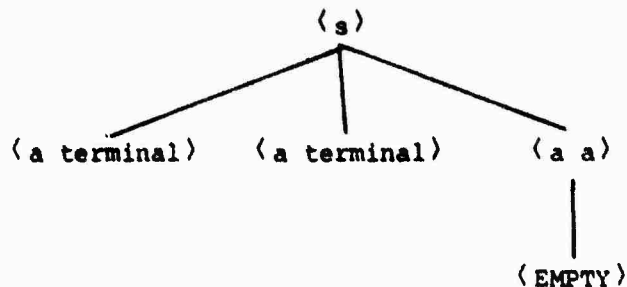
$\langle \text{ALPHA ALPHA} \rangle \rightarrow \langle \text{EMPTY} \rangle$

again by the URC the two instances of ALPHA must be replaced by the same string of proto-symbols, so this rule underlies just an additional 26 production rules of the sort:

$\langle \text{a a} \rangle \rightarrow \langle \text{EMPTY} \rangle$

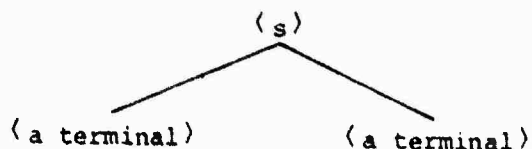
$\langle \text{b b} \rangle \rightarrow \langle \text{EMPTY} \rangle$

and so forth. Accordingly, the second hyper-rule will provide for re-writing the pairs of letters generated by the first hyper-rule, just in case they are the same letter. We will have derivations such as:



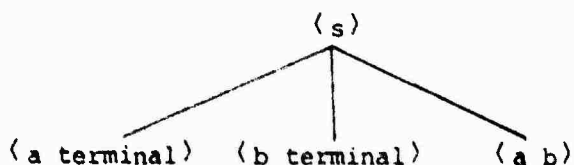
In just the cases we want, then, the strings in which the same terminal is repeated, we will get a tree like this one (EMPTY produces the

terminal null string  $\lambda$ , but by our agreement we have instead the meta-symbol EMPTY). Since the terminal string of the third non-terminal  $aa$  is the null string, we have the correct string of terminals; and in accordance with the convention explained earlier that all nodes in a tree which dominate only EMPTY are pruned (that the tree is identified with another tree which is the same except that those nodes are missing), we will treat the structural description above as equivalent to the structural description



and so both the string and the tree are what we want.

So much for how the strings of double letters are generated, which was the language to be defined. Now in very many cases the partial derivations produced by our grammar will not be like the ones we have inspected. Instead, they will be like the following:



This is a very different situation. The production-symbol  $\langle \text{a b} \rangle$  is not a terminal, obviously, because it does not end with the proto-symbol 'terminal'. It is a non-terminal, but the grammar provides no way to re-write that non-terminal symbol; there is no production-rule with the production-symbol  $\langle \text{a b} \rangle$  on the left side (because hyper-rule 2 only provides production-rules for rewriting such symbols when they are composed of the same character twice). So this last tree is not a structural description underlying any string in the language of the grammar. This attempted derivation just "blocks" since it cannot be completed into a string of terminals.

Of the production-rules, 650 rules introduce such pair-symbols which are not matched; only 26 introduce double letters. The 26 production-rules of the second hyper-rule then re-write the double-letter symbols as EMPTY. The 650 rules always introduce non-terminal symbols which cannot be rewritten, thus blocking a derivation in a "blind alley." Thus, this second grammar also defines just the language of the 26 pairs of double letters, the same language as the first grammar did.

Although in this example the second grammar is more cumbersome than the first, the general strategy is useful in more complicated grammars: it is often clearer to write a hyper-rule so that 't gives rise to unwanted production-rules, and then "restrict" those production rules by failing to provide additional production-rules to rewrite all the symbols introduced.

The name "predicate" is used to describe hyper-symbols such as  $\langle \text{ALPHA1 ALPHA2} \rangle$  which are inserted only to either (a) yield EMPTY and disappear, or (b) yield a "blind alley" and block. It is possible to be more suggestive by adding some extra proto-symbols to a predicate as window-dressing. For example, instead of  $\langle \text{ALPHA1 ALPHA2} \rangle$ , we might toss in two extra proto-symbols and make the hyper-symbol  $\langle$  where ALPHA1 is ALPHA2  $\rangle$ . This would let us rewrite the hyper-rules of the last grammar as

Hyper-rules:

h1.  $\langle s \rangle \rightarrow \langle \text{ALPHA1 terminal} \rangle \langle \text{ALPHA2 terminal} \rangle$

$\langle$  where ALPHA1 is ALPHA2  $\rangle$

h2.  $\langle$  where ALPHA is ALPHA  $\rangle \rightarrow \langle \text{EMPTY} \rangle$

but this does not change the method in the least, it only gives more suggestive hyper-symbols and the ability to define more than one predicate with the same meta-symbols. One might have

<where ALPHA1 is ALPHA2>

<where ALPHA1 is not ALPHA2>

<where FEATURES1 are contained in FEATURES2>

<where FEATURES1 nonconflicts with FEATURES2>

and so on, each of which would block in different circumstances.

It is required that all such predicate hyper-symbols be defined in strictly syntactic terms, and we have not really shown yet how this is done. (The example of <where ALPHA is ALPHA> is hardly representative, because it uses the URC to immediately take over all the work.) How to write the syntactic rules to make the predicates have their intended effect is rather specialized, and for the present it is sufficient to believe that a great deal is possible. As an aid in acquiring such a belief we will go just one step farther here, and define a simple but not trivial predicate hyper-symbol. Detailed examples are worked out at length in sections 5.1 and 5.4 below.

For this example, we will define a van Wijngaarden grammar to generate a language just the opposite of the last one -- this time, we will have the language of pairs of letters which are not the same. (This is again a finite language: ab, ac, ac, ..., ay, az, ba, bc, ....) We will use the same meta-rules again (this time omitting the rules for ALPHA1 and ALPHA2 in accordance with our convention that they are assumed to be synonyms for ALPHA), but we will need additional meta-rules for STRINGS and new hyper-rules.

Meta-rules:

m1. ALPHA ::→ a | b | c | ... | x | y | z

m2. EMPTY ::→ λ

m3. STRING ::→ ALPHA | STRING ALPHA

m4. OPTSTRING ::→ STRING | EMPTY

(A STRING is just one or more ALPHAs, and an OPTSTRING is zero or more ALPHAs.)

Hyper-rules:

h1.  $\langle s \rangle : \rightarrow \langle \text{ALPHA1 terminal} \rangle \langle \text{ALPHA2 terminal} \rangle$

$\langle \text{where ALPHA1 is not ALPHA2} \rangle$

h2.  $\langle \text{where ALPHA1 is not ALPHA2} \rangle : \rightarrow$

$\langle \text{where ALPHA1 precedes ALPHA2 in abcdefghijklmnopqrstuvwxyz} \rangle |$

$\langle \text{where ALPHA2 precedes ALPHA1 in abcdefghijklmnopqrstuvwxyz} \rangle$

h3.  $\langle \text{where ALPHA1 precedes ALPHA2 in OPTSTRING1 ALPHA1}$

$\text{OPTSTRING2 ALPHA2 OPTSTRING3} \rangle : \rightarrow \langle \text{EMPTY} \rangle$

The length of the hyper-symbols in the hyper-rules makes them fit the lines badly, but careful attention will sort them out. Rule h1 rewrites  $\langle s \rangle$  as a terminal, followed by another terminal, followed by a predicate. Rule h2 rewrites a predicate as either of two alternative more-primitive predicates. Rule h3 rewrites a single monstrously-long left-side symbol as EMPTY; note in this last rule h3 that everything but EMPTY is in a single pair of angle-brackets and is on the left side of the rule.

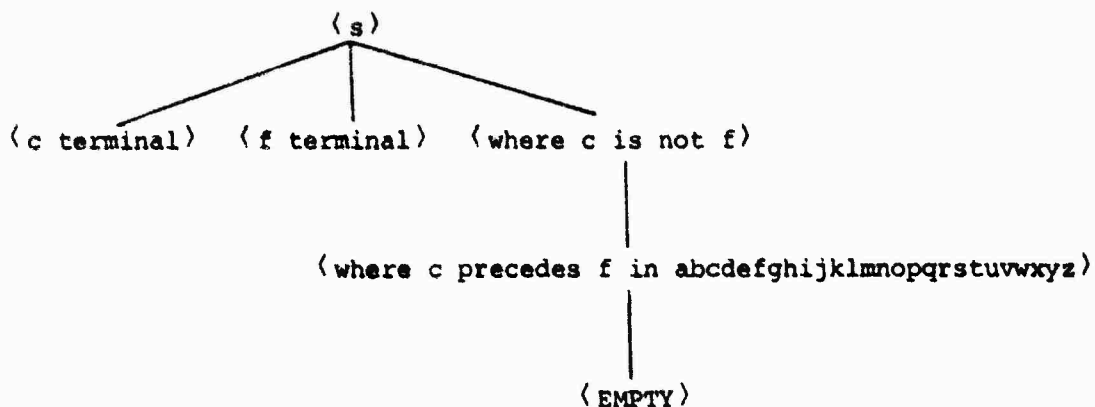
Clearly the basic idea of this grammar is the same as the last one, because in the first hyper-rule the only element which is changed is the final predicate; as before we generate two terminals and a restriction on them, but now the restriction is the reverse of what it was.

Rule h2 defines what it means for one ALPHA to "be not" another ALPHA, by saying that it means either that the first precedes the second in the alphabet or else the second precedes the first in the alphabet. (If neither of these is true, then they are the same letter.)

Rule h3 then defines what it means for one character to precede another in a string, by saying that if character 1 precedes character 2 in the string, then it will be possible to divide the string into five parts so that the first part is zero or more characters, the second part is character 1, the third part is zero or more characters, the fourth part is character 2, and the fifth part is zero or more characters.

But the summary just given of the hyper-rules is not quite adequate, because the effect described is achieved not by some kind of process of

trying a string and seeing if it can be divided, and so on, but purely in terms of symbols and rewrite rules. A sample derivation in this grammar could be



We know that there is a production-rule which produces EMPTY from that last symbol, because it comes from the last hyper-rule:

$\langle \text{where ALPHA1 precedes ALPHA2 in OPTSTRING1 ALPHA1 OPTSTRING2 ALPHA2 OPT-STRING3} \rangle$   
 where c precedes f in ab c de f ghijkl...

ALPHA1 and ALPHA2 are replaced by the same values at their repeated appearances, and OPTSTRING1, OPTSTRING2, and OPTSTRING3 are independently chosen as sequences of zero or more characters; therefore, this must be a production-rule validly produced from hyper-rule 3. Prolonged inspection will convince the interested reader that when the two characters are the same (ALPHA1 and ALPHA2 are the same terminal proto-symbols) and the string introduced by hyper-rule 2 is the alphabet, there is no way that any values can be chosen to make a production-symbol which is also on the left of any production-rule generated by hyper-rule 3, and hence when the two characters are the same the derivation will lead to a blind alley.

Now, once two letters can be distinguished, then two strings of letters can be distinguished (one letter at a time), and so any information which can be coded into strings (that is, any information) can be mani-



ulated. The details of this manipulation are fascinating, and to an enthusiast may be the most interesting facet of van Wijngaarden grammars; it is satisfying to be able to define the predicates with no additional machinery whatsoever. (van Wijngaarden 1974 contains a two-level grammar to simulate a Turing machine, using 9 meta-rules and 30 hyper-rules.)

As a practical matter, though, the exact definition of the predicates is of secondary importance. Once basic predicates are defined, then they can be used in one grammar after another without changes (something like a subroutine library). Also, the predicates would never be used in a computer program as they are in the grammars because they would be implemented instead with primitive tests for identity, non-identity, etc., outside the grammatical apparatus. Moreover, since anything can be coded as a predicate hyper-symbol, there is no actual restriction on expressive power in requiring that the definition be syntactic. It is important, however, that the URC restricts the predicate to appearing in the very hyper-rule where the items to be restricted are introduced; we gain 'locality of definition' when the restriction is on nodes introduced by a single rule, rather than on global tree configurations.

#### 4. Prior Linguistic Uses of Grammars with Structured Vocabulary

It is immediately suggestive that there have been several links between the development of the van Wijngaarden grammar formalism which we have been considering and grammars of natural languages. Adriaan van Wijngaarden himself, at the Mathematisch Centrum, applied its earliest computing machinery to a study of newspaper Dutch, saying that "we hope that this and similar information to be obtained in the future will help us to get better insight in the formal properties of our language" (van Berckel et al. 1965), and since then has shown an interest in natural language analysis using Algol 68 (van Wijngaarden 1970, Smith 1976).

C. H. A. Koster, an author of the Algol 68 Report and an influential advocate of the van Wijngaarden descriptive method, had first used a similar formalism in writing a context-free grammar of English in 1962 (see Koster 1965). The author of a technical note on van Wijngaarden grammars (Deussen 1975) makes reference to a doctoral dissertation of

the same period which consists of a large surface grammar of German utilizing essentially the van Wijngaarden grammar format (Schneider 1965, 1966). And one of the very earliest formalizations of the van Wijngaarden method was undertaken at the Université de Montreal for use in a French-English machine translation project (de Chastellier and Colmerauer 1969).

It seems likely that this repeated invention has been prompted by necessity when context-free grammars are used to describe natural languages. Evidence that this is so is forthcoming when one begins to consider other previous linguistic work in light of the model of grammars with structured vocabularies provided by van Wijngaarden grammars, and finds repeatedly the same essential use of structured vocabulary as a natural part of the descriptive techniques used.

Since it is not at once obvious exactly how to go about making best use of structured vocabulary in describing natural languages, it is reasonable to examine the independent approaches toward structured vocabulary which linguists have needed. The uses exemplified may be instructive, even when the understanding is as frustrated but as tantalizingly close as in this passage where Hockett (1966) describes a "componential alphabet" on two-levels:

A simple example is Potawatomi noun stems (N) which are either animate (An) or inanimate (In) in gender (Gn), and also either independent (Ind) or dependent (Dep) in dependency (Dp). ... One could provide for the whole situation in a single composite rule subsuming four elementary rules:

N      NAnInd, NAnDep, NInInd, NInDep

(Footnote 63:) I am not sure how the procedure developed here would fit into the rest of the grammar. I am not sure whether my notations such as 'NAnInd' are single characters or strings of characters; perhaps indeed one here needs to use a componential alphabet so that NAnInd (and the like) can be a single character but with components susceptible to separate manipulation."

Hockett's puzzlement in his footnote appears to have been shared by many others who felt that phrase-structure grammars missed some essential features of natural languages, and surprisingly often it turns out to be possible to interpret the missing element as provision for structured vocabulary.

#### 4.1 Pre-Chomskian Uses of Structured Vocabulary

Although there is little doubt that diligence could locate the true source of two-level grammars in Panini, we will for brevity in this initial essay begin with the American linguists who immediately preceded Chomsky. Their usage of structured vocabularies in describing natural languages may be most conveniently studied in Chomsky's own early manuscript The Logical Structure of Linguistic Theory (1955/1975) and in Paul Postal's provocative demonstration that a whole range of descriptive methods amount to very little more than phrase-structure grammars (Postal 1964b). These summaries are interesting enough, however, to suggest that additional first-hand consideration could be rewarding.

A lengthy development related to the ideas of Zellig Harris is provided by Chomsky (1955/1975), which seems to be of chief importance in understanding the ideas about structured vocabulary which played a role in Chomsky's own early theories. There the motivation for structured vocabulary arises from an elaborate scheme to establish "grammatical categories" so as to explain "degrees of grammaticalness" (i.e., why "of of the of" is less grammatical than "colorless green ideas sleep furiously" -- at this period, however, the second of these strings was considered to be grammatical, a status which it was later to lose).

Chomsky describes a process of hierarchical sorting, or clustering which establishes many extremely tiny grammatical categories containing only one or a few lexical items as members, then groups some of these into larger categories, some of the resulting categories into still larger categories, etc. Finally, there is an evaluation procedure (whose details are unknown, as is usual with evaluation procedures) by which to choose the level of grouping which is optimal for describing the language, and this level so chosen is called the "absolute analysis." Chomsky explains:

The absolute analysis embodies the major grammatical restrictions. Presumably these will be stated in terms of such classes as Noun, Verb, Preposition, etc. There will then be many further grammatical restrictions that have to do with limited and special contexts, and that will, presumably, be reflected in superior degrees of grammaticalness (i.e., smaller, lower-order categories). These further restrictions correspond in part to

what Harris has called selection. Thus selectional restrictions can be defined as those which refer to an account of grammaticalness which is more detailed and specific than that provided by the absolute analysis. Although Preposition may well turn out to be a class of the absolute analysis for English, there will be subclasses of prepositions that occur with different nouns and verbs, etc....

The difficulty is that categories of different sizes on different levels may simultaneously make different linguistically-significant generalizations:

There will, for instance, be various strings which we would like to say are noun phrases, even though they do not all appear grammatically (with first-order grammaticalness) as subjects of the same verb phrases, although each occurs grammatically with some verb phrase.

It is the desire to keep multiple levels of generalization which leads eventually to treating these category symbols as complex symbols (as in section 3.3 above), so that the complex similarities and differences can be preserved.

Some understanding akin to the one just outlined, that various grammatical categories were alike for some purposes but different for others, seems to have been general among American linguists of the period. This understanding was of principal use in two situations: (1) describing the occurrence of items whose environments were nearly identical, and (2) describing agreement phenomena, or perhaps more generally "discontinuous constituents". Although it is sometimes difficult to sharply separate these two, they do seem to be different uses.

A convenient example of the first kind of use is provided by Zellig Harris. Harris's formulas introduce several kinds of description reminiscent of two-level description. In one of these, symbols are given numbers and "each higher numbered symbol represents all lower numbered symbols, but not vice-versa" (Harris 1951). For instance, the formula  $N^2 - S = N^3$  also represents the additional formula  $N^1 - S = N^3$ ; but such generalization only occurs on the left sides of formulas (the right sides of rules). This system would be modeled by a van Wijngaarden grammar which made use of meta-symbols on only one side of hyper-rules. E.g.,

NTHREE ::→ nthree | NTWO

NTWO ::→ ntwo | NONE

NONE ::→ none

⟨nthree⟩ :→⟨NTWO⟩ ⟨s⟩

The numbers are introduced by Harris because an  $N^3$  can occur everywhere an  $N^2$  can occur, except in the rules turning an  $N^2$  plus something else into an  $N^3$ ; thus, by collapsing all these rules for items in the same environments, much duplication of rules is avoided.

A related example is given by Harris to show the utility of grouping morpheme classes into classes of "positions" in which morphemes occur (Harris 1951). If morphemes of class Q occur in positions which are the same as those of class R morphemes (the two being differentiated only in which adjuncts such as '-ly', '-al' they occur before), then it is possible to make a "positional category" N which includes Q and R (which are now to be written  $N_a$  and  $N_b$  respectively, to show that they are members of class N). The adjuncts -ly, -al are similarly classified into  $Na_a$  and  $Na_b$ . "...we have the equations  $N_a + Na_a = A$ ,  $N_b + Na_b = A$ , etc., all of which can be summarized in the position-class equation  $N + Na = A$ . It is understood that this equation, unlike our previous ones, holds not for every member of the classes involved but only for certain members (or sub-classes)." The ones for which it holds, of course, are the corresponding ones which appeared together among the sub-class equations, and which have been here suppressed.

The similarity of this understanding to the Uniform Replacement Convention suggests that a similar van Wijngaarden grammar could be written along the lines of

CLASS ::→ a | b

⟨a⟩ :→⟨n sub CLASS⟩ ⟨na sub CLASS⟩

which serves as the "position-class equation"  $N + Na = A$ . The sub-class equations summarized by this one can then be recorded in further

hyper-rules:

$\langle n \text{ sub } a \rangle : \rightarrow \langle q \rangle$   
 $\langle na \text{ sub } a \rangle : \rightarrow \langle \text{-ly terminal} \rangle$   
 $\langle n \text{ sub } b \rangle : \rightarrow \langle r \rangle$   
 $\langle na \text{ sub } b \rangle : \rightarrow \langle \text{-al terminal} \rangle$

As Harris remarks, "it is impossible to eliminate from our records the explicit sub-class equations"; their rule is here assumed by these further hyper-rules which preserve what each sub-class of N and Na represents. The "understanding" or convention of Harris's that these position-class equations are to be understood specially as holding only of corresponding sub-classes is not formally represented in his notation; in the van Wijngaarden grammar, of course, this is reconstructed by the meta-symbols in the hyper-rules and by the URC governing their replacement.

The motivation here again seems to have been a desire to collapse rules dealing with the identical or nearly-identical environments of the "Q" and "R" morphemes, but in this case there is also a need to define slots and to require that they be filled subject to agreement; so this usage shades over into our second type.

The second motivation for use of structured vocabulary in earlier linguistic work appears to be the desire to represent phenomena of "agreement" or "concord."

This was sometimes seen as of a piece with the problem of "discontinuous constituents." Harris, again (1951), employed what he called "long components" (by analogy to supra-segmental phenomena in phonology) to express agreement: "Similarly, ...a...a is a single morphemic segment, meaning female" in Latin filia bona 'good daughter'; such a female "long component" is treated as a component part of several complex symbols. This process is further extended and is really rather sophisticated, but it is never at all comfortable within an immediate constituent analysis.

Quite a number of people appear to have thought of utilizing "variables" in their formulas or representations, with something like the Uniform Replacement Convention to govern them and thus to enforce

agreement. Chomsky, in § 54 of (Chomsky 1955/1975) considers this very interpretation of 'long components' and proposes a notation for rules:

$$\begin{aligned}
 P_1 &\rightarrow P_2^k \sim P_3^k \\
 k &\rightarrow a \\
 k &\rightarrow b
 \end{aligned}$$

Here the "long component" is the symbol "k". Obviously the first rule is something like a van Wijngaarden hyper-rule, and the last two rules are something like meta-rules (and so we see that there is no separation of levels in this grammar). Observe that, just as in the Harris superscript numbers, the variables occur only on the right side of the rule.

Chomsky then goes on to propose something very like the URC to govern the interpretation of these rules, adding "suppose further that by convention all identical superscripts assume the same value in derivations .... The derivations would now work out exactly right, the algebra would be restricted, and the notations NP, VP, etc. would be retained with all essential generality." What this means is that because of the "componential" nature of category symbols like  $P_j^k$ , it is possible to let the P be "NP" and to recognize it as the same symbol regardless of what value of k qualifies it. Chomsky's rules and convention are clearly intended to be interpreted to be virtually identical to the van Wijngaarden rules

$$\begin{aligned}
 K &::\rightarrow a \mid b \\
 (P_1) &::\rightarrow (P_1 K) \quad (P_2 K)
 \end{aligned}$$

Chomsky decides, on the basis of having tried such a system to represent long components in his Morphophonemics of Modern Hebrew (1951) that it could well be useful for such things as number agreement, but it is not an appropriate device for imposing the vast complex of restrictions necessary to avoid "the rearming of Germany is at dinner" -- that is, apparently, for telling grammatical sense from grammatical nonsense. Chomsky then concludes with a note of much interest to those

studying van Wijngaarden grammars:

This is an important question, deserving of a much fuller treatment, but it will quickly lead into areas where the present formal apparatus may be inadequate. The difficult question of discontinuity is one such problem. Discontinuities are handled in the present treatment by construction of permutational mappings from P to W (phrase structures to words), but it may turn out that they must ultimately be incorporated into P (phrase structure) itself.

Chomsky himself appears to have believed that transformations were another tool for dealing with agreement, and so naturally he was more interested in following up the transformational approach to the question. Postal (1964b) extolls the incomparable value of the transformation to achieve agreement, and Koutsoudas (1966) exhibits many examples of the technique of generating an item of agreement in one constituent and then transformationally copying it into other constituents with which agreement was required. Postal (1964b) also credits Sydney Lamb (1962) in "stratificational grammar" and Elson and Pickett (1960) in "tagmemics" with introducing the ideas of variable-symbols and their uniform replacement to describe agreement phenomena; further study would probably uncover several more similar developments.

Our preliminary scan, then, indicates early use of structured vocabulary for two major purposes: (1) to indicate that two category symbols share many properties but are not subject to expansion by all the same rules -- that is, to encode "rule features"; (2) to indicate that two category symbols share many properties, but differ in their "contextual restrictions" of the sort usually thought of as "agreement". Either use can be naturally incorporated into a van Wijngaarden grammar.

#### 4.2 Structured Vocabulary in Transformations and 'Extended Phrase Structure Grammars'

Chomsky, throughout the period of Syntactic Structures (1957) and "A Transformational Approach to Syntax" (1962) continues to employ symbols which have the informal appearance of being structured, but he does not provide any formal method of representing their relatedness. Thus, he gives rules such as



$$NP \rightarrow \left\{ \begin{array}{l} NP_{sing} \\ NP_{pl} \end{array} \right\}$$

$$\left\{ \begin{array}{l} NP_{sing} \rightarrow T + N + \phi \\ NP_{pl} \rightarrow T + N + s \end{array} \right\}$$

and

$$Pred \rightarrow \left\{ \begin{array}{l} NP_{sing} \text{ in env. } NP_{sing} + Aux \left\{ \begin{array}{l} be \\ become \end{array} \right\} \text{ ————} \\ NP_{pl} \text{ in env. } NP_{pl} + Aux \left\{ \begin{array}{l} be \\ become \end{array} \right\} \text{ ————} \end{array} \right\}$$

$$V_t \rightarrow V_T \left\{ \begin{array}{l} Comp \\ Prt \end{array} \right\}$$

$$V_T \rightarrow \left\{ \begin{array}{l} V_{Ta'}, V_{Tb'}, \dots, V_{Tg'}, \text{ in env. } N_h \dots \text{ ———— } Comp \\ V_{Tx} \text{ in env. } \text{ ———— } Prt \end{array} \right\}$$

Here in the first rules the symbols  $NP_{sing}$  and  $NP_{pl}$  are being used for number agreement; but the two symbols, no matter how suggestively similar, are different and unrelated symbols in the grammar. It makes no difference that the brackets are drawn around alternatives in the first rule, or drawn around two entire rules; the symbols  $NP_{sing}$  and  $NP_{pl}$  remain as distinct as Aux and Prt in the vocabulary of the grammar. The rules dealing with verbs use context-sensitive format to impose rule-features as well as agreement.

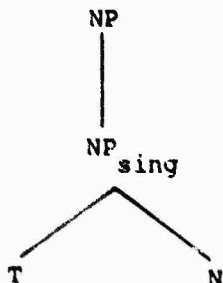
The reason why Chomsky gets along reasonably well without a method for relating various symbols in the vocabulary is that the only "relatedness"

of importance, in much of this work, is that a group of symbols should be treated the same way by transformations, and this kind of relatedness is reconstructed in the definition of a transformation rather than in the vocabulary of node labels.

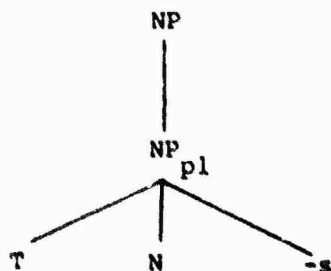
A transformation operates on terminals, but segments them into terms based on the complicated notion of a "proper analysis" which amounts to a string of non-overlapping sub-trees which together exhaustively dominate the terminal string. This means, among other things, that if the structural description of the passive transformation is

NP - Aux - VPP - NP  
 1    2    3    4

then its first term is as well satisfied by a subtree



as it is satisfied by the alternative subtree



because only the top node of the sub-tree is relevant to satisfying the structural description of the transformation.

This means that there is one way  $NP_{pl}$  and  $NP_{sing}$ , for example,

are related in the grammar, and that is that they can both be rewritings of NP. This is utilized by the transformational rules, since in the structural description of a transformational rule a node label is like a variable which stands for any string of terminals which can be derived from that node in the base grammar. (Distinguish this from the usual meaning of "variable" in transformational specifications, which is any unspecified left-to-right factorization.)

This interplay between the structure of the base grammar and the structural description specifications of transformational rules, then, acts in some ways like a two-level grammar. To see how this is so, we can model this particular aspect of a transformational grammar as a van Wijngaarden grammar. (This is purely to guide intuition about the similarity; there is no suggestion that a van Wijngaarden grammar can be made to act like a transformational grammar in any satisfactory way. But on this one point, the similarity is striking.) We imagine that the base grammar of a transformational grammar is the meta-grammar, and the transformations are hyper-rules; modeling the rules on those presented at the first of this section, we could have:

```

NP ::→ NPSING | NPPL
NPSING ::→ t n
NPPL ::→ t n -s
VP ::→ AUX VPP
VPP ::→ VTCM comp | VTPR prt
VTCM ::→ VTA | VTB | ... | VTG
VTPR ::→ VTX

```

and so forth; the entire base component is simply a context-free meta-grammar. Then the passive transformation is a hyper-rule:

$\langle \text{NP1 AUX VPP NP2} \rangle \rightarrow \langle \text{NP2} \rangle \langle \text{AUX} \rangle \langle \text{be -en} \rangle \langle \text{VPP} \rangle \langle \text{by} \rangle \langle \text{NP1} \rangle$

(Notice that there is only one hyper-symbol on the left of this rule --

a string of meta-symbols -- but there are six hyper-symbols on the right side.)

This hyper-rule simply and correctly represents all the (many, many, many) rules in which a terminal string derived (in the meta-grammar) from the meta-symbols in the left-hand hyper-symbol is re-written as its correct permutation under the passive transformation. The Uniform Replacement Convention will assure that the subject and object NP's (1 and 2) are reversed, and that they are exactly the same terminal strings in the active and passive versions.

Thus, again: the reason that  $NP_{sing}$  and  $NP_{pl}$  are treated alike by transformations, is exactly the same reason that NPSYNG and NPPL are treated alike in the "passive hyper-rule"; the permutation rule is written in terms of the category NP, and both symbols can be derived from NP in the base grammar (meta-grammar).

(To bring up this comparison suggests the question of whether van Wijngaarden grammars can, or should, be used in this way to replace transformational grammars. The answer appears to be no, to both, although an attempt has been made to use van Wijngaarden grammars in just this way to describe natural languages (de Chastellier and Colmerauer 1969) from which attempt a still more general formalism for manipulating tree-structures was later developed (Colmerauer's Q-system). In any case some additional conventions are necessary to understand van Wijngaarden grammars as tree-manipulating systems, since ordinarily the trees of the meta-grammar have no interpretation. This leads to different usages, which we will not consider here.)

An important point, however, is that transformational rules gain some of their naturalness from their ability to refer to many different strings under a single variable cover-term, and their ability to specify the strings corresponding to the variable by a phrase-structure grammar (the base grammar of the transformations); this is so basic to the system that it is usually thought of as inherent, and is the property called "structure-dependence" in transformational theory.

At the same time that Chomsky was making use of this special kind of "two-level" property in his transformational grammars, there was a

quite different proposal for two-level grammars which met a bizarre fate; this was the "extended phrase structure grammar" of Gilbert Harman (Harman 1963).

The Harman proposal was not really novel, being pretty much a proposal to write rules in the programming language COMIT (Yngve 1960, 1961, 1972); Chomsky will have it that the notation was really devised by G. H. Matthews while writing a grammar of German in 1957-58, and that Matthews really developed the COMIT system as well (Chomsky 1965, pp. 79 and 213). Harman's paper has become well known only because of some unwontedly colorful remarks about ants and antelopes and about extended baboons in which Chomsky attacked it (Chomsky 1966); his criticism must have been chiefly motivated by Harman's provocative stance in maintaining that transformations had been proved to be unnecessary, because the substance of Harman's paper should not have been offensive.

Basically, the Harman rules envisioned a structured vocabulary consisting of a set of category symbols, each augmented with a set of "subscripts" or "tags". An ordinary context-free rule such as

$$A \rightarrow B C$$

is then understood to mean that A is rewritten as B followed by C, and that all of A's tags are "copied" onto B and onto C; it is just like the static notion of a schema

$$\langle a \text{ TAGS} \rangle : \rightarrow \langle b \text{ TAGS} \rangle \langle c \text{ TAGS} \rangle$$

The definition of the tags is not clearly separated, however, and thus there are many additional conventions for adding to, modifying, and erasing the tag set associated with a particular symbol. Rule features are implemented by conditions on the tags associated with the left-side symbol.

Harman's "defense of phrase-structure" (the subtitle of his paper) is that Chomsky has not properly represented the tradition of immediate-constituent analysis in defining context-free grammars; among other things, Chomsky has not represented discontinuous elements or agreement. (And this was, as we have remarked above, a legitimate complaint.) If Chomskian

phrase-structure grammars are augmented to restore these traditional parts of immediate-constituent analysis, Harman believes that such "extended phrase-structure grammars" can then adequately describe natural languages; unfortunately, however, the resulting grammars will be very large -- too large to handle in practice. Harman replies to this admitted difficulty in using extended phrase-structure grammars by saying that

the proper answer to this practical problem is that it is only a technical difficulty. Being only technically a difficulty, it can be overcome by changing techniques. We require some technique which will enable us to write and grasp millions of rules at once; that is, we require a useful way of abbreviating large sets of rules.

The desire to "write and grasp millions of rules at once" is probably the best description given so far of the motivation leading to two-level van Wijngaarden grammars and similar grammars with structured vocabulary.

But Harman's purpose of abstractly abbreviating large sets of rules gets lost in Chomsky's rebuttal, covered over by quarrels about whether the result is still properly called a phrase-structure grammar. Just how lost it was can be seen from McCawley's review of Chomsky's argument (McCawley 1968b), in which McCawley says that if Chomsky had penetrated behind the terminological question

He could have made a far more devastating criticism of it than he presented. Harman is able to dispense with agreement transformations only at the cost of having separate rules, e.g. to select the number of the subject NP (which must be attached as a feature of the S-node which dominates it, so as to allow that feature to be 'inherited' by the VP through Harman's 'feature inheritance' mechanisms), and to select the number of all other NP's. (Footnote: Because Harman neglected to include this latter type of rule in his restatement of the rules of Chomsky (1957), his rules generate only sentences in which all NP's have the same number.) Since the inheritance of features from a common dominating node in the surface structure is Harman's only means of incorporating selectional restrictions into a grammar, and since there are infinitely many types of verb-NP selectional restrictions which can hold in surface structure, Harman's treatment would require an infinite number of selectional features and infinitely many rules to attach them to the relevant S-nodes.

McCawley's account of Harman's method is quite accurate, particularly

in describing the feature-inheritance mechanism which it shares with van Wijngaarden grammars; but McCawley believes that this accurate account is a devastating criticism because of its obvious absurdity. McCawley seems not to have appreciated the point that it is not absurd to define a language by millions of rules (or by an infinite number of rules), so long as you have a technique for defining those rules which permits you to "write and grasp" them.

There are two ways in which Harman's use of structured vocabulary in grammars differs from Chomsky's use in transformations, however, and both are of interest in the context of machine-translation and computational linguistics.

First, Harman used (implicit) variables in the phrase-structure rules themselves, and did not try particularly to simulate transformations in the way explained earlier (rewriting permuted node strings); the tags of Harman's symbols are used both to enforce agreement, and to collapse rules by utilizing rule-features -- much like the similar uses by pre-Chomskian linguists.

Second, the notation scheme used by Harman was developed as a programming language, specifically for work in machine translation and natural language research, which suggests that the idea of category symbols qualified by features immediately appealed to the linguists who tried to use the early computers (hopelessly short of software) for natural language processing. In fact, some interesting work was done in COMIT, and in a version of COMIT coded into Lisp ('METEOR'; see Bobrow 1964).

#### 4.3 Post-Aspects Use of Complex-Symbol Vocabularies

The next chapter of the story is of unusual interest, because the device rejected so strongly is made the cornerstone of the theory of the base component of a transformational grammar (Chomsky 1965).

In Aspects of the Theory of Syntax, Chomsky takes the point of view that non-branching re-write rules such as

$$NP \rightarrow \left\{ \begin{array}{l} NP_{\text{sing}} \\ NP_{\text{pl}} \end{array} \right\}$$

are clearly not the correct way to achieve subcategorization. "Although this defect was pointed out quite early," he says, "there was no attempt to deal with it in most of the published work of the last several years." Chomsky gives credit for the earliest recognition of this fact to G. H. Matthews, developer of the COMIT notation for "Extended Phrase-Structure Grammars." (Other schemes are given in Bach 1964 and Schachter 1962). A base component very close in structure to a two-level grammar is proposed in Seuren 1968.)

The Aspects theory of base grammars is developed twice. First, a proposal is made to have four kinds of rules: (1) Context-free rewrite rules which develop the entire non-terminal phrase structure of a phrase-marker; (2) context-free subcategorization rules which introduce terminals with "inherent features"; (3) context-sensitive strict subcategorization rules which introduce "contextual features" of the geometry of the phrase-marker; (4) context-sensitive selectional rules which introduce "contextual features" of other terminals.

Examples of these four types of rules would be:

- (1) CF rewrite:  $S \rightarrow NP \hat{\ } \text{Predicate-Phrase}$
- (2) CF Subcategorization:  $[+N] \rightarrow [+Det \ \underline{\quad}]$   
 $[+Count] \rightarrow [+Animate]$
- (3) CS Strict Subcategorization:  $[+V] \rightarrow [+ \ \underline{\quad} \ NP] / \ \underline{\quad} \ NP$
- (4) CS Selectional:  $[+V] \rightarrow [+ [+Abstract]-Subject] / [+N, +Abstract] \text{ Aux } \underline{\quad}$

(There is a very strong resemblance between these last three types of rules which have left-sides meaning "a symbol with at least the features [f]", and the COMIT-Harmon scheme.)

The obvious redundancy of these last rules is then used to motivate



a proposal to have their effect achieved by conventions on lexical insertion, such that lexical items are inserted with their inherent features from a lexicon, observing the constraints that items with strict-subcategorization features are inserted only into conforming base structures, and items with selectional features are inserted only into base structures with conforming lexical items.

This set of conventions is then treated as defining "lexical insertion transformations", with the observation that subcategorization is achieved by "local transformations" which only affect a substring dominated by a single category symbol, and which (Chomsky suggests in a note) may be sensitive to the "vertical context" of dominating nodes as well as to the "horizontal context" usually employed in context-sensitive rules.

Rules with the properties of these Chomskian "local transformations" have recently been studied by Joshi and Levy (1977), who generalize the very satisfying result of Peters and Ritchie (1973) regarding context-sensitive rules, to the expected further result that "local transformations" (context-free rules constrained by Boolean combinations of proper-analysis predicates and domination predicates -- really quite a general definition) when used as node-admissibility conditions to constrain structural descriptions, admit terminal strings which are only context-free languages.

Thus, it is reasonable to consider the entire Aspects base component as specifying a set of derivation-initial phrase-markers in a two-level grammar, where only the hyper-grammar is made explicit (although possible meta-symbols are characterized by the redundancy rules for features), and restrictions on the hyper-symbols are stated as predicate hyper-symbols.

Additional motivation for such a formulation of the base component is afforded by more recent work (Chomsky 1970, 1977, Chomsky and Lasnik 1977) in which a complex-symbol analysis of non-terminal as well as terminal categories is used to capture additional regularities and restrict the possible rules of the base; some of this is similar to the material discussed in section 3.3 above on the X-bar convention.

(The complex-symbol notational conventions were originally developed

in connection with phonological rules, and the description of the formalism in (Chomsky and Halle 1968, pp. 390-399) may be of some use in understanding how a van Wijngaarden grammar would be employed on sets represented as strings.)

#### 4.4 Computational Linguists and Structured Vocabulary in Grammars

Without having a good understanding of what counts as structured vocabulary for grammars, it would be possible to come to the conclusion that computational linguists had made comparatively trivial use of grammars with structured vocabulary; nearly every grammar is written with names for category symbols which are related to one another, but the way in which their relatedness is exploited by the processing programs is not as obvious. A closer look, however, reveals that ideas importantly related to grammars with structured vocabulary have been used by some of the most notable computational linguistics projects, often in slightly disguised form.

The general line of development is usually considered under the heading of parsers for unrestricted rewriting systems (type-0 grammars in the terms of Chomsky 1959). Thus, one of the earliest significant systems of this sort was Yngve's COMIT programming language (Yngve 1961, 1972) designed specifically for research on natural languages and machine translation, and discussed briefly in section 4.2 above in connection with Gilbert Harman's use of the notation.

The COMIT language is based upon the format of Markov's "normal algorithms"; the individual statements in the language are for this reason called "rules", and they operate by identifying a string and re-writing it. COMIT adds labels and transfers to the notation, and the resulting "labeled Markov algorithms" are sufficiently convenient to be used for many string-oriented tasks as a general programming language (see Brainerd and Landweber 1974, Chapter 5).

The basic type-0 grammar format of the COMIT language is in principle sufficient to achieve any computation, but from its earliest versions an additional mechanism of "subscripts" was used, which amounts to a type of two-level grammar.

Each symbol in a COMIT program may have "subscripts," which may in turn have values (values are "essentially sub-subscripts" (Yngve 1961, § 11121). The subscripts are something like feature bundles, the sub-subscripts something like individual features. (Numeric subscripts are also available, which have somewhat different properties). COMIT rules manipulate the symbols as basic constituents, subject to the convention that re-writing a symbol means including its subscripts on the resulting symbols; there are a great many possible variations which may be stated, such as minimum or maximum sets of subscript features necessary on the left-side symbol for the rule to apply, and explicitly setting, erasing, and merging subscript sets.

Although a COMIT 'grammar' (program) is only a one-level grammar, its vocabulary is structured in a way which -- like a two-level grammar -- makes it possible to abbreviate large sets of rules in schemata. There is nothing in the theory of Markov algorithms to suggest this step, so its inclusion must have been prompted by the linguist-designers' feelings that for natural-language rules the use of systematically structured symbols would enhance the ease and naturalness of using the COMIT system. Given the developments in linguistics reviewed in previous sections, this is not surprising.

There are two principal paths of development from this early work of Yngve's (and of G. H. Matthews', according to Chomsky). The first is the work of pattern-matching, which results in Snobol4 (Griswold et al. 1971, Gaskins and Gould 1972) and its underlying theory (Gimpel 1973, 1975, 1976). Although related to two-level grammars, this line will not be followed up here. The other line of work is in type-0 parsers, and here the most influential publication is Martin Kay's "powerful parser" (Kay 1967).

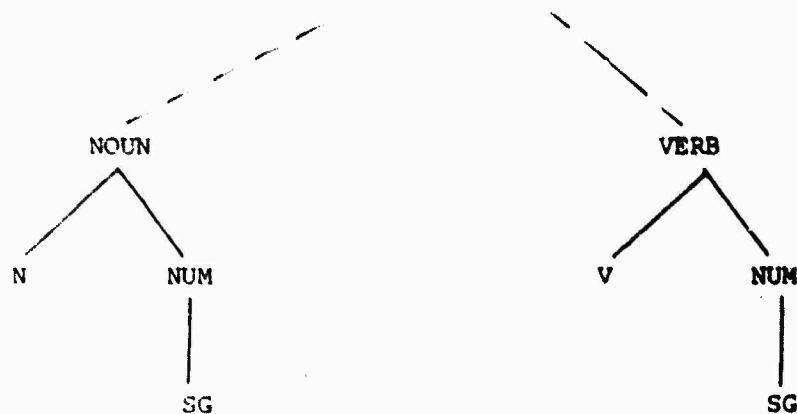
Kay's grammars are again not well separated into levels, but they contain rules such as:

SG.1 = NUM(1)

PL.1 = NUM(1)

N.1 NUM.2 V.3 2 = NOUN(12) VERB(32)

(Since these are recognition rules, Kay writes them backwards -- the left side of the rule is on the right side of the paper, and vice versa. The numbers after dots assign identifying numbers; the numbers in parentheses use those identifications to build constituent structure.) The first two rules here recognize either singular or plural number as being of category NUM. (The single item -- dot one -- is made the sole constituent of a NUM node -- parenthesized one.) The third rule then recognizes four elements: (1) a noun N, (2) a number morpheme NUM, (3) a verb V, and (4) a second number morpheme which is the same (SG or PL) as item two -- and so this fourth item is not assigned a number of its own. These four items are rewritten as two items (the left side of the rule, seen here on the right side of the paper): (1) a noun phrase NOUN dominating right-side items 1 and 2, and (2) a verb phrase VERB dominating right-side items 3 and 4 (3 and 2, since 4 and 2 are required to be identical). After this rule has applied a possible partial parse-tree would be



Agreement of the SG's during recognition is forced here, it should be noted, by the analogue of the Uniform Replacement Convention; the two instances of NUM can be required to have the same value because they are recognized in a single rule; and that is why what are essentially two unrelated context-free rules

NOUN → N + NUM

VERB → V + NUM

are applied together in a single type-0 rule application.

Unfortunately not too much development of this sort of use is given, because the "main concern" of Kay's paper is "to discuss the extent to which the program we have been describing can be made to function as a transformational analyzer." A compulsion to use a type-0 grammar to effect "transformations" runs through this whole line of work. It is exactly the same ability as the use of van Wijngaarden grammars to effect "transformations" which we discussed above (section 4.2), and unfortunately it always exceeds the range of manageable complexity rather quickly.

Immediate successors of Kay are Kaplan's General Syntactic Processor (Kaplan 1973) -- Kay's parser plus some extras from Woods's ATN's -- and the REL (Rapidly Extensible Language) System (Dostert and Thompson 1971, 1972). The REL development is of some interest, because to the original concept of Kay's type-0 parser has been added a layer of features, so that the resulting grammar has the same structure as a COMMIT program.

Coverage of a large part of English is claimed, with only 300 rules in the REL English grammar. These are really rule schemata, like hyper-rules, since a rule has the form:

VP' → NP VP

FEATURE CHECK: VP must be (-Passive) (-Subject) and (-Agentive).

FEATURE SET: Assign (+Subject) and (+Agentive) to VP' together with the features of VP.

This summarizes (as we may reinterpret the formalism) all the rules (production-rules) in which categories representing all kinds of feature-bundles participate, so long as the feature-bundles of NP' and VP are related as "dynamically" specified in the final condition, and so long

as the feature-bundle of the VP has the specified values. Hidden under the procedural terminology of "check" and "set" or "assign" we discern a hyper-grammar with hyper-rules such as

$$\langle \text{vp-prime VFEATURES1} \rangle \rightarrow \langle \text{np NFEATURES} \rangle \langle \text{vp VFEATURES2} \rangle$$

$\langle \text{where VFEATURES2 includes } (-\text{pas}, -\text{subj}, -\text{agt}) \rangle$

$\langle \text{where VFEATURES1 is VFEATURES2 but } (+\text{subj}, +\text{agt}) \rangle$

In this version the "feature check" and "feature set" actions of the REL rule have become the two predicate hyper-notions. It is not too hard to see how to define predicates like these syntactically, although in an implementation they would of course be implemented just as the "check" and "set" actions are implemented in REL.

The importance of the features, or rather the inadequacy of a grammar of 300 rules, is not easy to overestimate. The example sentence "Has John attended the school of Cambridge's Mayor?" is said to parse unambiguously in REL English with features, but to be 2,701-ways ambiguous in the same grammar without features (Dostert and Thompson 1972). In REL English the subscript features are said to have three roles: (1) to subcategorize parts of speech; (2) to prevent ungrammatical strings (i.e., to collapse nearly-identical rules); and (3) to determine the preferred order of syntactic groupings (such as preventing multiple ambiguities in strings of nouns or adjectives -- a motivation analogous to that for Harris's superscript numbers).

The REL grammar is rather easy to see as a very large context-free grammar abbreviated in a way somewhat like a van Wijngaarden grammar; the "category symbols" provide a gross check on the applicability of a rule during parsing, and only if that test is passed is it necessary to check to see if a detailed rule which is a refinement of the gross form is actually applicable. (And, the detailed rules do not physically exist, but are made up on the fly from the abbreviatory schemata plus the features actually discovered to be present.)

This same form of grammar has been used in several other recent computational linguistics projects, but concealed still further in

"procedural" terminology, and making use of program fragments instead of a more abstractly-defined data base of rules. This is the work of Woods, of Sager, and of Winograd.

It is perhaps no accident that each of these three approaches is somewhat "an implementation in search of a theory." The engineering approach is to write a program and work out any problems as they come up, and the linguistic engineers of early machine translation days fell into the same error (see, e.g., Garvin 1966 for a defense of the practice) of encoding their grammars as recognition procedures. It is undeniably odd, however, that such a practice should have persisted up to the present. Outside the circles of the artificial intelligentsia, the current view is rather more typified by Grishman's remarks that "The 'grammar in program' approach which characterized many of the early machine translation efforts is still employed in some of today's systems." "...research goals should be the ability to manage grammatical complexity and the ability to communicate successful methods to others. In both these regards, a syntactic analyzer using a unified, semi-formal set of rules is bound to be more effective (Grishman 1976)." Today's systems, it should be noted, are more apt to have some set of formal rules, but then to compromise this by inserting in the rules the names or addresses of arbitrary bits of program to carry out procedures -- thus effectively putting an essential part of the grammar into programs, which are hard to control.

Woods's Augmented Transition Networks (Woods 1969, 1970, 1975) are the best-known example of such a procedural way of analyzing natural language. They actually represent a use of structured vocabulary, although because of the organization of the systems they sometimes give the impression of being completely ad-hoc recognition systems. They are, in the usual sincere flattery claimed for this kind of work, "capable of doing everything that a transformational grammar can do" (Woods 1970), in the usual uninteresting sense. In addition to the use of structured vocabulary which we shall attempt to identify, the ATN grammars also model themselves after the special factored form of a "regular right-part grammar" explained in section 2.4. They further confuse matters by interspersing actions

to build a tree structure which is distinct from the structural description of the string analysed; this is of no interest here, being merely an ill-structured translation of a context-free grammar made at an inconvenient time.

In a Woods ATN, the structured vocabulary of the grammar is nowhere explicit, but is held in the state of various (software) registers over time. When a subject NP is recognized, the features of its head noun are put into a special "subject register" by an "action"; then, later, when a VP is at hand, the subject register is interrogated by a "condition" which checks compatibility.

This is obviously one possible low-level implementation of a two-level grammar -- although Woods's grammars have been naive in details such as providing a limited set of grammatical relations to program into registers, and in attaching features to words alone so that other actions must "look inside" larger constituents to find the features (Burton and Woods 1976).

It would perhaps be worth exploring the application of a more abstract van Wijngaarden approach to ATN's -- introducing many states with structured names, and so forth, while retaining the regular-right-part format --- to see whether those who like ATN's would like the resulting version. Such a development would be merely a notational variant of the restricted grammars with structured vocabulary to be introduced in following sections, and would remove the procedural flavor from the definition of conditions and actions, while retaining it for the basic recursive networks.

A similar project using similar means is the "Linguistic String Project" of Sager, which is influenced by Harris's String Analysis notions (Harris 1962). As Sager and Grishman observe, "This basic strategy of grammar design, in which a context-free framework is augmented by a set of conditions written as procedures, has become quite popular; it is used, for example, in the systems of Woods and Winograd" as well as in their own Linguistic String Project system (Sager and Grishman 1975). Their implementation again employs "registers" which are set and checked, and if anything it is less constrained than Woods's systems



because they have invented a "restriction language" in which to program the conditions.

The last of these implementations which we shall mention is that of Terry Winograd. Winograd is much influenced by the idea of the compactness in a structured vocabulary built on category symbols plus features:

We allow each symbol to have additional subscripts, or features, which control its expansion. In a way, this is like the separation of the symbol NP into NP/PL and NP/SG in our augmented context-free grammar. But it is not necessary to develop whole new sets of symbols with set of expansions for each. A symbol such as CLAUSE may be associated with a whole set of features (such as TRANSITIVE, QUESTION, SUBJUNCTIVE, OBJECT-QUESTION, etc.) but there is a single set of rules for expanding CLAUSE. These rules may at various points depend on the set of features present. (Winograd 1971)

This is not a bad description of the practical advantages of structured vocabulary and rule schemata, as we have described them previously.

Unfortunately, Winograd somehow came to believe that M. A. K. Halliday is the only professional linguist who shares this appreciation (probably because only there did he see explicitly written features, outside of Chomsky), and so Winograd develops his own grammar in the form of a program working on Halliday's hints about "systemic grammar" (Halliday 1961). Winograd rapidly programs himself into an ad-hoc mess:

How, for example, can we handle agreement? One way to do this would be for the VP program to look back in the sentence for the subject, and check its agreement with the verb before going on. We need a way to climb around on the parsing tree, looking at its structure.

...The call (\*C DLC PV (NP)) will start at the current node, move down to the rightmost completed node (i.e., not currently active) then move left until it finds a node with the feature NP (Down-Last-Completed, PreVIOUS)....

When this idea is elaborated over several years, the result is a hacker's dream. This is precisely the sort of approach to the advantages of a structured vocabulary from which we are saved by the invention of a two-level van Wijngaarden grammar.

## 5. Restrictions on Grammars with Structured Vocabulary

We have seen in section 2 above that context-free grammars of the classic one-level type are in practice unwieldy for describing natural languages. In section 3, we saw that by introducing a structured vocabulary in a van Wijngaarden "two-level" grammar, it was possible to overcome some of the practical difficulties, but that the resulting class of grammars contains formidably complex systems, equivalent to unrestricted rewriting systems. More than just being theoretically powerful, there is the practical question of how to apportion function for maximum insight in a two-level grammar.

In the review of prior linguistic uses of structured vocabulary in section 4, we have seen that there is a strong tradition going back to pre-Chomskian linguistics to work with basic units of syntactic categories, qualified by the addition of features (tags, subscripts) to provide for agreement or co-occurrence restrictions and to permit rule features to collapse nearly identical rules. We saw that this tradition was continued without question by early implementors of systems and languages for natural language processing; and that because the same formal devices could be used to encode "transformations" or tree-manipulation rules, that purpose was added to the traditional uses of structured vocabulary by some computational linguists. It also appeared that these last extensions have been counterproductive, and that the two purposes of abbreviating a large context-free grammar and of transforming structural descriptions should be conceptually separated.

Accordingly, we examine in this section restrictions designed to model a grammar with category symbols and features, as a restriction on the form of van Wijngaarden grammars. These restrictions are not introduced to alter to weak generative capacity of the grammars (the restricted grammars are still type-0 grammars), but they do restrict grammars to a class which is easier to write and easier to parse. Moreover, by restricting the format of grammars it is possible that the notation can be made less cumbersome.

The restrictions proposed here are somewhat similar to those in (Greibach 1974), which reduce the generative capacity of the grammars

to a sub-class of context-sensitive languages and yield other pleasant theoretical properties; but here our interest is exclusively in the practical ease and naturalness of the grammars when written by people to describe natural languages.

It is likely that grammars of the class described in this section can be written which would be adequate to serve as recognition grammars for parsing natural languages, in such practical applications as machine translation. These grammars do not effect any inter-language transformation or correspondence, which would be left to a separate component.

### 5.1 A Restricted van Wijngaarden Grammar

In this section we will introduce a simple van Wijngaarden grammar to define an artificial language; its interest lies in the fact that the grammar is constructed using a restricted part of the potential techniques available in a van Wijngaarden grammar. Generally, we mean to restrict every non-predicate hyper-symbol to be a single proto-symbol (the "category symbol") and a string of meta-symbols (the "features"), and further to require that each proto-symbol is always accompanied by the same set of meta-symbols. Only predicate hyper-symbols (those which dominate only EMPTY or blind alleys) are not restricted in this way. The style of grammar which results from these restrictions will be shown in the following sections, where alternative notations for such restricted grammars will be shown and exemplified by transcribing the same grammar into them. Following these samples, a more complex grammar related to natural languages (though still simplified for exposition) will be exhibited in all three notations.

Suppose we wish to define a language in which names may be "declared" and "used", in a way much like in ordinary programming languages. (This example language is modeled after that of (Watt 1977), and is related to the larger example to follow which concerns the proper characterization of quantified logical formulas.) In this language every name must be declared (as "new information") before it is used (as "old information"); no name may be declared more than once, and every name must be declared before it is used. (Names may be declared without being used, however,

and may be used more than once after being declared.) This language resembles a programming language with semi-strict declarations and without block structure.

For example, good strings in this language are ones such as

dcl x use x end

in which x is declared (dcl) and then used (use), and other good strings would be:

dcl x dcl y use y use x end

dcl z dcl x dcl y use x use z use z end

dcl x use x use x use x end

But this language does not include strings such as the following:

\*dcl x use x use y end (no declaration for y before use)

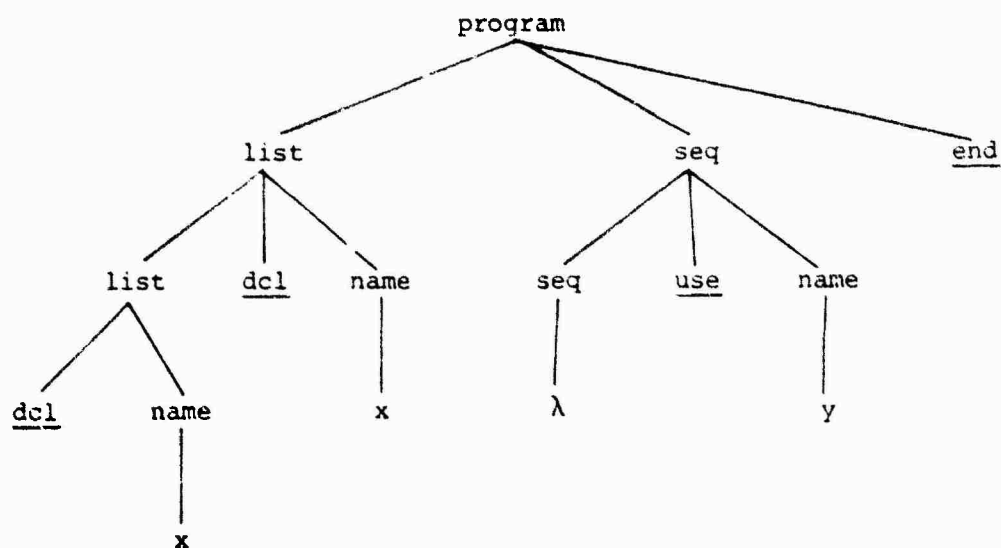
\*dcl x dcl x use x end (x declared twice)

We will be employing only the names x, y, and z, so special ad-hoc methods could be used to define this language; however, it is a well-known theorem that in general languages like this one are not context-free languages (have no grammars which are context-free grammars), so this will serve as a sample of a language which has no context-free grammar.

A context-free grammar which gives strings of the correct general form is the following, which generates "programs" composed of a "list" of declarations followed by a "sequence" of uses:

```
program → list . seq end
list → dcl name | list dcl name
seq → λ | seq use name
name → x | y | z
```

(As before lower-case lambda [ $\lambda$ ] is the empty string.) A typical structural description derived in this grammar would be:



dcl x dcl x use y end

Of course, as this sample indicates, the restrictions on declaration and use are not observed in this grammar, so that the tree above is generated by the grammar, but the terminal string is not one which we wished to be in the language (x is declared twice and y is used without declaration). We can correct this flaw by employing a two-level van Wijngaarden grammar in which the list of declarations and the sequence of uses are constrained to be the same members, after which the declarations are peeled off one by one (destructively) and the uses are checked for membership. Such a grammar is the following:

Meta-rules:

- m1. NAME  $::\rightarrow$  x | y | z
- m2. SET  $::\rightarrow$  NAME | SET NAME
- m3. EMPTY  $::\rightarrow$   $\lambda$

Restricted Hyper-rules:

- h1.  $\langle$ program $\rangle$   $::\rightarrow$   $\langle$ list SET $\rangle$   $\langle$ seq SET $\rangle$   $\langle$ end $\rangle$

- h2.  $\langle \text{list SET} \rangle \rightarrow \langle \underline{\text{dcl}} \rangle \langle \text{name NAME} \rangle$   
        $\langle \text{where NAME is not in EMPTY} \rangle$   
        $\langle \text{where SET is union of EMPTY NAME} \rangle \mid$   
        $\langle \text{list SET1} \rangle \langle \underline{\text{dcl}} \rangle \langle \text{name NAME} \rangle$   
        $\langle \text{where NAME is not in SET1} \rangle$   
        $\langle \text{where SET is union of SET1 NAME} \rangle$
- h3.  $\langle \text{seq SET} \rangle \rightarrow \langle \text{EMPTY} \rangle \mid$   
        $\langle \text{seq SET} \rangle \langle \underline{\text{use}} \rangle \langle \text{name NAME} \rangle$   
        $\langle \text{where NAME is in SET} \rangle$
- h4.  $\langle \text{name NAME} \rangle \rightarrow \langle \text{NAME terminal} \rangle$

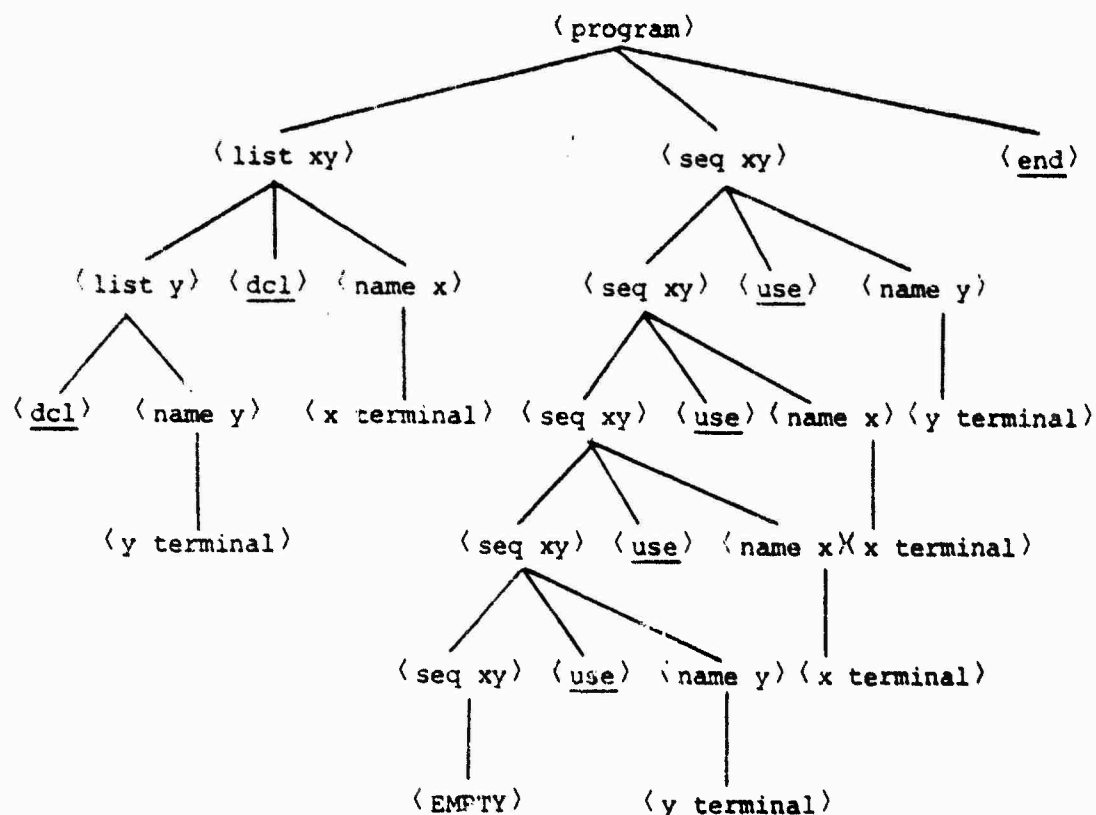
Additional meta-rules and hyper-rules to define predicate hyper-symbols  
 (first-time readers may skip the remaining rules):

- m4.  $\text{OPTIONAL} \rightarrow \text{NAME} \mid \text{EMPTY}$
- h5.  $\langle \text{where NAME1 is not in NAME2 SET} \rangle \rightarrow$   
        $\langle \text{where NAME1 is not NAME2} \rangle$   
        $\langle \text{where NAME1 is not in SET} \rangle$
- h6.  $\langle \text{where NAME1 is not in NAME2} \rangle \rightarrow$   
        $\langle \text{where NAME1 is not NAME2} \rangle$
- h7.  $\langle \text{where NAME1 is not NAME2} \rangle \rightarrow$   
        $\langle \text{where NAME1 precedes NAME2 in xyz} \rangle$   
        $\mid \langle \text{where NAME2 precedes NAME1 in xyz} \rangle$
- h8.  $\langle \text{where NAME1 precedes NAME2 in OPTIONAL1 NAME1 OPTIONAL2}$   
        $\text{NAME2 OPTIONAL3} \rangle \rightarrow$   
        $\langle \text{EMPTY} \rangle$

- h9.  $\langle \text{where NAME is not in EMPTY} \rangle : \rightarrow$   
        $\langle \text{EMPTY} \rangle$
- h10.  $\langle \text{where NAME1 is in NAME2 SET} \rangle : \rightarrow$   
        $\langle \text{where NAME1 is NAME2} \rangle$   
       |  $\langle \text{where NAME1 is in SET} \rangle$
- h11.  $\langle \text{where NAME1 is in NAME2} \rangle : \rightarrow$   
        $\langle \text{where NAME1 is NAME2} \rangle$
- h12.  $\langle \text{where NAME is NAME} \rangle : \rightarrow$   
        $\langle \text{EMPTY} \rangle$
- h13.  $\langle \text{where EMPTY is in EMPTY} \rangle : \rightarrow$   
        $\langle \text{EMPTY} \rangle$
- h14.  $\langle \text{where SET1 is union of SET2} \rangle : \rightarrow$   
        $\langle \text{where SET1 is subset of SET2} \rangle$   
        $\langle \text{where SET2 is subset of SET1} \rangle$
- h15.  $\langle \text{where SET1 NAME is subset of SET2} \rangle : \rightarrow$   
        $\langle \text{where NAME is in SET2} \rangle$   
        $\langle \text{where SET1 is subset of SET2} \rangle$
- h16.  $\langle \text{where NAME is subset of SET} \rangle : \rightarrow$   
        $\langle \text{where NAME is in SET} \rangle$
- h17.  $\langle \text{where EMPTY is union of EMPTY} \rangle : \rightarrow$   
        $\langle \text{EMPTY} \rangle$

This van Wijngaarden grammar generates exactly the language specified, with all restrictions observed. (The detailed syntax of the predicate hyper-symbols in rules h5 -- h17 will not be discussed here, but generally follows the pattern introduced in section 3.5 above, with which comparison would be useful.)

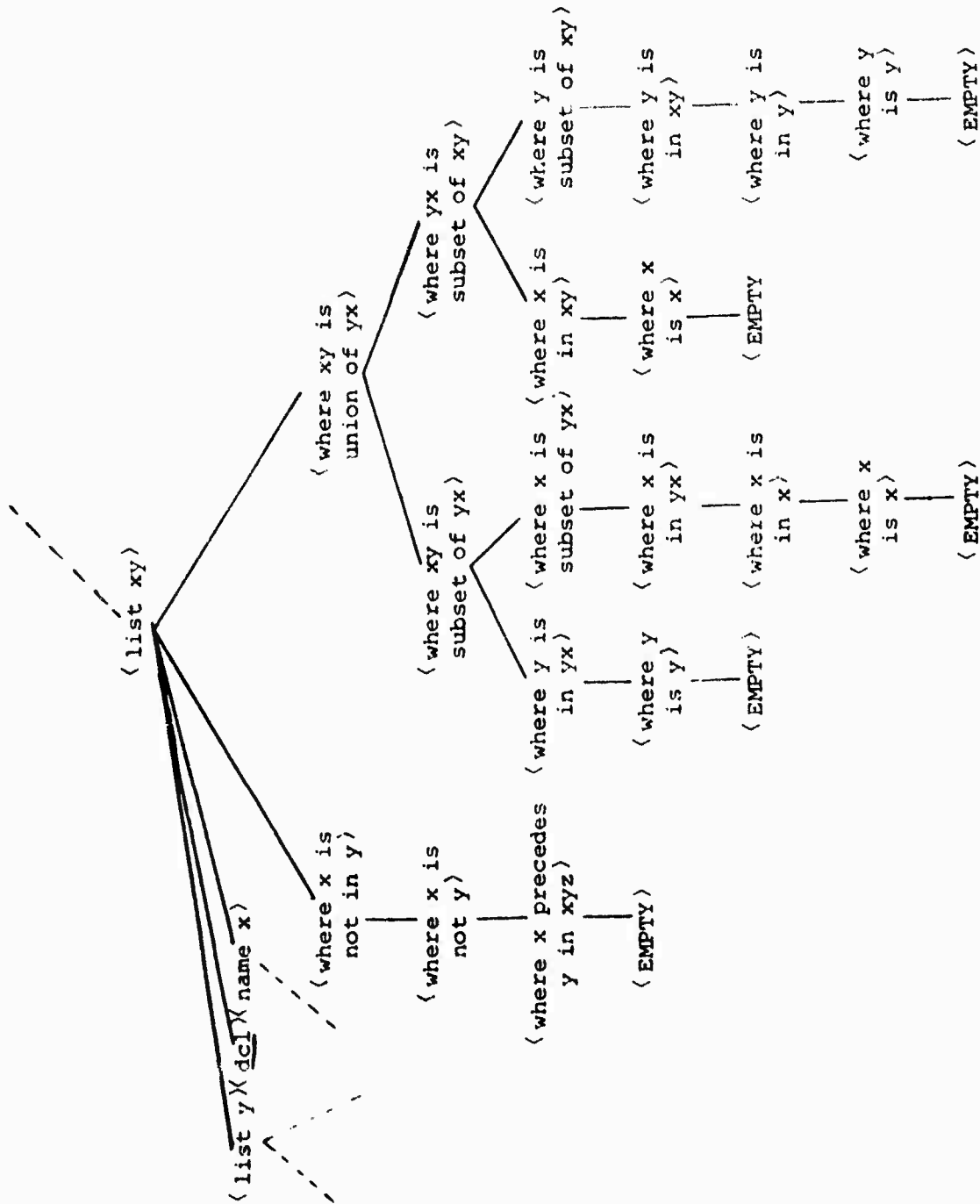
A sample structural description in this grammar would be:



dcl y dcl x use y use x use x use y end

(This grammar, for simplicity, uses the convention that either the underlined symbols or a production-symbol ending in 'terminal' are terminals.) The predicate hyper-symbols in the hyper-rules enforce that only one declaration per name can take place in the left branch and that names used on the right are in the declared set; but observe that the extensive sub-trees dominating only EMPTY and headed by predicate symbols have been suppressed in the tree above. A fuller diagram would be much more complicated at every level -- a sample level would be:





This shows all the rule applications which had to be possible in order for the rule rewriting  $\langle \text{list } x y \rangle$  as  $\langle \text{list } y \rangle \langle \text{dcl } \rangle \langle \text{name } x \rangle \langle \text{where } x \text{ is not in } y \rangle \langle \text{where } x y \text{ is union of } y x \rangle$  to be part of a valid derivation. The last two hyper-symbols (the two predicates) and all their dominated material was removed from the structural description because it dominated only EMPTY terminals. Trees like this are generated at each declaration, and a smaller tree is generated at each use. If the sample derivation had not been correct, then some of the predicate nodes would not have been able to generate only EMPTY, because the derivation would have blocked at a non-terminal which could not be re-written.

It should be clear, looking at this grammar, how proto-symbols such as 'list' and 'seq' are used as category symbols, while meta-symbols such as 'SET' and 'NAME' are used as features. The production-rules generated from the basic hyper-rules will re-write a category symbol and any possible feature set, leaving it to the rules re-writing predicate hyper-symbols to block the derivations containing features which are not correctly arranged.

## 5.2 Koster's Affix Grammars

In the preceding section we exhibited a correctly-matched structural description and associated string in the language of the sample van Wijngaarden grammar. But it is a nice question how we could have started with the string and the grammar, and discovered the structural description (if any). We could not, for instance, have expanded the van Wijngaarden rules out to their equivalent production rules, since there are an unbounded number of production rules produced by the first hyper-rule alone:

SET  $:: \rightarrow$  NAME | SET NAME

$\langle \text{program} \rangle :: \rightarrow \langle \text{list SET} \rangle \langle \text{seq SET} \rangle \langle \underline{\text{end}} \rangle$

Any value of SET can be used to form a production rule, even though not all sets initiate valid derivations (for example, those with repeated instances of the same name do not).

Clearly we need to begin the other way, and to find some method

to discover from a string to be parsed what rules are relevant to it -- and then see if those rules exist. One proposed way to do this is to re-formulate a van Wijngaarden grammar as a different but related kind of "affix grammar" specifically designed to make this strategy trivial (Koster 1971, 1974b, 1975, Crowe 1972, Watt 1977). This must be done by hand, since not every van Wijngaarden grammar can be so re-written, and the conversion is not mechanical. But we will examine affix grammars here, anyway, since they employ a notation which is related to our restricted van Wijngaarden grammars and since they do suggest intuition as to how parsing could proceed.

An affix grammar can be seen as a restricted van Wijngaarden grammar which (like those of the preceding section) contains only non-predicate hyper-symbols which consist of a single proto-symbol and a string of meta-symbols, always the same string for any single proto-symbol. But in addition, for each appearance of any meta-symbol in a rule the grammar writer must specify whether it is "inherited" or "synthesized" -- that is, whether its value in that appearance depends on the values of symbols in its constituents alone (synthesized), or whether its value depends in part on the context of its use (inherited). Finally, predicates are defined in non-syntactic ways (for convenience), and may compute the values of some meta-symbols. Since the meta-symbols in this sort of grammar so clearly qualify the proto-symbols, they are called "affixes".

As an example, we give here an affix grammar for the same language defined in the previous section. The new notation here consists of a rising arrow  $\uparrow$  to precede synthesized affixes, and a down arrow  $\downarrow$  to precede inherited affixes. This definition can be compared rule for rule with the van Wijngaarden grammar given before.

- m1. NAME  $\rightarrow$  x | y | z
- m2. SET  $\rightarrow$  NAME  $\downarrow$  SET NAME
- m3. EMPTY  $\rightarrow$   $\lambda$

(affix-style hyper-rules:)

```

h1. (program) :→ (list ↑SET) (seq ↓SET) (end)
h2. (list ↑SET) :→ (dcl) (name ↑NAME)
           (add(EMPTY,NAME) return (SET)) |
           (list ↑SET1) (dcl) (name ↑NAME)
           (add(SET1,NAME) return (SET))
h3. (seq ↓SET) :→ (EMPTY) |
           (seq ↓SET) (use) (name ↑NAME)
           (identify(SET,NAME))
h4. (name ↑NAME) :→ (↑NAME terminal)

```

Here "add" and "identify" are names of predicates; they are defined procedurally, by giving the ranges of their parameters and a specification of their actions:

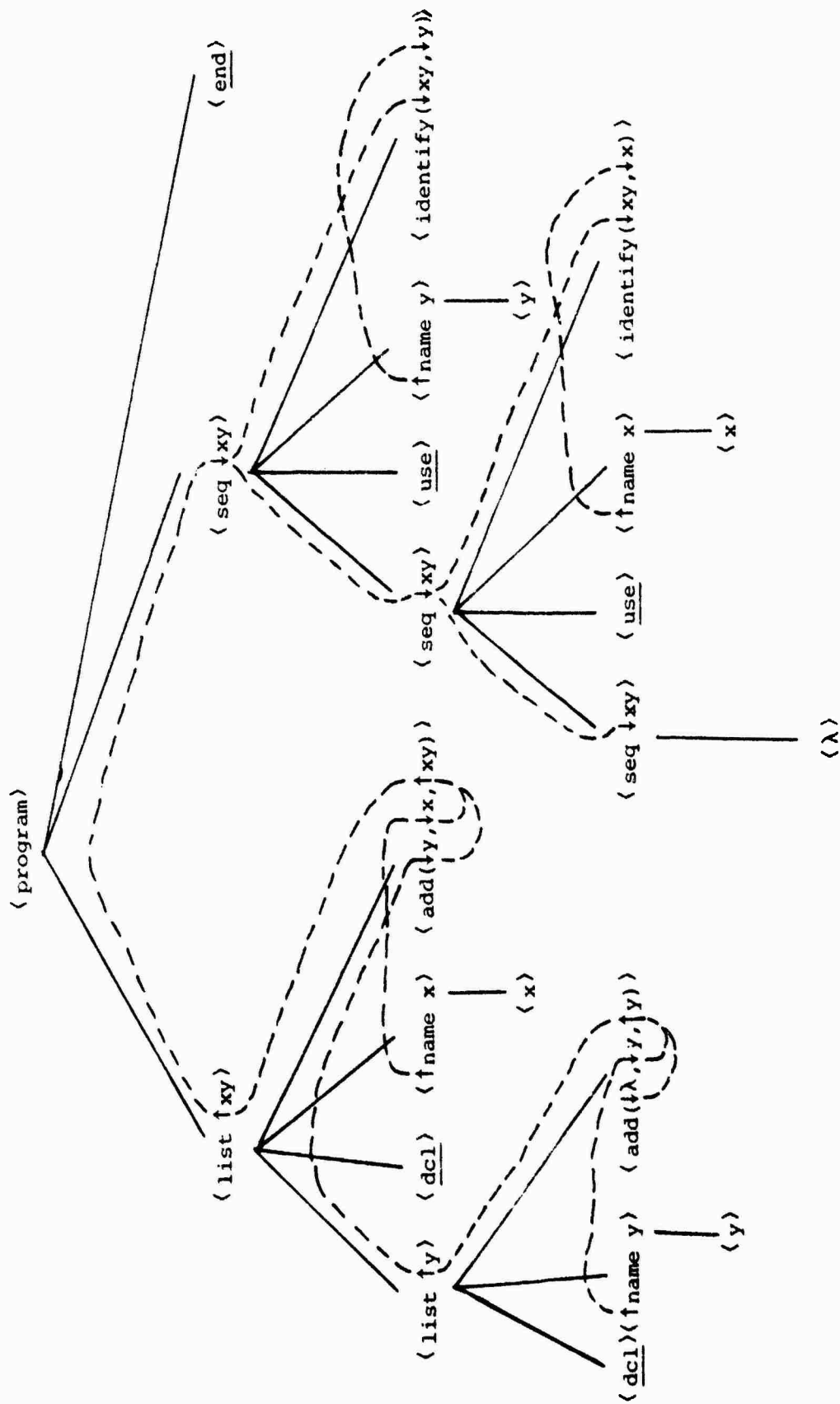
Predicates:

<u>name</u>	<u>input parameters</u>	<u>result parameters</u>	<u>function</u>
add	1. SET1 2. NAME	3. SET	<u>if</u> NAME ∈ SET1 <u>then</u> <u>fail</u> <u>else</u> SET := SET1 ∪ {NAME} <u>fi</u>
identify	1. SET 2. NAME	--	<u>if</u> NAME ∈ SET <u>then</u> <u>succeed</u> <u>else</u> <u>fail</u> <u>fi</u>

It will be observed that the basic rules here are typographically

almost identical to the corresponding rules of the van Wijngaarden grammar, but these rules contain extra information.

Interpreted as a van Wijngaarden grammar, this affix grammar defines the same language in exactly the same way as before -- that is, the language generated by production-rules which can be obtained from the affix-grammar rules by the Uniform Replacement Convention. But hidden in the arrows is a further assertion that a possible parsing strategy would be to construct a parse tree according to the proto-symbols alone, and then check the affixes in the order indicated by the arrows, letting the  $\uparrow$  affixes move up the tree, while the  $\downarrow$  affixes move down. For example, the parse tree corresponding to an example similar to the one given before would be:



dcl y dcl x use x use y end

In this diagram, the dotted arrows running through the affixes of every node describe a feasible pattern of checking which would be effective; beginning with the terminal names, values are synthesized upward and inherited back down, computing the value of all the affixes and checking the compatibility of them. In general, of course, the declarations are gathered moving up the left side of the tree, and then uses are checked moving down the right side of the tree.

Without going too deeply into the restriction philosophy, we can distinguish in any hyper-rule affixes which are in "defining occurrences" from those which are in "applied occurrences." Defining occurrences are those of (i) inherited affixes on the left side of a rule, or (ii) synthesized affixes on the right side of a rule. Applied occurrences are just the opposite: (i) synthesized affixes on the left side, or (ii) inherited affixes on the right side. For example, in

$$\langle \text{list } \uparrow \text{SET} \rangle : \rightarrow \langle \text{dcl} \rangle \quad \langle \text{name } \uparrow \text{NAME} \rangle$$

$$\langle \text{add}(\text{EMPTY}, \text{NAME}) \text{return}(\text{SET}) \rangle$$

the synthesized affix  $\uparrow \text{NAME}$  is "defining" on the right, while the synthesized affix  $\uparrow \text{SET}$  is "applied" on the left; this corresponds to passing information up the tree while parsing, and the predicate "add" computes a value for SET using the value of NAME.

Affix grammars in this form are usually subject to some additional constraints, because they have been defined for programming languages where the goal is to parse very rapidly, and the constraints make it possible to parse in one pass over the input string left to right, propagating affixes through the tree in the same pass as parsing is completed. We will not explore the effect of these constraints here, because for natural language analysis a looser set of constraints is inevitable -- and that is the subject of the following section.

What is notable, however, is that the restriction of van Wijngaarden grammars to be essentially context-free grammars with features and tests rapidly moves them quite far down the scale of generality, so that convenient parsing algorithms become available.

And, as we might expect, the resulting grammars are also easier for humans to read and understand. Simonet (1977) suggests defining programming languages by van Wijngaarden grammars, but restricted van Wijngaarden grammars modeled after affix grammars for ease of human use. That suggestion seems plausible, given the widespread popularity of the same representation for natural language analysis.

### 5.3 Knuth's Attribute Grammars

Still another variant of a grammar with structured vocabulary is the sort introduced by Donald Knuth and now generally referred to as an "attribute grammar" (Knuth 1968, 1971, Fang 1972, Wilner 1972). Although originally motivated by rather different goals, attribute grammars may for our purposes be considered as simply a set of notational proposals for affix grammars or van Wijngaarden grammars.

The advantage of the Knuth approach is that it is once again quite easy to achieve an abstract, "declarative" way of looking at the grammar. In a van Wijngaarden grammar, the declarative view is obvious: a class of well-formed strings and their associated structural descriptions is characterized, but no procedure for parsing is implied. Affix grammars, by contrast, use a notation which suggests preoccupation with passing things around, one step after another. A little bit of this is useful to suggest how practical implementations could be achieved, but to insist on this procedural view is to complicate the task of writing a grammar by raising to notice just those details which it is the glory of a grammar to suppress. Affix grammars can be viewed as declarative, with a bit of effort, but once they are viewed in that way the Knuth approach may recommend itself as more natural.

A first orientation to attribute grammars should include a warning that (unlike affix grammars) attribute grammars are not (explicitly) two-level grammars, were not motivated by the syntax of Algol 68, nor were they introduced to admit of efficient parsing. Instead, attribute grammars were proposed by Knuth as a way to specify the "Semantics of Context-Free Languages" (the title of Knuth 1968). The idea was to associate "attributes" with the categories of a context-free grammar,



and to associate rules for computing attributes with the rules of the grammar. Such attributes could be more elaborate than "features" in the usual sense -- lists, functions, and especially trees could represent the "meaning" of a phrase expressed in some suitable way. This idea is so fruitful, and has been used in so many ways, that no proper appreciation of Knuth's proposals can be gained from the limited use of attribute grammars exemplified here. Still, this is an interesting and valid use of attributes; it has before been suggested by (Agafonov 1976) briefly.

In Knuth's arrangement, as in affix grammars, every category symbol has a fixed repertoire of named attributes which always accompany it at every appearance, and any particular attribute is invariably "inherited" or "synthesized" (in contrast to an affix grammar, where each appearance of an attribute is one or the other, but where a single attribute may appear in both roles from time to time). This being so, it is thought redundant to write out all the attributes repeatedly after every category symbol of the grammar in an invariant fashion, so they are removed to another table to accompany the rules. For our continuing example from the previous sections, we would have to break the single affix SET into two attributes, say PSET for the synthesized uses of declared names, and ISET for the inherited uses of checking when names are used.

If the attributes are not mentioned in the rules of the grammar, then it would seem that we could simply copy the original context-free rules of section 5.1 with which we started -- and that is true. But each syntactic rule will be accompanied by semantic rules, which will specify the relations among the attributes which must hold if the (non-context-free, semantic) restrictions of the language are to be met. In such semantic rules, an attribute of a category symbol is referred to by writing the attribute name before the symbol in parentheses; e.g., DR (list), which is read "DRSET of list."

The semantic rules are in part the counterparts of predicate hyper-symbols in a van Wijngaarden grammar, but every "defining occurrence" of an attribute (a synthesized attribute on the left, an inherited attribute on the right) must be the subject of a condition relating

its value to the values of other attributes in the same rule.

A sample grammar (for the same language as those of the preceding sections, with which the rules may once again be compared one for one) would be:

Attributes:

<u>name</u>	<u>inherited or synthesized</u>	<u>type value</u>	<u>for category symbols</u>
NAME	synthesized	string	name
DSET	synthesized	set of names declared	list
USET	inherited	set of names used	seq
ERR	synthesized	Boolean (true or false)	name, list, seq, program

Rules:

SYNTAX		SEMANTICS	
1. program $\rightarrow$ list seq <u>end</u>	1.1a	USET(seq) = DSET(list)	
	1.1b	ERR(program) = ERR(list) .OR. ERR(seq)	
2. list $\rightarrow$ <u>decl</u> name	2.1a	DSET(list) = NAME(name)	
	2.1b	ERR(list) = ERR(name)	
list <u>decl</u> name	2.2a	DSET(list1) = DSET(list2) $\cup$ {NAME(name)}	
	2.2b	ERR(list1) = ERR(list2) .OR. (if NAME(name) $\in$ DSET(list2) then true else false)	
3. seq $\rightarrow$ $\lambda$	3.1a	ERR(seq) = false	
seq <u>use</u> name	3.2a	USET(seq2) = USET(seq1)	
	3.2b	ERR(seq1) = ERR(seq2) .OR. (if NAME(name) $\in$ USET(seq1) then false else true)	
4. name $\rightarrow$ x   y   z	4.1a	NAME(name) = (string name)	
	4.1b	ERR(name) = false	

(In semantic rules, digits following node names are used to distinguish repeated instances of the same symbol -- DSET(list1) is the DSET attribute of the first occurrence of the category symbol 'list' in the rule, etc.)

There are two ways in which this attribute grammar is more explicit than the affix grammar. First, no analogue of the Uniform Replacement Convention is assumed. Thus, it is necessary to say explicitly that in rule 1

$$\text{program} \rightarrow \text{list seq } \underline{\text{end}}$$

the DSET of 'seq' must be the same as the DSET of 'list', by adding the semantic condition

$$\text{DSET(seq)} = \text{DSET(list)}$$

(In the affix grammar, the rule

$$\langle \text{program} \rangle \rightarrow \langle \text{list } \uparrow \text{SET} \rangle \langle \text{seq } \downarrow \text{SET} \rangle \langle \underline{\text{end}} \rangle$$

made use of the URC to assure the same result, quite apart from the information in the arrows that the one can be computed from the other.)

In this brief grammar that lack is no handicap, but in larger grammars it is useful to adopt a convention for the same purpose, saying that if a "defining" occurrence and an "applied" occurrence of an attribute occur in the same rule, then the two must have the same value unless the contrary is stated.

Second, the affix grammar contained predicate-procedures which just "failed" somehow if conditions were not met, with no indication of what that meant. Here, by contrast, a new attribute ERR() has been given to all non-terminals, and its value is defined at every node (this would, among other things, indicate exactly the location of the error). But we still need to add an additional understanding to be complete, to the effect that a structural description is well-formed just in case (i) all its branchings are correct according to the syntactic rules, and (ii) all the relations among the attributes of its nodes are correct according to the semantic rules, and (iii) the value of the attribute ERR(program) at the root is false.

(It is just such attributes as ERR which can go mostly unmentioned

if the above convention on unwritten rules is adopted.) Clearly, other conventions about errors which would be more like those of the affix grammar could be used.

Finally, the convention of calling the context-free rules the "syntactic rules" and of calling the conditions on attribute values the "semantic rules" is purely an historical artifact and new nomenclature can be introduced. The same distinction has been so long used in natural language systems, however, where the category symbols of "the syntax" are augmented with "semantic features" and tests, that there is not much risk of confusion in the present context.

#### 5.4 An Experiment with Three Notations

Although there is no question of the utility of grammars with structured vocabulary in defining syntax, there are many practical questions about how to go about writing and grasping a grammar large enough to be a comprehensive grammar for a natural language. Restricted van Wijngaarden grammars, in a sense, only work with "inherited" attributes; all information lower in a structural description is imposed by higher levels. Knuth (1968) points out that alternatively it is always possible to use only "synthesized" attributes; the entire form of the tree can be encoded into attributes of successively higher nodes, and then any function of that attribute computed at the root. But the claim is that an interplay of inherited and synthesized definitions is more natural, and easier for people to think about.

It is by no means clear whether or not this is true, so as an aid in evaluating the claim we present here three definitions of the same language, this time a language more closely related to the non-context-free phenomena of natural languages.

Natural language examples tend to be very large relative to what they reveal, so we once again consider a language abstracted from natural language studies so as to be able to write revealing grammars in reasonable space. This sample draws on material familiar to linguists, and so **should** give the flavor of the enterprise. Such abstract structures as are used here are not envisioned as playing a part in any machine translation

system directly, although analogous problems arise in machine translation.

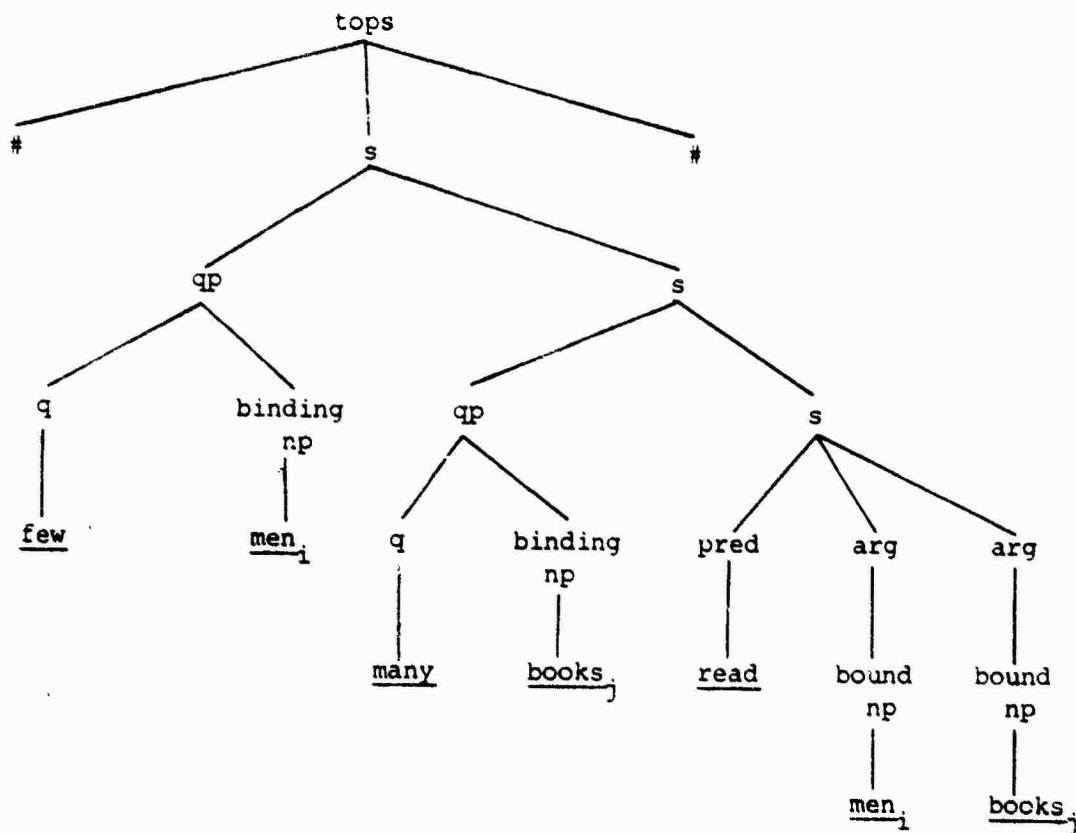
The problem is that of specifying the set of derivation-initial phrase-markers for a grammar of the sort associated with "Generative Semantics". In such a "semantically-based" grammar, one needs something like a base component to initiate derivations; as McCawley says, "the closest analogue to the base component of Aspects is a set of rules specifying what semantic representations are well-formed" (McCawley 1973). Although no such set of rules has ever been made explicit, in general we are talking about structures in which quantified noun phrases originate in higher sentences, and in which clauses of the ordinary type contain only references to the indexes of these noun phrases.

A tiny context-free grammar for such structures could be:

1. tops  $\rightarrow$  # s #
2. s  $\rightarrow$  qp s
3. qp  $\rightarrow$  q binding-np
4. s  $\rightarrow$  pred arg arg
5. arg  $\rightarrow$  s
6. arg  $\rightarrow$  bound-np

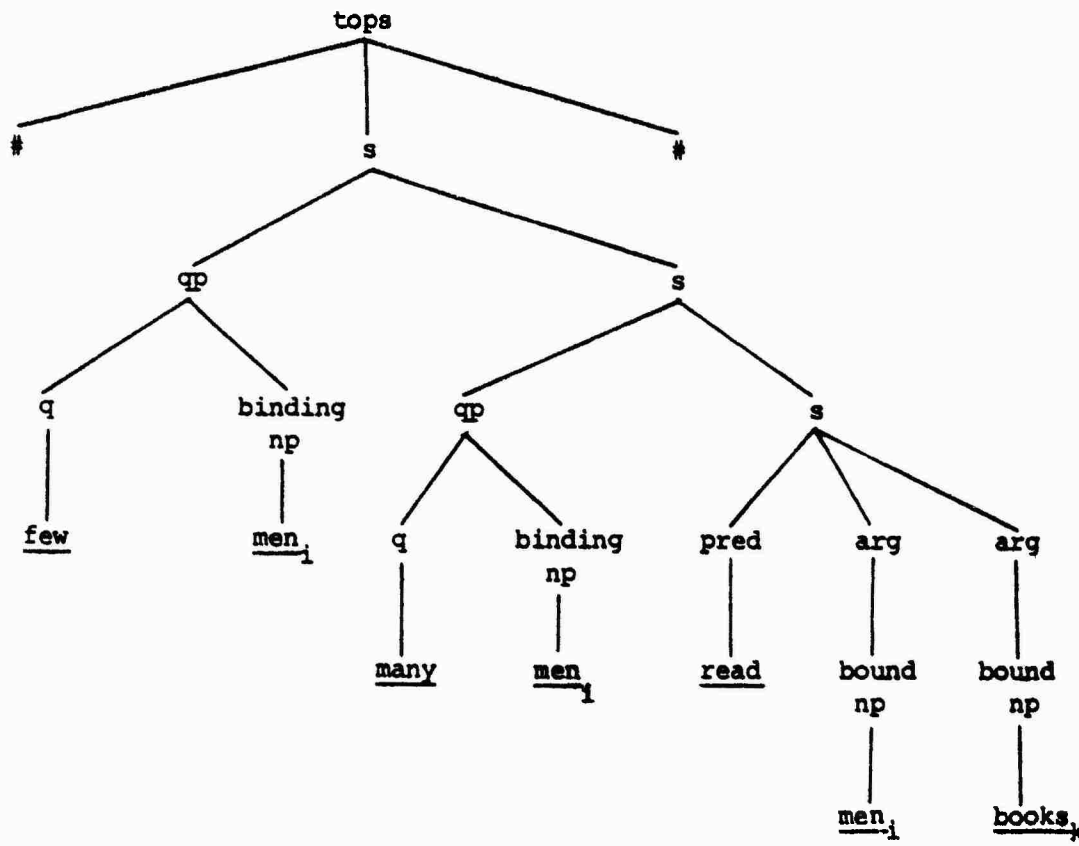
We will imagine that lexical terminals appear under "q" (quantifier), "pred" (predicate), "binding-np", and "bound-np" by some auxiliary non-grammatical process. This is perhaps not the best grammar for such structures, but it has the advantage of being very simple; still more interesting, though longer, examples could be constructed around the structures of (McCawley 1973, pp. 79ff.).

A sample derivation in this grammar might be (with lexical terminals inserted):



which could possibly underlie "Few men read many books."

But the context-free base rules are not adequate to define the structures we want, since they could equally well produce the trees such as the following (assuming that the lexical insertion process is not smart, but merely randomly inserts lexical nouns):



This tree has two binding occurrences for men<sub>i</sub>; that is not possible, so the outer men<sub>i</sub> would be a vacuous quantifier. Books<sub>k</sub> appears free in this expression, but it should be bound; it lacks a defining quantifier expression. Thus, this tree could not underlie any well-formed expression since it is incoherently formed.

So we should add to our context-free grammar a requirement that every bound-np must be the same as one and only one binding-np above it, must be the same index as the lowest instance of the same name above it, and there must be no excess binding-np's.

Since these linguistic trees were designed to be parallel in structure to logical formulae, it is not surprising that these requirements are the same as the requirements for "coherent quantification" in the quantificational expressions used by logicians. But the fact is that such languages are not context-free (have no context-free grammars). (This

fact as applied to logical formulae has been discussed by Janet Dean Fodor 1970; McCawley has discussed the difficulty in specifying a base grammar, but without giving this natural linguistic characterization of it.)

Obviously this logical language is related to the one discussed in the preceding three sections, but it is more complicated in two ways. First, the same "name" may be introduced several times with different meanings, so long as each layer of binding and bound occurrences meets all the tests, and so the syntax must sort out such multiple uses. Second, vacuous binding occurrences are not permitted, so it is necessary to impose the requirement that every name (every binding use of a name) has a bound use somewhere else.

A van Wijngaarden grammar for such a language is extremely straightforward to set down, although it is rather tricky to read and understand. The important part is the first six hyper-rules (corresponding to the six context-free rules), which have predicate symbols that require that every sentence has a properly-formed table consisting of "layers" of bindings which is consistent with the bindings actually present, that a parallel table of uses also contain all bindings, and that bound variables should be the only item used in their tables and should be identified in the proper scope of the bindings table.

The characterization of well-formed tables of names and binding occurrences also turns out to be straightforward, if tedious, in purely grammatical terms. A well-formed table is a set of pairs (TEXT, IDENT) divided into layers by the punctuation mark "new". All names must be distinct in every layer. The same name TEXT may occur in different layers, but every unique name IDENT must be unique in the table. Observe that the meta-rules generate all the possible TABLES there are for the hyper-rules, and it is only the predicate tests which restrict the tables to be well-formed.

Meta-rules:

- m1. ALPHA ::→ a | b | c | ... | x | y | z
- m2. STRING ::→ ALPHA | STRING ALPHA



- m3. EMPTY ::→ λ
- m4. STRINGETY ::→ STRING | EMPTY
- m5. LETTER ::→ letter ALPHA
- m6. TEXT ::→ LETTER | TEXT LETTER
- m7. NUMBER ::→ i | ii | iii | ... | iiiiiiiii
- m8. DIGIT ::→ digit NUMBER
- m9. IDENT ::→ DIGIT | IDENT DIGIT
- m10. DEF ::→ ( TEXT , IDENT )
- m11. DEFS ::→ DEF | DEFS DEF
- m12. DEFSETY ::→ DEFS | EMPTY
- m13. NEW ::→ new
- m14. TABLE ::→ new DEFS | TABLE new DEFS
- m15. BOUND ::→ TABLE
- m16. USED ::→ TABLE
- m17. ALPHABET ::→ abcdefghijklmnopqrstuvwxyz

Basic Hyper-rules:

- h1. ⟨ tops ⟩ :→ ⟨ # symbol ⟩ ⟨ s BOUND USED ⟩ ⟨ # symbol ⟩  
                   ⟨ where BOUND is NEW ⟩  
                   ⟨ where USED differs from NEW ⟩
- h2. ⟨ s BOUND USED ⟩ :→ ⟨ qp DEFS ⟩ ⟨ s BOUND NEW DEFS USED ⟩  
                   ⟨ where BOUND NEW DEFS is a well-formed table ⟩  
                   ⟨ where BOUND NEW DEFS is a subset of USED ⟩
- h3. ⟨ qp DEF ⟩ :→ ⟨ q ⟩ ⟨ binding-np DEF ⟩

- h4.  $\langle s \text{ BOUND USED1} \rangle : \rightarrow \langle \text{pred} \rangle \langle \text{arg BOUND USED2} \rangle \langle \text{arg BOUND USED3} \rangle$   
 $\langle \text{where USED1 is union of USED2 and USED3} \rangle$
- h5.  $\langle \text{arg BOUND USED} \rangle : \rightarrow \langle s \text{ BOUND USED} \rangle$
- h6.  $\langle \text{arg BOUND USED} \rangle : \rightarrow \langle \text{bound-np DEF} \rangle$   
 $\langle \text{where DEF is identified in BOUND} \rangle$   
 $\langle \text{where NEW DEF is USED} \rangle$

Hyper-rules to expand predicate hyper-symbols

- h7.  $\langle \text{where TABLE new DEFSETY is a well-formed table} \rangle : \rightarrow$   
 $\langle \text{where TABLE is a well-formed table} \rangle$   
 $\langle \text{where DEFS is a well-formed layer} \rangle$   
 $\langle \text{where DEFS does not confuse TABLE} \rangle$
- h8.  $\langle \text{where DEFSETY (TEXT,IDENT) is a well-formed layer} \rangle : \rightarrow$   
 $\langle \text{where (TEXT, is not in DEFSETY)} \rangle$   
 $\langle \text{where ,IDENT) is not in DEFSETY} \rangle$   
 $\langle \text{where DEFSETY is a well-formed layer} \rangle$   
 $| \langle \text{where DEFSETY is EMPTY} \rangle$
- h9.  $\langle \text{where (TEXT1, is not in (TEXT2,IDENT) DEFSETY)} \rangle : \rightarrow$   
 $\langle \text{where TEXT1 differs from TEXT2} \rangle$   
 $\langle \text{where (TEXT1, is not in DEFSETY)} \rangle$   
 $| \langle \text{where TEXT1 differs from TEXT2} \rangle$   
 $\langle \text{where DEFSETY is EMPTY} \rangle$
- h10.  $\langle \text{where ,IDENT1) is not in (TEXT,IDENT2) DEFSETY} \rangle : \rightarrow$   
 $\langle \text{where IDENT1 differs from IDENT2} \rangle$   
 $\langle \text{where ,IDENT1) is not in DEFSETY} \rangle$   
 $| \langle \text{where IDENT1 differs from IDENT2} \rangle$   
 $\langle \text{where DEFSETY is EMPTY} \rangle$

- h11.  $\langle$ where DEFS does not confuse TABLE new DEFSETY $\rangle : \rightarrow$   
        $\langle$ where DEFS does not confuse TABLE $\rangle$   
        $\langle$ where DEFS does not confuse DEFSETY $\rangle$
- h12.  $\langle$ where DEFSETY (TEXT,IDENT) does not confuse DEFS $\rangle : \rightarrow$   
        $\langle$ where ,IDENT) is not in DEFS $\rangle$   
        $\langle$ where DEFSETY does not confuse DEFS $\rangle$
- h13.  $\langle$ where DEF is identified in TABLE new DEFSETY $\rangle : \rightarrow$   
        $\langle$ where DEF resides in DEFSETY $\rangle$   
       |  $\langle$ where DEF is independent of DEFSETY $\rangle$   
        $\langle$ where DEF resides in TABLE $\rangle$
- h14.  $\langle$ where DEF1 resides in DEFS DEF2 $\rangle : \rightarrow$   
        $\langle$ where DEF1 resides in DEFS $\rangle$   
       |  $\langle$ where DEF1 resides in DEF2 $\rangle$
- h15.  $\langle$ where (TEXT1,IDENT1) resides in (TEXT2,IDENT2) $\rangle : \rightarrow$   
        $\langle$ where TEXT1 is TEXT2 $\rangle$   
        $\langle$ where IDENT1 is IDENT2 $\rangle$
- h16.  $\langle$ where USED1 is union of USED2 and USED3 $\rangle : \rightarrow$   
        $\langle$ where USED1 is setequal to USED2 USED3 $\rangle$
- h17.  $\langle$ where USED is setequal to BOUND $\rangle : \rightarrow$   
        $\langle$ where USED is subset of BOUND $\rangle$   
        $\langle$ where BOUND is subset of USED $\rangle$
- h18.  $\langle$ where TABLE1 new DEFSETY is subset of TABLE2 $\rangle : \rightarrow$   
        $\langle$ where TABLE1 is subset of TABLE2 $\rangle$   
        $\langle$ where DEFSETY is subset of TABLE2 $\rangle$

- h19.  $\langle$ where DEFSETY DEF is subset of TABLE $\rangle \rightarrow$   
        $\langle$ where DEF is identified in TABLE $\rangle$   
        $\langle$ where DEFSETY is subset of TABLE $\rangle$
- h20.  $\langle$ where EMPTY is subset of TABLE $\rangle \rightarrow$   
        $\langle$ EMPTY $\rangle$
- h21.  $\langle$ where STRINGETY1 ALPHA1 differs from STRINGETY2 ALPHA2 $\rangle \rightarrow$   
        $\langle$ where STRINGETY1 differs from STRINGETY2 $\rangle$   
       |  $\langle$ where ALPHA1 precedes ALPHA2 in ALPHABET $\rangle$   
       |  $\langle$ where ALPHA2 precedes ALPHA1 in ALPHABET $\rangle$
- h22.  $\langle$ where STRING differs from EMPTY $\rangle \rightarrow$   
        $\langle$ EMPTY $\rangle$
- h23.  $\langle$ where EMPTY differs from STRING $\rangle \rightarrow$   
        $\langle$ EMPTY $\rangle$
- h24.  $\langle$ where ALPHA1 precedes ALPHA2 in STRINGETY1 ALPHA1  
       STRINGETY2 ALPHA2 STRINGETY3 $\rangle \rightarrow$   
        $\langle$ EMPTY $\rangle$
- h25.  $\langle$ where STRINGETY is STRINGETY $\rangle \rightarrow$   
        $\langle$ EMPTY $\rangle$

An affix grammar for the same language is considerably more detailed (in the part corresponding to the first six hyper-rules, naturally, since there is nothing to correspond to the rest of them). It is no longer possible to just take the lofty view that we did in the van Wijngaarden grammar that every binding variable must be used and every variable used must be bound; now we must devise an explicit arrangement whereby a list of binding occurrences is passed down, while both lists of binders and

uses are passed up to be compared at the root.

Affix grammar rules:

1.  $\langle \text{tops} \rangle \rightarrow \langle \# \text{ symbol} \rangle \langle s \downarrow \text{BOUND} \uparrow \text{USED} \uparrow \text{INTR} \rangle \langle \# \text{ symbol} \rangle$   
 $\langle \text{assignset}(\{\}) \text{ returns}(\text{BOUND}) \rangle$   
 $\langle \text{checkuse}(\text{INTR}, \text{USED}) \rangle$
2.  $\langle s \downarrow \text{BOUND1} \uparrow \text{USED} \uparrow \text{INTR1} \rangle \rightarrow \langle \text{qp} \uparrow \text{INTR2} \uparrow \text{BDING} \rangle$   
 $\langle s \downarrow \text{BOUND3} \uparrow \text{USED} \uparrow \text{INTR3} \rangle$   
 $\langle \text{assignset}(\text{INTR2} \cup \text{INTR3}) \text{ returns}(\text{INTR1}) \rangle$   
 $\langle \text{overridepairs}(\text{BOUND1}, \text{BDING}) \text{ returns}(\text{BOUND3}) \rangle$
3.  $\langle \text{qp} \uparrow \text{INTR} \uparrow \text{BDING} \rangle \rightarrow \langle q \rangle \langle \text{binding-np} \uparrow \text{TEXT} \uparrow \text{IDENT} \rangle$   
 $\langle \text{assignset}(\{\text{IDENT}\}) \text{ returns}(\text{INTR}) \rangle$   
 $\langle \text{overridepairs}(\{\text{TEXT}, \text{IDENT}\}, \{\}) \text{ returns}(\text{BDING}) \rangle$
4.  $\langle s \downarrow \text{BOUND} \uparrow \text{USED1} \uparrow \text{INTR1} \rangle \rightarrow \langle \text{pred} \rangle \langle \text{arg} \downarrow \text{BOUND} \uparrow \text{USED2} \uparrow \text{INTR2} \rangle$   
 $\langle \text{arg} \downarrow \text{BOUND} \uparrow \text{USED3} \uparrow \text{INTR3} \rangle$   
 $\langle \text{assignset}(\text{USED2} \cup \text{USED3}) \text{ returns}(\text{USED1}) \rangle$   
 $\langle \text{assignset}(\text{INTR2} \cup \text{INTR3}) \text{ returns}(\text{INTR1}) \rangle$
5.  $\langle \text{arg} \downarrow \text{BOUND} \uparrow \text{USED} \uparrow \text{INTR} \rangle \rightarrow \langle s \downarrow \text{BOUND} \uparrow \text{USED} \uparrow \text{INTR} \rangle$
6.  $\langle \text{arg} \downarrow \text{BOUND} \uparrow \text{USED} \uparrow \text{INTR} \rangle \rightarrow \langle \text{bound-np} \uparrow \text{TEXT} \rangle$   
 $\langle \text{assignset}(\{\}) \text{ returns}(\text{INTR}) \rangle$   
 $\langle \text{identify}(\text{TEXT}, \text{BOUND}) \text{ returns}(\text{USED}) \rangle$

Predicates:

<u>name</u>	<u>input parameters</u>	<u>output parameters</u>	<u>function</u>
assignset	1. SET1	2. SET2	SET1 := SET2
checkuse	1. SET1 2. SET2	--	<u>if</u> (SET1 \ SET2) = {} <u>then</u> <u>succeed</u> <u>else</u> <u>fail</u> <u>fi</u>
overridepairs	1. PAIRS1 2. PAIRS2	3. PAIRS3	PAIRS3 := (PAIRS1 overridden by PAIRS2)
identify	1. TEXT 2. PAIRS	3. IDENT	<u>if</u> (TEXT is first member of a pair in PAIRS) <u>then</u> IDENT := (corresponding second member) <u>else</u> <u>fail</u> <u>fi</u>

And finally, an attribute grammar will be much like an affix grammar in its strategy.

Attributes:

Inherited

<u>name</u>	<u>for symbols</u>	<u>type of value</u>	<u>comment</u>
↓BOUND	s, arg	function	specifies unique ident for any text, relative to context

Synthesized

<u>name</u>	<u>for symbols</u>	<u>type of value</u>	<u>comment</u>
↑UNUSED	tops	set of idents	vacuous bindings
↑USED	s, arg	set of idents	binding variables used in this subtree
↑INTR	s, arg, qp	set of idents	binding variables intro- duced in this subtree
↑BDING	qp	function	specifies ident of a single binding text
↑TEXT	binding-np, bound-np	string	text of variable name
↑IDENT	binding-np	unique number in tree	identification of binding np

Convention:

A tree is well-formed only if the value of ↑UNUSED at the root  
is the null set.

Attribute grammar rules:

1. tops  $\rightarrow$  # s #

$$\uparrow\text{UNUSED}(\text{tops}) = \uparrow\text{INTR}(s) \setminus \uparrow\text{USED}(s)$$

$$\downarrow\text{BOUND}(s) = \{\}$$

2.  $s_1 \rightarrow$  qp  $s_2$

$$\uparrow\text{USED}(s_1) = \uparrow\text{USED}(s_2)$$

$$\uparrow\text{INTR}(s_1) = \uparrow\text{INTR}(s_2) \cup \uparrow\text{INTR}(\text{qp})$$

$$\downarrow\text{BOUND}(s_2) = \downarrow\text{BOUND}(s_1) \cup \uparrow\text{BDING}(\text{qp})$$

3. qp  $\rightarrow$  q binding-np

$$\uparrow\text{INTR}(\text{qp}) = \uparrow\text{IDENT}(\text{binding-np})$$

$$\uparrow\text{BDING}(\text{qp}) = \{\uparrow\text{TEXT}(\text{binding-np}),$$

$$\uparrow\text{IDENT}(\text{binding-np})\}$$

4. s  $\rightarrow$  pred arg<sub>1</sub> arg<sub>2</sub>

$$\uparrow\text{USED}(s) = \uparrow\text{USED}(\text{arg}_1) \cup \uparrow\text{USED}(\text{arg}_2)$$

$$\uparrow\text{INTR}(s) = \uparrow\text{INTR}(\text{arg}_1) \cup \uparrow\text{INTR}(\text{arg}_2)$$

$$\downarrow\text{BOUND}(\text{arg}_1) = \downarrow\text{BOUND}(s)$$

$$\downarrow\text{BOUND}(\text{arg}_2) = \downarrow\text{BOUND}(s)$$

5. arg  $\rightarrow$  s

$$\uparrow\text{USED}(\text{arg}) = \uparrow\text{USED}(s)$$

$$\uparrow\text{INTR}(\text{arg}) = \uparrow\text{INTR}(s)$$

$$\downarrow\text{BOUND}(s) = \downarrow\text{BOUND}(\text{arg})$$

6. arg  $\rightarrow$  bound-np

$$\uparrow\text{USED}(\text{arg}) = \downarrow\text{BOUND}(\text{arg}) (\uparrow\text{TEXT}(\text{bound-np}))$$

$$\uparrow\text{INTR}(\text{arg}) = \{\}$$



A comparison of these three definitions is somewhat difficult, all the more so because there are choices of style made in connection with each which are arbitrary and which could be changed. But it does seem that the van Wijngaarden grammar may offer "too smooth a surface," and conceal too much behind its meta-symbols and the predicates to restrict their productions. The greater length of the van Wijngaarden grammar should not be held against it -- quite the contrary, since the excess comes entirely in the expansion of the predicate hyper-symbols, in which every jot and tittle of convention is formalized completely in syntactic terms, whereas the other definitions both rely on many "understood" notions from mathematics and programming languages. However, it seems hard to avoid some feeling of a Turing machine simulation in the strings of the production symbols in a van Wijngaarden grammar, and it is probably helpful to relax a little as the others do and accept sets and trees (say) as primitives, with operations on them directly.

The affix grammar seems to conceal just the wrong part of the grammar in its separately-defined predicates. The long lists of (mostly redundant) affixes are written out in full, but the actions are hidden in separate procedures which define relations among the affixes. The reverse arrangement of Knuth's attribute grammars seems preferable; and it is possible to read (and perhaps necessary to write) the attribute grammar in purely a "declarative" frame of mind, treating the semantic rules as static conditions on the well-formedness of feature sets.

A practical notation for van Wijngaarden grammars, based on this very limited experiment, might be to write rules with conditions in Knuth's form. It would doubtless in this case be wise to add some analogue of the Uniform Replacement Convention, as we discussed before. Observe, for instance, how rule 5 in the van Wijngaarden grammar and in the affix grammar are exquisitely simple, whereas rule 5 in the attribute grammar contains (predictable) conditions to "pass along" every attribute of the symbols in the rule.

### 5.5 The Parsing Problem for Restricted Grammars with Structured Vocabulary

The parsing problem for general van Wijngaarden grammars is the problem of parsing type-0 languages, but for restricted languages such

as we have explored in the last section, the basic parsing strategy is clear: these are just context-free grammars plus "a little more." As a result, it is not difficult to see how to proceed.

For basic context-free parsing, there is one algorithm of greatest importance, or one family of algorithms: this is the "nodal spans" algorithm of John Cocke (Cocke and Schwartz 1970) and its extension with the predictive elements of Knuth's LR(k) technique (Knuth 1965) by Jay Earley (Earley 1968, 1970). This "chart parser" technique (Benson 1969, Woods 1975) is the most flexible and general of the parsing algorithms, with excellent speed when implemented correctly. This algorithm has been employed in the Quince system at Berkeley, and in its predecessor Syntax Analysis System, since 1967 -- perhaps uniquely, since a recent survey of the field (Grishman 1976) remarks that these "algorithms have, to the best of our knowledge, not yet been used in natural language parsing."

The only remaining question, then, is how to handle the "little more" of the features.

If one is given the restrictions customarily placed on affix grammars (not detailed here) (Watt 1977), then it is possible to check all affixes in a single pass over a parse tree, and moreover this can be interleaved with parsing itself in a deterministic parsing algorithm (such as LR(k) or LL(k)). It is not clear, however, that a grammar for any natural language could be written conforming to these restrictions.

For completely unrestricted attribute grammars, Fang (1972) wrote a non-deterministic parsing system, of unsatisfactory efficiency. For restricted attribute grammars of the sort considered here, however, it is not clear that such generality is needed, either.

The best choice seems to be the procedure of (Bochmann 1976), in which a number of left-to-right passes over a parse tree are used to evaluate all attributes. In practical cases it appears that a very few passes would be sufficient, unless the definition of attributes is circular, because the depth of nesting in grammars for natural languages is not great. (Bochmann includes an algorithm for determining the maximum number of passes necessary for an attribute grammar, and such algorithms can be of practical use with restricted grammars, in spite

of the demonstration (Jazayeri, Ogden, and Rounds 1975) of the intrinsically exponential complexity of the circularity problem for attribute grammars.)

The problem is well-understood in the case of attribute grammars applied to programming languages (see, e.g., Lewis, Rosenkrantz, and Stearns 1976). A chief difference between these grammars and grammars used for natural languages is that programming language grammars are typically unambiguous (or nearly so) even without considering attributes. The ambiguity of natural language grammars without considering attributes is intended to be high -- that is part of what the attributes are for -- which suggests that as much as possible of the attribute-processing should be interfactored with parsing to eliminate false partial trees as early as possible. This requirement could well mean that a determination should be made by a grammar pre-processor as to which attributes have the "one-pass" property, and which must be computed over completed derivation trees, with different strategies used for the two kinds of attributes. This determination applied to an attribute grammar would be straightforward as compared to a compiler for Woods's ATNs (Burton and Woods 1976) because of the more regular structure of the attribute grammar.

At the level of implementation tactics, as opposed to strategy, there are a number of challenging problems in using grammars with structured vocabulary. Some of the techniques have been worked out in connection with the current Quince parser in use at Berkeley, and references to such of this work as has been published will be found in the final section.

#### 6. The Quince System and Grammars with Structured Vocabulary

It would be more surprising than not if grammars with structured vocabulary played no part in the existing Quince parsing approach and in our plans for future progress -- especially is this so in light of the argument made in the preceding sections that structured vocabulary is important (however disguised) in the grammars or procedures of all current natural language systems. We have, however, been perhaps more systematic than most research groups in our past development of this topic.

## 6.1 Previous Uses of Structured Vocabulary at POLA

How to utilize a grammar of Chinese with structured vocabulary has been a topic of research at the Project on Linguistic Analysis since at least 1970. Prior to that time machine translation research at Berkeley (going back to Sidney Lamb's work on Russian in the early 1961's) had made use of grammars with symbols which were systematically related in the minds of the grammarians, but this relatedness was not exploited in the computer systems. By the early 1970's, a succession of grammar-writers had introduced several different and in part incompatible systems of structuring the vocabulary of the grammars.

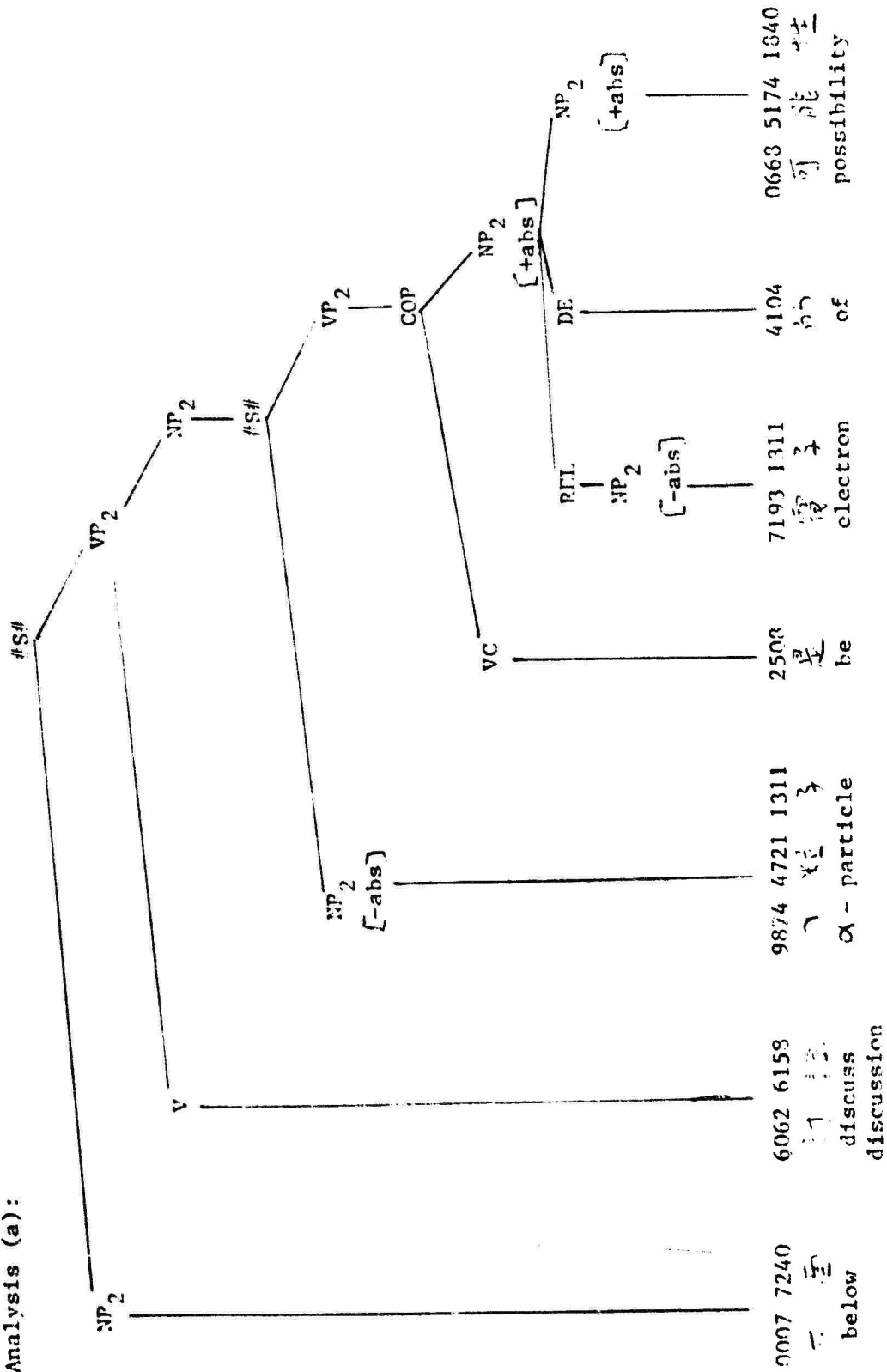
Accordingly, research was begun on how a set of features should be used with the existing grammar of Chinese. This eventually led to a project (described at length in Wang and Chan 1974) to write a "core grammar" of basic syntactic categories, augmented with a set of features. In support of this project a reclassification of all syntactic categories was undertaken, and a program was written which translated between the basic category symbols plus features, and a second "extended" set of category symbols which included the feature information.

Thus, since about 1972 Chinese grammars at POLA have been maintained using indexes of "core grammar rules" followed by their feature instantiations, so that a grammarian did not need to understand over 3000 rules directly.

Studies of features and simplifications of the grammar continued, and (Wang and Chan 1975) reports a wide variety of Chinese examples with the features needed for correct syntactic analyses in terms of the basic categories. For example, the feature "abstractness" is found to be essential for analysing copula sentences. The subject noun phrase must agree with the object noun phrase with respect to this feature in a copula sentence. An actual example from our Physics texts is extracted below for illustration. Two syntactic analyses are possible for this sentence, given as (a) and (b) below, but only (b) gives the correct analysis of the structure of the sentence.

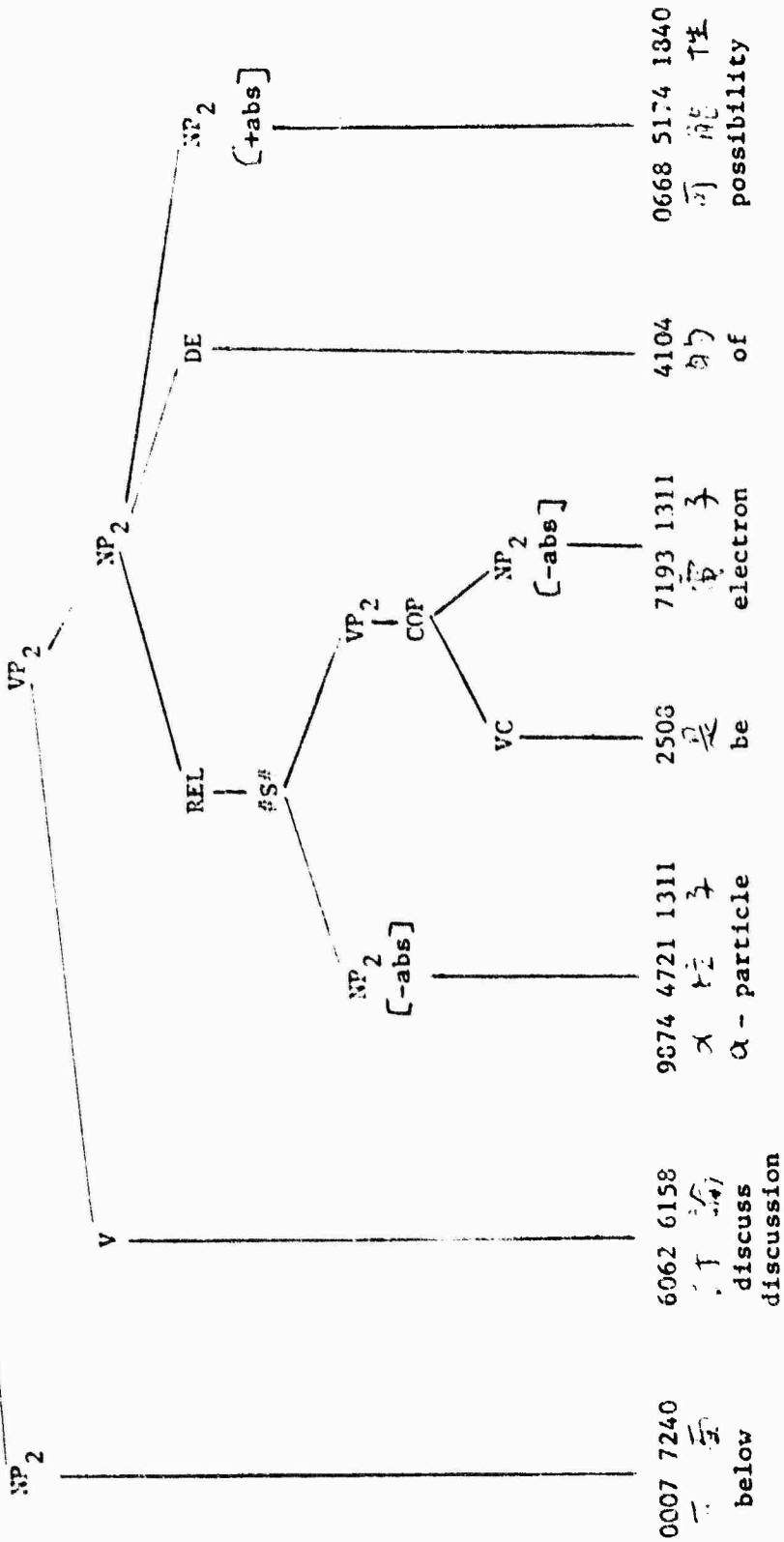
Example: from Physics I-2, 1972-10, p. 61, 5th parag.

Analysis (a):



'below is discussed -particle is the possibility of an electron'

Analysis (b)



'below is discussed the possibility of alpha-particle being an electron'

A mechanism for "feature parsing" was designed into the Quince parsing system, and provisions for it were made in the initial implementation of the parser. The practical problem of giving feature encodings to a complete Chinese dictionary was intractable with the staff available, and so a smaller dictionary was constructed with which to work from selected physics texts for feature encoding.

At the same time, other distinct changes in the grammar and in the method of applying "interlingual transformations" were made for the Quince system, so that a version of Knuth's attribute grammars could be employed, with the interlingual specification of structural change carried as attributes imposed by the Chinese grammar rules.

We did not then recognize the close connections between these two topics, although the fact that they were being worked on by the same staff members should have assured that they would converge eventually, had development not come to a temporary halt shortly thereafter.

## 6.2 Research Areas for Future Study

All work to date indicates that the primary research problem in the area of parsing for Chinese-English machine translation lies in how to define/describe natural languages in general, and how to define/describe Chinese and English in particular. This linguistic analysis is the really difficult task, compared to which the implementation of programs to carry out the analysis is straightforward.

Therefore, the next task is to make some trials of recasting our Chinese grammar into various notations for grammars with structured vocabulary, to see what format appears to give maximal insight in use. At present, based upon the earlier experiences described above, it seems that a grammar notation based on Knuth's attribute grammars is the most promising. Without attempting to formulate a realistically-large fragment of a grammar for a natural language, there is no way to be sure about some of the minor (though perhaps crucial) details, because prior systematic uses of these formalisms have been in connection with programming languages.

The strong tradition of using structured vocabulary, however informally, in prior linguistic description and in computational projects, makes us quite certain that systematic attention to creating a grammar

in this general form would constitute an advance over previous work.

There are two specific areas of uncertainty which we wish to begin immediately to clear up:

(1) What is the trading relation between encoding information as category symbols, versus features? Clearly, any grammar with a finite number of rules (any such hyper-grammar) can be encoded directly by multiplying out category symbols and rules (no matter that this may be wildly impractical). At the other extreme, one can imagine a grammar containing a single category symbol ("NODE") and one rule for combining each length string of such symbols as a single symbol, with all the information carried in attributes. Linguistic tradition suggests an intuitive division of grammatical information in the two classes, but can a clearer description be formulated?

(2) We have not attempted to exploit the "internal structure" given by the meta-grammar of a van Wijngaarden grammar, and in passing to an attribute representation one passes naturally to sets, functions, etc. as attributes. But is there a way to exploit the fact that a tree is given by a meta-grammar for every meta-symbol replaced, without getting confused by the tree-manipulation systems? And if so, would it be advantageous to restrict attributes to being tree-structured (ignoring as much structure as desired in particular cases)?

Doubtless other similar questions will suggest themselves as work continues.

While such research goes forward on the grammatical side, there are also many questions to explore about a computer parsing procedure for such grammars. The fact that such good strategies exist for parsing the restricted affix grammars suggests that a related procedure could be devised for restricted attribute grammars which would work similarly (interfactoring parsing and attribute value calculation) wherever possible, and only do more work (in the form of post-parse processing of attributes in one or more additional passes over the tree) when necessary. This could be of importance in making the full exploitation of such grammars possible, although the techniques reported by us in the past would themselves certainly be adequate to permit the most important uses of grammars with structured vocabulary to be incorporated directly into the Quince system.



Acknowledgement. The initial research at POLA into grammars with structured vocabulary was begun by Herbert Doughty, who also devised plans for an ingenious and efficient implementation of features, and played a major role along with the author in elaborating proposals to incorporate this work into the Quince system.

## References

(Agafonov 1976)

Agafonov, V. N., "On Attribute Grammars", Antoni Mazurkiewicz (ed.), Mathematical Foundations of Computer Science 1976, (Proceedings, 5th symposium, Gdansk, September 6-10, 1976), Lecture Notes in Computer Science No. 45, Berlin: Springer 1976, pp. 169-72.

(Aho 1968)

Aho, Alfred V., "Indexed Grammars -- An Extension of Context-Free Grammars," Journal A.C.M. 15, 1958, pp. 647-71.

(Aho and Ullman 1973)

Aho, Alfred V., and Ullman, Jeffrey D., The Theory of Parsing, Translation, and Compiling, 2 vols., Englewood Cliffs, N.J.: Prentice-Hall, 1973.

(Bach 1964)

Bach, Emond, "Subcategories in Transformational Grammars", Lunt, H. (ed.), Proceedings of the Ninth International Congress of Linguists, The Hague: Mouton, 1964, pp. 672-78.

(Baker 1970)

Baker, John L., "Some Formal Properties of the Syntax of ALGOL 68," Doctoral dissertation, University of Washington, May 1970.

(Baker 1972)

Baker, John L., "Grammars with Structured Vocabulary: A Model for the ALGOL-68 Definition," Information and Control 20, 1972, pp. 351-95.

(Benson 1969)

Benson, David B., "The Algebra of Derivations and a Semithue Parser," Proceedings of the 24th National Conference, New York: Association for Computing Machinery, 1969, pp. 1-10.

(Bobrow 1964)

Bobrow, Daniel G., "Meteor: A List Interpreter for String Transformations," Berkeley, E.C., and Bobrow, D. G. (eds.), The Programming Language Lisp: Its Operation and Application, Cambridge, Mass.: Information International, Inc., 1964, pp. 161-90.

(Bobrow and Collins 1975)

Bobrow, Daniel G., and Collins, Allan (eds), Representation and Understanding: Studies in Cognitive Science, New York: Academic Press, 1975.

(Bochmann 1976)

Bochmann, Gregor V., "Semantic Evaluation from Left to Right," Comm. A.C.M. 19, 1976, pp. 55-62.

(Book 1974)

Book, Ronald V., "Topics in Formal Language Theory," Aho, A. V. (ed.), Currents in the Theory of Computing. Englewood Cliffs, N.J.: Prentice-Hall, 1973, pp. 1-34.

(Brainerd and Landweber 1974)

Brainerd, Walter S., and Landweber, Lawrence H., Theory of Computation, New York: John Wiley, 1974.

(Burton and Woods 1976)

Burton, R. R., and Woods, William A., "A Compiling System for Augmented Transition Networks", Paper presented at International Conference on Computational Linguistics, Ottawa, Canada, June 1976.

(Chomsky 1951)

Chomsky, Noam, "Morphophonemics of Modern Hebrew," revision of 1951 Masters Thesis, University of Pennsylvania.

(Chomsky 1955/1975)

Chomsky, Noam, The Logical Structure of Linguistic Theory, Unpublished 1955 MS available from MIT Libraries, printed with omissions and revisions, New York: Plenum Press, 1975.

(Chomsky 1956)

Chomsky, Noam, "Three Models for the Description of Language," I.R.E. Transactions on Information Theory, vol IT-2, 1956, pp. 113-24. Reprinted with corrections in Luce, R.D., Bush, R., and Galanter, E., (eds.), Readings in Mathematical Psychology, vol. II, New York: Wiley, 1965.

(Chomsky 1957)

Chomsky, Noam, Syntactic Structures, Janua Linguarum, series minor, no 4, The Hague: Mouton, 1957.

(Chomsky 1959)

Chomsky, Noam, "On Certain Formal Properties of Grammars," Information and Control 2, 1959, pp. 133-67. Reprinted in Luce, R. D., Bush, R., and Galanter, E., (eds.), Readings in Mathematical Psychology, vol II, New York: Wiley, 1965.

(Chomsky 1962)

Chomsky, Noam, "A Transformational Approach to Syntax," Hill, A. A. (ed.), Proceedings of the 1958 Conference on Problems of Analysis in English, Austin, Texas, 1962, pp. 124-48. Reprinted in Fodor, Jerry A., and Katz, Jerrold J. (eds), The Structure of Language: Readings in the Philosophy of Language, Englewood Cliffs, N.J.: Prentice-Hall, 1964.

(Chomsky 1965)

Chomsky, Noam, Aspects of the Theory of Syntax, Cambridge, Mass.: M.I.T. Press, 1965.

(Chomsky 1966)

Chomsky, Noam, "Topics in the Theory of Generative Grammar," Sebeok, Thomas A. (ed.), Current Trends in Linguistics: III, Theoretical Foundations, The Hague: Mouton, 1966, pp. 1-60.

(Chomsky 1970)

Chomsky, Noam, "Remarks on Nominalization," Jacobs, Roderick A., and Rosenbaum, Peter S., (eds.), Readings in English Transformational Grammar, Waltham, Mass.: Ginn & Co. Reprinted in Chomsky, Noam, Studies on Semantics in Generative Grammar, The Hague: Mouton, 1972.

(Chomsky 1977)

Chomsky, Noam, Essays on Form and Interpretation, (Studies in Linguistic Analysis, 2), New York: Elsevier North-Holland, 1977.

(Chomsky and Halle 1968)

Chomsky, Noam, and Halle Morris, The Sound Pattern of English, New York: Harper and Row, 1968.

(Chomsky and Lasnik 1977)

Chomsky, Noam, and Lasnik, Howard, "Filters and Control," Linguistic Inquiry 8, 1977, pp. 425-504.

(Cleaveland and Uzgalis 1977)

Cleaveland, J. Craig, and Uzgalis, Robert C., Grammars for Programming Languages, (Elsevier Computer Science Library, Programming Languages Series No. 4), New York: Elsevier North-Holland, 1977.

(Cocke and Schwartz 1970)

Cocke, John, and Schwartz, J. T., "Programming Languages and Their Compilers," Preliminary Notes, Second Revised Version, Lecture Notes, Courant Institute of Mathematical Sciences, New York University, April 1970.

(Crowe 1972)

Crowe, David, "Generating Parsers for Affix Grammars," Comm. A.C.M. 15, 1972, pp. 728-34.

(de Bakker 1969)

de Bakker, J. W., "Semantics of Programming Languages," Tou, Julius T., (ed.), Advances in Information Systems Science, vol. 2, New York: Plenum Press, 1969, pp. 173-227.

(de Chastellier and Colmerauer 1969)

de Chastellier, Guy, and Colmerauer, Alain, "W-Grammar," Proceedings of the 24th National Conference, New York: Association for Computing Machinery, 1969, pp. 511-18.

(Dijkstra 1972)

Dijkstra, Edsger W., "Notes on Structured Programming," Dahl, O.-J., Dijkstra, E. W., and Hoare, C. A. R., (eds.), Structured Programming, A.P.I.C. Studies in Data Processing No. 8, New York: Academic Press, 1972.

(Donahue 1976)

Donahue, James E., Complementary Definitions of Programming Language Semantics, Lecture Notes in Computer Science No. 42, Berlin: Springer, 1976.

(Dostert and Thompson 1971)

Dostert, B. H., and Thompson, F. B., "How Features Resolve Syntactic Ambiguity," Proceedings of Symposium on Information Storage and Retrieval, University of Maryland, April 1971.

(Dostert and Thompson 1972)

Dostert, B. H., and Thompson, F. B., "Syntactic Analysis in REL English: A Computational Case Grammar," Statistical Methods in Linguistics 8, 1972, pp. 5-38.

(Earley 1968)

Earley, Jay, "An Efficient Context-Free Parsing Algorithm," Ph.D. thesis, Department of Computer Science, Carnegie-Mellon University, 1968.

(Earley 1970)

Earley, Jay, "An Efficient Context-Free Parsing Algorithm," Comm. A.C.M. 13, 1970, pp. 94-102.

(Elson and Pickett 1960)

Elson, Benjamin, and Pickett, Velma B., Beginning Morphology -- Syntax, Santa Ana, California, 1960.

(Fang 1972)

Fang, Isu, "Folds: A Declarative Formal Language Definition System," Ph.D. Thesis, Stanford University, 1972.

(Fidelholtz 1974)

Fidelholtz, James L., "On the Non-Context-Freeness of Natural Languages, with Some Comments on the Competence'Performance Distinction," Indiana University Linguistics Club Mimeo, 1974.

(Fischer 1968)

Fischer, M. J., "Grammars with Macrolike Productions," Ph.D. Thesis, Harvard University, 1968.

(Floyd 1962)

Floyd, Robert W., "On the Non-Existence of a Phrase-Structure Grammar for Algol 60," Comm. A.C.M. 5, 1962, --. 483-4.

(Fodor 1970)

Fodor, Janet Dean, "Formal Linguistics and Formal Logic," Lyons, John, (ed.), New Horizons in Linguistics, Harmondsworth: Penguin Books, 1970, pp. 198-214.

(Fodor and Katz 1964)

Fodor, Jerry A., and Katz, Jerrold J., (eds.), The Structure of Language: Readings in the Philosophy of Language, Englewood Cliffs,

N.J.: Prentice-Hall, 1964.

(Garvin 1966)

Garvin, Paul L., "Some Comments on Algorithm and Grammar in the Automatic Parsing of Natural Languages," Mechanical Translation 9, 1966. Reprinted in Garvin, Paul L., On Machine Translation, The Hague: Mouton, 1972, pp. 43-7.

(Gaskins 1973)

Gaskins, Robert, "Probative Parsing: The Use of Grammars to Resolve or Preserve Knowledge of Indistinct Sequences for Natural Language Recognition," Machine Translation Group, Project on Linguistic Analysis, University of California, Berkeley, December 1973.

(Gaskins and Gould 1972)

Gaskins, Robert, and Gould, Laura, "Snobol4: A Computer Language for the Humanities," Department of Computer Science and Campus Computer Center Report, University of California, Berkeley, 1972.

(Gimpel 1973)

Gimpel, J. F., "A Theory of Discrete Patterns and Their Implementation in Snobol4," Comm. A.C.M. 16, 1973, pp. 91-100.

(Gimpel 1975)

Gimpel, J. F., "Nonlinear Pattern Theory," Acta Informatica 4, 1975, pp. 213-29.

(Gimpel 1976)

Gimpel, J. F., Algorithms in Snobol4, New York: John Wiley, 1976.

(Greibach 1974)

Greibach, Sheila A., "Some Restrictions on W-Grammars," International Journal of Computer and Information Sciences 3, 1974, pp. 289-327.

(Grishman 1976)

Grishman, Ralph, "A Survey of Syntactic Analysis Procedures for Natural Language," American Journal of Computational Linguistics, microfiche 47, 1976.

(Griswold, Poage, and Polonsky 1971)

Griswold, Ralph E., Poage, J. F., and Polonsky, I.P., The Snobol4 Programming Language, 2nd ed., Englewood Cliffs, N.J.: Prentice-Hall, 1971.

(Gross, Halle, and Schützenberger 1973)

Gross, Maurice, Halle, Morris, and Schützenberger, Marcel-Paul, (eds.), The Formal Analysis of Natural Languages; Proceedings of the First International Conference, The Hague: Mouton, 1973.

(Halitsky 1975)

Halitsky, David, "Left-Branch S's and NP's in English: A Bar Notation Analysis," Linguistic Analysis 1, 1975, pp. 279-96.

(Halliday 1961)

Halliday, M. A. K., "Categories of the Theory of Grammar," Word 17, 1961, pp. 241-92.

(Harman 1963)

Harman, Gilbert H., "Generative Grammars without Transformations: A Defense of Phrase Structure," Language 39, 1963, pp. 597-616.

(Harris 1951)

Harris, Zellig S., Methods in Structural Linguistics, Chicago: University of Chicago Press, 1951. Reprinted 1963 as a Phoenix Book with title Structural Linguistics.

(Harris 1962)

Harris, Zellig S., String Analysis of Sentence Structure, The Hague: Mouton, 1962.

(Hockett 1966)

Hockett, Charles F., "Language, Mathematics, and Linguistics," Sebeok, Thomas A. (ed.), Current Trends in Linguistics: III, Theoretical Foundations, The Hague: Mouton, 1966, pp. 155-304.

(Jackendoff 1974)

Jackendoff, Ray S., "Introduction to the  $\bar{X}$  Convention," Indiana University Linguistics Club Mimeograph, October 1974.

(Jazayeri, Ogden, and Rounds 1975)

Jazayeri, Mehdi, Ogden, William F., and Rounds, William C., "The Intrinsically Exponential Complexity of the Circularity Problem for Attribute Grammars," Comm. A.C.M. 18, 1975, pp. 697-706.

(Joshi and Levy 1977)

Joshi, Aravind K., and Levy, Leon S., "Constraints on Structural Descriptions: Local Transformations," SIAM Journal on Computing, 6



1977, pp. 272-84.

(Kaplan 1973)

Kaplan, R. M., "A General Syntactic Processor," Rustin, Randall, (ed.), Natural Language Processing, New York: Algorithmics Press, 1973.

(Kay 1967)

Kay, Martin, "Experiments with a Powerful Parser," Santa Monica, California: RAND Corporation Memorandum No. RM-5452-PR, October 1967.

(Knuth 1965)

Knuth, Donald E., "On the Translation of Languages from Left to Right," Information and Control 8, 1965, pp. 607-39.

(Knuth 1968)

Knuth, Donald E., "Semantics of Context-Free Languages," Mathematical Systems Theory 2, 1968, pp. 127-45. An important correction appears in Math. Sys. Th. 5, 1971, pp. 95-6.

(Knuth 1971)

Knuth, Donald E., "Examples of Formal Semantics," Engeler, E. (ed.), Symposium on Semantics of Algorithmic Languages, Lecture Notes in Mathematics No. 188, Berlin: Springer, 1971, pp. 212-35.

(Koster 1965)

Koster, C. H. A., "On the Constructuon of Procedures for Generating, Analysing, and Translating Sentences in Natural Languages," Mathematisch Centrum, Amsterdam, Technical Report MR72, February 1965.

(Koster 1971)

Koster, C. H. A., "Affix Grammars," Peck, J. E. L. (ed.), Algol 68 Implementation, Proceedings IFIP Working Conference on Algol 68 Implementation, Munich, July 20-24, 1974, Amsterdam: North-Holland, 1971.

(Koster 1974a)

Koster, C. H. A., "Two-Level Grammars," Bauer, F. L., et al. (eds.), Compiler Construction: An Advanced Course, Lecture Notes in Computer Science No. 21, Berlin: Springer, 1974, pp. 146-56.

(Koster 1974b)

Koster, C. H. A., "Using the CDL Compiler-Compiler," Bauer, F. L., et al. (eds.), Compiler Construction: An Advanced Course, Lecture

Notes in Computer Science No. 21, Berlin: Springer, 1974, pp. 366-426.

(Koster 1975)

Koster, C. H. A., "A Technique for Parsing Ambiguous Languages," Siefkes, Dirk (ed.), Gesellschaft für Informatik 4, Jahrestagung, Lecture Notes in Computer Science No. 26, Berlin: Springer, 1975, pp. 233-46.

(Koutsoudas 1966)

Koutsoudas, Andreas, Writing Transformational Grammars: An introduction, New York: McGraw Hill, 1966.

(Kuno 1963)

Kuno, Susumo, "The Multiple-Part Syntactic Analyser for English," Report No. NSF-9 in Mathematical Linguistics and Automatic Translation, Computation Laboratory, Harvard University.

(Kuroda 1970)

Kuroda, S.-Y., "Remarks on Selectional Restrictions and Presuppositions," Kiefer, Ferenc (ed.), Studies in Syntax and Semantics, Dordrecht: D. Reidel, 1970, pp. 138-67.

(LaLonde 1977)

LaLonde, Wilif R., "Regular Right Part Grammars and Their Parsers," Comm. A.C.M. 20, 1977, pp. 731-41.

(Lamb 1962)

Lamb, Sidney M., Outline of Stratificational Grammar, Berkeley, California, 1962.

(Ledgard 1972)

Ledgard, Henry F., "Embedding Markov Normal Algorithms within the Lambda-Calculus," International Journal of Computer Mathematics Section A 3, 1972, pp. 131-40.

(Ledgard 1974)

Ledgard, Henry F., "Production Systems, or Can We Do Better than BNF?", Comm. A.C.M. 17, 1974, pp. 94-102.

(Lees 1957)

Lees, Robert B., "Review of Syntactic Structures," Language 33, 1957, pp. 375-407. Reprinted in Harman, Gilbert H. (ed.), On Noam Chomsky: Critical Essays, Garden City, N.Y.: Anchor Books, 1974,

pp. 34-79.

(Lewis, Rosenkrantz, and Stearns 1976)

Lewis, P. M., II, Rosenkrantz, D. J., and Stearns, R. E., Compiler Design Theory, Reading, Mass.: Addison-Wesley, 1976.

(McCawley 1968a)

McCawley, James D., "Concerning the Base Component of a Transformational Grammar," Foundations of Language 4, 1968, pp. 243-69.

Reprinted in McCawley, James D., Grammar and Meaning: Papers on Syntactic and Semantic Topics, Taishukan Studies in Modern Linguistics, New York: Academic Press, 1976, pp. 35-58.

(McCawley 1968b)

McCawley, James D., "Review of Sebeok (ed.) Current Trends in Linguistics: III, Theoretical Foundations," Language 44, 1968, pp. 556-93. Reprinted in McCawley, James D., Grammar and Meaning: Papers on Syntactic and Semantic Topics, Taishukan Studies in Modern Linguistics, New York: Academic Press, 1976, pp. 167-205.

(Marcotty, Ledgard, and Bochmann 1976)

Marcotty, Michael, Ledgard, Henry F., and Bochmann, Gregor V., "A Sampler of Formal Definitions (van Wijngaarden Grammars, Ledgard's Production Systems, Vienna Definition Language, and Knuth's Attribute Grammars)," Computing Surveys 8, 1976, pp. 191-276.

(Mazurkiewicz 1969)

Mazurkiewicz, A. W., "A Note on Enumerable Grammars," Information and Control 14, 1969, pp. 555-8.

(Peters and Ritchie 1973)

Peters, P. Stanley, Jr., and Ritchie, Robert W., "Context-Sensitive Immediate Constituent Analysis: Context-Free Languages Revisited," Mathematical Systems Theory 6, 1973, pp. 324-33.

(Petrick 1965)

Petrick, Stanley R. "A Recognition Procedure for Transformational Grammars," Ph.D. thesis, Department of Modern Languages, M.I.T.

(Petrick 1976)

Petrick, Stanley R., "On Natural Language Based Computer Systems," IBM Journal of Research and Development 20, 1976, pp. 314-25.

(Postal 1964a)

Postal, Paul M., "Limitations of Phrase-Structure Grammars," Fodor, Jerry A., and Katz, Jerrold J., (eds.), The Structure of Language: Readings in the Philosophy of Language, Englewood Cliffs, N.J.: Prentice-Hall, 1964, pp. 137-54.

(Postal 1964b)

Postal, Paul M., Constituent Structure: A Study of Contemporary Models of Syntactic Description, Indiana University Research Center in Anthropology, Folklore, and Linguistics Publication No. 30, Bloomington, Indiana: Indiana University, 1964. Also appeared as part III of International Journal of American Linguistics 30, no 1, 1964.

(Sager and Grishman 1975)

Sager, Naomi, and Grishman, Ralph, "The Restriction Language for Computer Grammars of Natural Language," Comm. A.C.M. 18, 1975, pp. 390-400.

(Schachter 1962)

Schachter, Paul, "Review of R. B. Lees 'Grammar of English Nominalizations'," International Journal of American Linguistics 28, 1962, pp. 134-45.

(Schneider 1965)

Schneider, H.-J., "Ein formales Verfahren zur maschinellen Sprachanalyse," Dissertation, Hannover, 1965.

(Schneider 1966)

Schneider, H.-J., "Die Berücksichtigung von Kasus, Genus und anderen Spezifikationen bei formalen Grammatiken natürlicher Sprachen," Elektronische Datenverarbeitung 5, 1966, pp. 245-8.

(Seuren 1968)

Seuren, Peter A. M., "Generative Grammar: Essay of Grammatical Description for a Limited Vocabulary in Dutch, I and II," Braffort, P., and van Scheepen, F. (eds.), Automation in Language Translation and Theorem Proving, Commission of the European Communities (EURATOM), Brussels, 1968.

(Simonet 1977)

Simonet, M., "An Attribute Description of a Subset of Algol 68," Proceedings of the Strathclyde Algol 68 Conference, Glasgow, March 1977, Association for Computing Machinery Special Interest Group on Programming Languages, SIGPLAN Notices 12, No. 6, June 1977, pp. 129-37.

(Sintzoff 1967)

Sintzoff, M., "Existence of a van Wijngaarden Syntax for Every Recursively Enumerable Set," Annales de la Societé Scientifique de Bruxelles 81, 1967, pp. 115-8.

(Sintzoff 1969)

Sintzoff, M., "Grammaires superposees et autres systemes formels," Journées d'Etude sur l'Analyse Syntaxique, Centre d'Automatique, Fontainebleau, 1969.

(Smith 1976)

Smith, Joan M., "Third Annual General Meeting, 1975: Minutes," Association for Literary and Linguistic Computing Bulletin 4, 1976, p. 55.

(Stearns and Lewis 1969)

Stearns, R. E., and Lewis, F. M., "Property Grammars and Table Machines," Information and Control 14, 1969, pp. 524-9.

(van Berckel et al. 1965)

van Berckel, J. A. Th. M., Corstius, H. Brandt, Mokken, R. J., and van Wijngaarden, A., Formal Properties of Newspaper Dutch. Mathematical Centre Tracts No. 12, Amsterdam: Mathematisch Centrum, 1965.

(van der Poel 1971)

van der Poel, W. L., "Some Notes on the History of Algol," de Bakker, J. W., et al. (eds.), MC-25 Informatica Symposium, Mathematical Centre Tracts No. 37, Amsterdam: Mathematisch Centrum, 1971, chapter 7.

(van Wijngaarden, Adriaan 1965)

van Wijngaarden, Adriaan, "Orthogonal Design and Description of a Formal Language," Technical Report MR76, Mathematisch Centrum, Amsterdam, October 1965.

(van Wijngaarden et al. 1969)

van Wijngaarden, A. (ed.), Mailloux, B. J., Peck, J. E. L., and Koster, C. H. A., "Report on the Algorithmic Language Algol 68," Technical Report No. MR101, Mathematisch Centrum, Amsterdam, February 1969. Also in Numerische Mathematik 14, 1969, pp. 79-218.

(van Wijngaarden 1970)

van Wijngaarden, Adriaan, "On the Boundary between Natural and Artificial Languages," Linguaggi nella societa e nella tecnica, Milano: Edizioni di Comunita, 1970, pp. 165-75.

(van Wijngaarden, 1974)

van Wijngaarden, Adriaan, "The Generative Power of Two-Level Grammars," Loeckx, Jacques (ed.), Automata, Languages and Programming, 2nd Colloquium, University of Saarbrücken July 29 - August 2 1974, Lecture Notes in Computer Science No. 14, Berlin: Springer, 1974, pp. 9-16.

(van Wijngaarden et al. 1975)

van Wijngaarden, A., Mailloux, B. J., Peck, J. E. L., Koster, C. H. A., Sintzoff, M., Lindsey, C. H., Meertens, L. G. L. T., and Fisker, R. G., "Revised Report on the Algorithmic Language Algol 68," Acta Informatica 5, 1975, pp. 1-236. Reprinted Springer, Berlin and New York, 1976. Originally appeared as Technical Report No. TR 74-3, Department of Computer Science, University of Alberta (Edmonton), March 1974.

(Wang and Chan 1974)

Wang, William S-Y., and Chan Stephen W., "Development of Chinese-English Machine Translation System," Final Technical Report 01 September 1970 - 30 August 1972, Rome Air Development Command Report No. RADC-TR-74-22, 776813, Rome Air Development Center, Air Force Systems Command, Griffiss Air Force Base, New York, February 1974.

(Wang and Chan 1975)

Wang, William S-Y., and Chan, Stephen W., "Chinese-English Machine Translation System," Final Technical Report 01 September 1972 - 31 August 1974, Rome Air Development Command Report No. RADC-TR-75-109, A011715, Rome Air Development Center, Air Force Systems Command,

Griffiss Air Force Base, New York, April 1975.

(Wang, Chan, and Robyn 1976)

Wang, William S-Y., Chan, Stephen W., and Robyn, Philip, "Chinese-English Machine Translation System," Final Technical Report 01 September 1974 - 31 October 1975, Rome Air Development Command Report No. RADC-TR-76-21, A021969, Rome Air Development Center, Air Force Systems Command, Griffiss Air Force Base, New York, February 1976.

(Watt 1977)

Watt, David Anthony, "The Parsing Problem for Affix Grammars," Acta Informatica 8, 1977, pp. 1-20.

(Wilner 1972)

Wilner, Wayne T., "Formal Semantic Definition Using Synthesized and Inherited Attributes," Rustin, Randall (ed.), Formal Semantics of Programming Languages, Englewood Cliffs, N.J.: Prentice-Hall, 1972, pp. 25-40.

(Winograd 1971)

Winograd, Terry, "Procedures as a Representation for Data in a Computer Program for Understanding Natural Language," Ph.D. thesis, Department of Mathematics, M.I.T. Printed as Project MAC Report No. MAC-TR-34, Project MAC, M.I.T., February 1971.

(Winograd 1975)

Winograd, Terry, "Frame Representations and the Declarative/Procedural Controversy," Bobrow, Daniel G., and Collins, Allan (eds.), Representation and Understanding: Studies in Cognitive Science, New York: Academic Press, 1975, pp. 185-210.

(Woods 1969)

Woods, William A., "Augmented Transition Networks for Natural Language Analysis," Aiken Computation Laboratory Report No. CS-1, Harvard University, December 1969.

(Woods 1970)

Woods, William A., "Transition Network Grammars for Natural Language Analysis," Comm. A.C.M. 13, 1970, pp. 591-606.

(Woods 1975)

Woods, William A., "Syntax, Semantics, and Speech," BBN Report No. 3067, A. I. Report No. 27, Cambridge, Mass.: Bolt Beranek and Newman Inc., April 1975.

(Woods, Kaplan, and Nash-Webber 1972)

Woods, William A., Kaplan, Ronald N., and Nash-Webber, Bonnie, "The Lunar Sciences Natural Language Information System: Final Report," BBN Report No. 2378, Cambridge, Mass.: Bolt Beranek and Newman Inc., 1972.

(Yngve 1960)

Yngve, Victor H., "A Model and an Hypothesis for Language Structure," Proceedings of the American Philosophical Society 104, 1960, pp. 444-66.

(Yngve 1961)

Yngve, Victor H., COMIT Programmers' Reference Manual, Research Laboratory of Electronics and The Computation Center, Massachusetts Institute of Technology, 1961.

(Yngve 1972)

Yngve, Victor H., Computer Programming with COMIT II, Cambridge, Mass.: MIT Press, 1972.



APPENDIX: RULES FOR A FRAGMENT OF CHINESE GRAMMAR

- 0 . < env TF > : --> < rrs TF CF >
- 1 . < rrs TF CF > : --> (<sadv>)(<time TF>) <rs TF CF >
- CF :: --> INTER PASS NEG IMPER
- TF :: --> TENSE ASPECT
- TENSE :: --> PAST | PRES | FUT
- PAST :: --> +past -pres -fut
- PRES :: --> -past +pres - fut
- FUT :: --> -past -pres +fut
- ASPECT :: --> PROG PERF
- PROG :: --> +prog | - prog | EMPTY
- PERF :: --> +perf | -perf | EMPTY

Notes::

env	=	environment
TF	=	Time Features
rrs	=	root root sentence
CF	=	Clause Features
sadv	=	sentence adverbials
rs	=	root sentence
INTER	=	interrogative
PASS	=	passive
NEG	=	negative
IMPER	=	imperative
PRES	=	present
FUT	=	future
PROG	=	progressive
PERF	=	perfective

2. <rs TF CF> :--> <np2 CF SUBJ> <vp2 CF TF VF VIRSUBJ VIROBJ>  
 <where VF contain +intran>  
 <where SUBJ nonconflicts VIRSUBJ>
- <rs TF CF> :--> <vp2 CF TF VF VIRSUBJ VIROBJ>  
 <where CF contain +imper>
- <rs TF CF> :--> <np2 CF SUBJ> <vp2 CF TF VF VIRSUBJ VIROBJ>  
 <np2 CF OBJ>  
 <where VF contain -intran>  
 <where SUBJ nonconflicts VIRSUBJ>  
 <where OBJ nonconflicts VIROBJ>
- <rs TF CF> :--> <np2 CF SUBJ> <vp2 CF TF VF VIRSUBJ VIROBJ>  
 <np2 CF OBJ>  
 <where VF contain +copula>  
 <where SUBJ nonconflicts OBJ>
- <rs TF CF> :--> <np2 CF SUBJ> <np2 CF OBJ> <vp2 CF TF VF  
 VIRSUBJ VIROBJ>  
 <where VF contain -intran>  
 <where SUBJ nonconflicts VIRSUBJ>  
 <where OBJ nonconflicts VIROBJ>
- <rs TF CF> :--> <np2 CF OBJ> <np2 CF SUBJ> <vp2 CF TF VF  
 VIRSUBJ VIROBJ>  
 <where VF contain -intran>  
 <where OBJ nonconflicts VIROBJ>  
 <where SUBJ nonconflicts VIRSUBJ>
- <rs TF CF> :--> <np2 CF OBJ> <vp2 CF TF VF VIRSUBJ VIROBJ>  
 <where VF contain -intran>  
 <where OBJ nonconflicts VIROBJ>
- <rs TF CF> :--> <vp2 CF TF VF VIRSUBJ VIROBJ> <np2 CF OBJ>  
 <where VF contain -intran>  
 <where OBJ nonconflicts VIROBJ>
- <rs TF CF> :--> <vp2 CF TF VF VIRSUBJ VIROBJ> <np2 CF OBJ>  
 <where VF contain +exist>  
 <where OBJ nonconflicts VIROBJ>

SUBJ        ::-->   NF  
 VIRSUBJ    ::-->   NF  
 OBJ         ::-->   NF  
 VIROBJ     ::-->   NF  
 NF         ::-->   +noun UCOMMON  
 UCOMMON    ::-->   +common UABSTRACT  
               -abstract UANIMATE  
 UABSTRACT  ::-->   +abstract UMOBILE  
               - abstract UANIMATE  
 UANIMATE   ::-->   +animate UHUMAN  
               -animate UBIOTIC  
 UBIOTIC    ::-->   +biotic  
               -biotic UMOBILE  
 UMOBILE    ::-->   +mobile | - mobile  
 UHUMAN     ::-->   +human | -human  
 VF         ::-->   INTRAN AGENT ERG REFLEX AUX EXIST ...  
 <rs TF CF>  ::-->  <np2 CF SUBJ> <vp2 CF TF VF VIRSUBJ VIROBJ>  
                   <np2 CF OBJ1> <np2 CF OBJ2>  
                   <where VF contain -intran>  
                   <where OBJ1 contain +human>  
                   <where SUBJ nonconflicts VIRSUBJ>  
                   <where OBJ nonconflicts VIROBJ>  
 <rs TF CF>  ::-->  <np2 CF OBJ2> <np2 CF SUBJ> <vp2 CF TF VF  
                   VIRSUBJ VIROBJ> <np2 CF OBJ1>  
                   <where VF contain -intran>  
                   <where OBJ1 contain +human>  
                   <where OBJ nonconflicts VIROBJ>  
                   <where SUBJ nonconflicts VIRSUBJ>  
 <rs TF CF>  ::-->  <np2 CF OBJ1> <np2 CF SUBJ> <vp2 CF TF VF  
                   VIRSUBJ VIROBJ> <np2 CF OBJ2>  
                   <where OBJ1 contain +human>  
                   <where VF contain -intran>  
                   <where OBJ nonconflicts VIROBJ>  
                   <where SUBJ nonconflicts VIRSUBJ>

Notes:

VF	= Verb Features
VIRSUBJ	= virtual subject
VIROBJ	= virtual object
NF	= Noun Features
INTRAN	= intransitive
ERG	= ergative
REFLEX	= reflexive
AUX	= auxiliary
EXIST	= existential

Rule 2 includes the following types of sentences:

	Subject	Verb (intransitive)		
		Verb (imperative sentence)		
	Subject	Verb (nonintransitive)	Object	
	Subject	Verb (copula)	Object	
	Subject		Object	Verb (nonin- transitive)
Object	Subject	Verb (nonintransitive)		
Object		Verb (nonintransitive)		
		Verb (nonintransitive)	Object	
		Verb (existential)	Object	
	Subject	Verb (nonintransitive)	Object (indirect)	Object (direct)
Object (direct)	Subject	Verb (nonintransitive)	Object (indirect)	
Object (indirect)	Subject	Verb (nonintransitive)		Object (direct)

3. <vp2 CF TF VF VIRSUBJ VIROBJ> :--> (<pp PF>)(<adv>) <neg>  
     <vp1 CF TF VF VIRSUBJ VIROBJ>  
     <where CF contain +neg>  
     <where PF nonconflict VF>
- <vp2 CF TF VF VIRSUBJ VIROBJ> :--> (<pp PF>)(<adv>)  
     <vp1 CF TF VF VIRSUBJ VIROBJ>  
     <where CF contain -neg>  
     <where PF nonconflict VF>

Notes:

PP = prepositional phrases  
 PF = Prepositional Features  
 adv = adverbial  
 neg = negative

4. <np2 CF1 SUBJ> :--> <s TF2 CF2 SUBJ> | <npl CF1 SUBJ> |  
     (<det>) <rel CF1 SUBJ> <de> |  
     (<det>) <rel CF1 HEAD> <de> <npl CF1 HEAD SUBJ>
- <np2 CF1 OBJ> :--> <s TF2 CF2 OBJ> | <npl CF1 OBJ> |  
     (<det>) <rel CF1 OBJ> <de> |  
     (<det>) <rel CF1 HEAD> <de> <npl CF1 HEAD OBJ>
- HEAD :--> NF

Notes:

s = sentence  
 det = determiner  
 rel = relative clause  
 de = relative clause marker (terminal symbol)

5. <rel CF1 SUBJ> :--> <s CF1 SUBJ TF2 CF2> | <npl CF1 SUBJ> |  
     <pp CF1 SUBJ>
- <rel CF1 OBJ> :--> <s CF1 OBJ TF2 CF2> | <npl CF1 OBJ> |  
     <pp CF1 OBJ>
- <rel CF1 HEAD> :--> <s CF1 HEAD TF2 CF2 > | <npl CF1 HEAD> |  
     <pp CF1 HEAD>

#### IV. FURTHER CONSOLIDATION OF THE LINGUISTIC DATA BASE: LEXICAL FEATURES AND INTERLINGUAL TRANSFER RULES

##### 1. Introduction

During the past few years, numerous improvements to the old SAS system had been made or conceived. To make it possible to incorporate these conceived improvements, a new system, the Quince system has been emerging. In this chapter, some previously conceived improvements to the two areas of the linguistic data base, the lexicon and interlingual transfer rules, will be elaborated in the light of recent developments in linguistic, artificial intelligence, and computational linguistic theories.

The discussion on the lexicon will be focussed on lexical features, whose implementation will be the most important single improvement to the lexicon. The preceding chapter has already provided us with a conceptual framework in which feature-handling mechanisms can be feasibly implemented. The nature and functions of these lexical features and their relationships to the other components of the linguistic data base will be described. The choice of the lexical features and the types of lexical information for the lexical entries in the lexicon will also be touched upon.

The interlingual transfer operation was conceived as an independent phase in the translation cycle. The interlingual transfer rules were regarded as belonging to an independent component of the linguistic data base. In this chapter, the actual separation of the interlingual transfer rules from the analysis rules will be emphasized once more. The nature, functions and different types of the interlingual transfer rules and a possible way of implementing them will be discussed. To further improve the interlingual transfer component, contrastive lexical and syntactic studies and contextual analysis will also be outlined as part of future endeavors.

## 2. Lexical Features

The lexical features as defined here include the following types: semantic, syntactic or morphological, contextual, and rule features. Each lexical entry in the lexicon may contain all or part of these feature types. Semantic features refer to what Katz (1972) called semantic markers. Examples are [Human], [Object], [Animate], etc. Syntactic or morphological features are those obligatory grammatical distinctions which a language imposes on its surface representation. Examples in English are gender distinction in the third person singular pronoun, singularity or plurality of countable nouns, etc. Contextual features refer to those contexts in which a lexical item may occur. For example, the contextual feature for an English noun is [+Det \_\_\_], and [+\_\_\_ NP] for an English transitive verb. The rule features are those which indicate which particular interlingual transfer rule(s) a particular lexical item will or will not participate in.

### 2.1 Nature of Semantic Features

The names "semantic features", "semantic markers", and "semantic primitives" are roughly equivalent terms used by different researchers in different disciplines. They are theoretical constructs intended to represent basic conceptual units or general sense-components. Since there is no unique way of breaking down the universe into the basic conceptual units, different researchers have different lists of those units. For example, Wilks (1973a) gave a list of sixty semantic primitives while Wierzbicka (1972) listed only fourteen. Based on our conception of the functions of semantic features as stated below, we are not providing an exhaustive list of them adequate for the analysis of the vocabulary of the Chinese language. Only those semantic features which will best account for our data and serve our practical ends will enter our feature list.

### 2.2 Functions of Semantic Features

Recent developments in semantic primitives or semantic features seem to have started in componential analysis in anthropology. In anthropological componential analysis, semantic features are intended to be the building

blocks of lexical fields. In some well-defined lexical fields, componential analysis has been successful, but it cannot be vigorously applied to the many fields which are not well defined with the same degree of success.

In linguistic semantics, semantic features are used mainly to indicate meaning relations among the lexical entries of the lexicon. They are also used to explain semantic anomaly in terms of feature incompatibility within a constituent.

In artificial intelligence, semantic primitives and relations are the building blocks of semantic networks, which represent meanings or conceptualizations of words or sentences in a language.

Within the conceptual framework of our feature grammar outlined in the previous chapter, the primary function of semantic features in the rules and the lexicon is to provide an elegant means of capturing general conditions on syntagmatic collocation or co-occurrence restriction. Those conditions on co-occurrence restriction are to be used as well-formedness conditions, to help disambiguate sentences and to throw out semantically anomalous interpretations.

In our future feature approach to analysis only those semantic features which appear both in the lexicon and rules will be used. That is, only those semantic features which have grammatical consequences will be entered in the lexicon. In this conceived grammar of Chinese, the grammar rules will incorporate semantic features as part of their well-formedness conditions. Those conditions on the rules will check the directly dominated non-terminals or terminals for compatibility. Only if no conflict arises will the constituent under consideration be accepted as being well-formed.

### 2.3 Building up the Semantic Feature Set

The first step to build up the semantic features for the future grammar rules and lexicon is to select and extract those relevant features contained in the existing grammar codes. Since those features were well-motivated to capture general co-occurrence restrictions in Chinese, they can be taken over without too much modification. The second step is to enter the extracted features in the respective lexical entries.

The existing features will be insufficient for our purpose in the future. As research goes on, we expect more features to be invoked to make



our grammar more sufficient. For example, a new set of semantic features will be needed if co-occurrence restrictions such as those between classifiers and nouns and in noun compounding are to be more adequately formulated than they are now. For ideas of semantic features that may be needed in the future, the feature set proposed in the 1974 Final Technical Report, p. 38, could be consulted. Other lists such as those given in artificial intelligence literature, Wilks (1972, 1973a) and Schank (1973, 1975c) may also be helpful.

The co-occurrence restrictions contained in the existing grammar codes can be extracted and restated in the new rules of our grammar. Some of the co-occurrence restrictions will have to be restated as our understanding of them deepens. For example, the co-occurrence restrictions between the verb and the subject and/or the object as indicated by the syntactic subcategorization in the current grammar have to be revised once they are better understood in terms of case relationships. Instead of a single co-occurrence restriction between the verb and subject and/or object, we may have to allow alternatives in a preferential scale in many cases.

In the existing grammar codes, the semantic features could only assume the positive value owing to the nature of the rules themselves. In the future grammar, each semantic feature should be allowed to have either the positive, negative or unmarked value. The unmarked value is intended to be used in those lexical entries where the dichotomous contrast with respect to a certain semantic dimension is neutralized. By allowing the negative and unmarked values for the semantic features the rules of the grammar and the lexicon will be greatly simplified.

#### 2.4 The Other Types of Lexical Features

The distinction between the semantic features discussed above and the syntactic or morphological features mentioned in section 2 can be at times very fussy, simply because the line drawn between syntax and semantics is not always clear. For our purpose, there is no need to make the distinction between these two types of features. The contextual features as defined above refer to the syntactic environments in which lexical items of a certain grammatical category or constituents can be predicted in terms of other categories or constituents when they concatenate. Both the syntactic and

contextual features are in part present in the grammar codes of the current grammar and can be selected, extracted, and entered into respective lexical entries. Whenever necessary new features of these two types can be incorporated.

The last feature type, rule features, are intended to either trigger interlingual transfer rules or to handle exceptions to them. The use of rule features will greatly simplify the interlingual transfer rules and only slightly increase the complexity of the lexicon. If the exceptions to the interlingual transfer rules were not registered in the lexicon, they would have to be handled by either writing less general rules or by further sub-categorizing. The rule features as defined here and in section 2 can be discovered only after the interlingual transfer rules have been formulated.

### 2.5 Types of Lexical Information

The incorporation of the lexical features into the lexicon will necessitate change of the existing dictionary format. The existing format used for the SAS allows only the information of grammar code, telecode, English gloss, and romanization. In the future format, at least the following types of information whenever applicable for each lexical entry in the dictionary should be included: telecode, syntactic category, syntactic and semantic features, contextual features, rules features, English gloss, and lexical disambiguating heuristics. As we will see below (section 3.3), the English gloss should be based on extensive contrastive lexical studies. The lexical disambiguating heuristics will be based on the immediate contextual information. Whenever low-level ambiguities arise this contextual information will be consulted first to resolve the ambiguities.

### 3. Interlingual Transfer Rules

The interlingual transfer was originally conceived as an independent phase of the translation cycle (see Wang, et al. 1971). Under this conception, the output of the analysis phase becomes the input to the interlingual transfer component. In practice, however, the interlingual transfer rules were incorporated into the Chinese grammar itself. Attempts to separate them from the analysis rules were made but have not yet been fully implemented

with appropriate machine programs. Although the analysis of Chinese is not an end in itself in machine translation, the mixture of these two phases may always confuse the issue and complicate the tasks at each phase. An independent component of interlingual transfer rules is both conceptually sound and linguistically practical. The separation should be carried out as soon as possible in the next phase of research.

### 3.1 Interlingual Component in Artificial Intelligence Approaches to Machine Translation

Wilks (1973a, 1973b) described an English-French machine translation system at Stanford University, which follows an "artificial intelligence" approach. Briefly speaking, in this approach English sentences of a paragraph are first converted into semantic or conceptual representations by-passing the syntactic analysis stage. Corresponding French sentences are then generated from those semantic or conceptual representations. Schank (1975b) also gave a brief account of an artificial intelligence approach to machine translation. The semantic or conceptual representation is represented by some kind of semantic network where the meaning(s) of a sentence is represented by primitive conceptual units and their relations (see Woods, 1975). It is supposed to be a universal interlingua. Any natural language can be decoded into it and encoded into another language.

One of the motivations behind the semantically-based approach to machine translation is "... that the space of meaningful expressions of a natural language cannot be determined or decided by any set of rules whatever -- in the way that almost all linguistic theories implicitly assume CAN be done." (Wilks, 1973a) According to Wilks, any string of words can be made meaningful by the use of explanations and definitions. But under current linguistic theories, those meaningful expressions may be excluded as unacceptable. Wilks' observation, though very true, should only be taken with some caution at this stage of machine translation research. Nobody has ever come up with any grammar for any natural language that is capable of describing all the well-formed sentences in that language under any linguistic theory, not to say a grammar for interlingua.

It is doubtful that the artificial intelligence to machine translation

described above really reflects human translation process. Besides, there are problems with semantic network representation of meanings for natural language. Woods (1975) gave a critical review of the semantic network representation of meanings for natural language. The problems seem to center around the issue of how to capture all the relevant meanings embodied in the very rich syntactic structure of a natural language in a semantic network representation. On the basis of the current state-of-the-art, our syntactically-based approach to machine translation should be maintained in the next phase of research.

### 3.2 Nature, Functions, and Types of and Formalism for the Interlingual Transfer Rules

In our conception of machine translation, it has been assumed that the English glosses in the dictionary will give us the necessary meaning elements in English sentences. Syntactic rearrangements of those elements in accordance with the English syntax and morphological adjustments to those English glosses in their base forms will give us the correct English output, semantically, syntactically, and morphologically. We will accept this conception as generally correct, except for some of the issues raised below.

Under the above conception, the output of the analysis phase, Chinese trees, will undergo syntactical and morphological adjustments, which are rules of the interlingual transfer component, to arrive at corresponding English sentences. Morphological adjustments are always idiosyncratic (i.e., lexical) in nature. In the following discussion, the types of interlingual transfer rules refer only to syntactic adjustments.

All the interlingual transfer rules contained in the current Chinese grammar will be sorted out and stated in terms of the three basic tree operations: deletion, substitution, and adjunction. We will follow the formalism presented in Friedman (1971) and Norin (1973) for representing these tree operations. So far only a small portion of the existing rules have been recast into this formalism.

Each interlingual transfer rule will take the form of a transformational rule. It will consist of a structural description and a structural change. The structural description will be based on the principle of

"analyzability" or "proper analysis."

### 3.3 Contrastive Lexical and Syntactical Studies

In the above discussion it was assumed that correct English glosses plus appropriate syntactic adjustments to a correctly-analyzed Chinese tree will produce a correct corresponding English sentence. How can we arrive at the correct glosses and appropriate syntactic adjustments? The solutions seem to hinge on extensive contrastive lexical and syntactic studies between Chinese and English.

#### 3.3.1 Contrastive Lexical Studies

In order to arrive at the correct English glosses for their Chinese counterparts, it is necessary to compare and contrast them in terms of their participation in a scene, or a schema, or a frame in their respective languages. Linguistically speaking, a frame refers to either a situational context or a lexical network which a word invokes. On the basis of its role(s) in a frame, the correct interpretation of a word in the source language can be rendered closest to its counterpart in the target language. This approach to contrastive lexical studies is partially in accord with the principle of structural semantics which states that the meaning of a word in a language is determined by its paradigmatic relationships to other lexical items in the same paradigm.

In addition to the paradigmatic relations, the syntagmatic lexical relations between any two words in both languages have to be taken into account. It has been familiar to linguists in the field of contrastive lexicography that words of same or similar meanings in two different languages may not have same or similar syntactic behavior. In many cases, the glosses of some Chinese lexical items, especially those "empty" words, cannot be given the appropriate ones without contrastive studies of their syntagmatic behavior in both languages being made. Right steps in this direction had been taken in the past. More work needs to be done in the future to update the whole dictionary to enable producing better English translations.

### 3.3.2 Contrastive Syntactic Studies

In the past, the linguists of our project had to carry out original contrastive syntactic studies between Chinese and English because of a lack of research by others in this area. Many interlingual transfer rules based on those studies were written. As research goes on, some old rules will have to be revised and new ones added.

In order to accommodate the earlier machine-implemented SAS, the interlingual transfer rules were embodied in the Chinese grammar, whenever a certain syntactic adjustment was needed for outputting correct English. It was introduced in all the rules that needed it. In the future when the interlingual rule component is separated from the grammar component, rule schemata can be used to capture the generalization of those structural changes.

The strategy to be followed to uncover the syntactic correspondences between Chinese and English will be to systematically compare and contrast the sentential and phrasal structures according to their types. Files of the Chinese sentence and phrase types will first be built up. Representative token sentences and phrases from each type will then be translated into the corresponding English sentences and phrases, with as few syntactic adjustments as possible. At the same time, the same syntactic adjustments will be attempted for sentences or phrases of the same type unless fidelity is violated. By doing so, it is hoped that rules of greater generalization can be uncovered. The Chinese sentences and phrases will later be compared and contrasted with their English counterparts. According to the scope of the systematic syntactic correspondences uncovered, interlingual transfer rules of varying generalization will be formulated. Sporadic exceptions to the rules will be entered as rule features in the relevant lexical entries so that other adjustments can be attempted.

The results of the extensive contrastive lexical and syntactic studies between Chinese and English outlined above will greatly enhance our linguistic data base. Those results will also be of great relevance to both teaching English or Chinese as a foreign language and Chinese-English bilingual dictionaries.

### 3.4 Contextual Analysis

In our syntax-based approach to machine translation, units of linguistic analysis and translation are sentences or sub-sentences. This decision was based on practical considerations. However, there are many cases where the syntactic and semantic information within those parse units alone is not enough to resolve ambiguities in them. Information from the surrounding contexts, both linguistic and/or situational, is necessary to do the work.

Linguistic contexts are provided by visible surrounding linguistic units, and the cues for resolving ambiguities in one sentence or sub-sentence are those lexical items or syntactic features in those units. The lexical and syntactical cues do not have to be in the immediately preceding or following sentence. Situational context, or more generally, knowledge of the world provides the richer and more important disambiguating information of the two. It is more reliable than linguistic context but more elusive. It is probably by far the most important criterion of selecting the correct or preferred interpretation among the many possible ones from the point of human language processing. However, world knowledge is enormous and cannot be feasibly incorporated in any natural language processing system of a great world domain. Even if we want to be more selective, we are always hampered by our incapability of predicting which part of our world knowledge will be useful or necessary in the system.

Although in a machine translation system, unlike a question-answering system, the world knowledge can be filled in by the reader of the translation; nonetheless, it is desirable to resolve as many ambiguities as possible during parsing and fill in as many linguistic gaps as possible during inter-lingual transferring for the reader of the translation. In the following, a few areas of the Chinese grammar will be exemplified to show the needs for contextual information.

#### 3.4.1 Elided Subjects

In Chinese writing in general, and scientific writing in particular, many sentences or clauses have their subjects elided. They are omitted because they are "understood" or "recoverable" from the contexts. In English, recoverable subjects are limited to a few well-defined linguistic

contexts, but in Chinese the elided subjects cannot be determined in purely linguistic terms. Extra-linguistic contexts are also involved. In order to output "readable" English, rules or heuristics to fill in the elided subjects have to be uncovered. Their discovery depends on a thorough contextual analysis of a huge corpus of Chinese texts.

Our preliminary investigation indicates that the author(s) of an article or textbook is the most frequently elided subject. In many other cases, the elided subject is the indefinite third person pronoun. Chinese sentences with elided subjects in the above two cases can always be translated into corresponding English subjectless passives. However, extra-linguistic considerations for contextual coherency may override these general stylistic conventions. It is those considerations that cause the trouble. Unless we come up with some principles of extra-linguistic contextual coherency other than the general conventions of stylistics in Chinese writing, we may recover the wrong subjects. Sometimes, linguistic well-formedness conditions may rule out the possibility of certain nouns in the surrounding sentences being the elided subject, but they cannot determine which noun is the one. In other occasions, linguistic cohesive devices, such as connectives, may provide cues for recovering the elided subject. Much research in the area of the principles of extra-linguistic contextual coherency needs to be done in the future to solve the problem of elided subjects in Chinese.

#### 3.4.2 Number, Tense, and Aspect

In Chinese the number of a noun and the tense and aspect of a verb are not morphologically marked. The number in many cases surfaces as quantifiers or determiners; and tense and aspect are often indicated by time nouns, adverbs, or particles, or any combination of them. In some cases, however, none of the overt markers exist. Since these syntactic features are obligatory in English, they have to be inferred from the Chinese contexts whenever the overt markers are absent. As in the case of elided subjects, we can sometimes rely on such cohesive device as connectives to provide the necessary information for the English reader. Information such as the organization of events along the temporal axis and its cues in a text may also be helpful in assigning the correct tense and aspect to the unmarked verbs.



Information of this type can be gathered only through contextual analysis.

### 3.4.3 Definite vs. Indefinite Reference

Definite vs. indefinite reference to nouns is an obligatory feature in the English grammar. In Chinese it is a derived feature and is not morphologically marked in all cases. When this opposition is not marked for a particular noun in a Chinese clause or sentence, only the context can provide information for the reader to make a decision. Generally speaking, the indefinite reference is often expressed by a preceding indefinite quantification expression or by default of any preceding definite expression while the definite reference is always expressed by repetition of a preceding noun, an anaphoric expression, or a preceding demonstrative. Other overt linguistic cues for the definite vs. indefinite reference to Chinese nouns need to be further investigated. In cases where no overt cues are available, extra-linguistic contextual analysis is necessary to provide the information for this opposition in English. The semantic notions of new vs. old or shared information, and of generic vs. specific or unique, may be helpful in the extra-linguistic contextual analysis.

We have briefly discussed the three areas of the Chinese grammar where some kind of contextual information has to be gathered and passed from Chinese sentences into the corresponding English sentences to produce "readable" translations. There are three major problems related to contextual analysis that have to be tackled in the near future. They are: (1) to determine the relevant contextual information, (2) to gather this information, and (3) to implement the information in the system. As the fields of linguistics, artificial intelligence, computational linguistics, and others advance, it is hoped that solutions to these problems may soon emerge.

## References

- Bobrow, Daniel G. and Allan Collins, eds. 1975. Representation and understanding: studies in cognitive science. New York: Academic Press, Inc.
- Brower, R.A., ed. 1959. On translation. Cambridge: Harvard University Press.
- Bruce, Betram C. 1975. Discourse Models and language comprehension. AJCL microfiche 35:19-35.
- Celce-Murcia, M. 1976. Verb paradigms for sentence recognition. AJCL microfiche 38.
- Charniak, E. and Y. Wilks, eds. 1975. Computational semantics: an introduction to artificial intelligence and natural language comprehension. New York: American Elsevier Publishing Company, Inc.
- Diller, T.C., ed. 1975. Proceedings, 13th annual meeting, ACL 4: Modeling discourse and world knowledge I. AJCL microfiche 35.
- \_\_\_\_\_. Proceedings, 13th annual meeting, ACL 5: Modeling discourse and world knowledge II, and text analysis. AJCL microfiche 36.
- Fillmore, C. 1969. Types of lexical information. In Studies in syntax and semantics, ed. by F. Kiefer, 105-137. Dordrecht-Holland: D. Reidel Publishing Company.
- Friedman, J., T.H. Bredt, R.W. Doran, B.W. Pollack, and T.S. Martner. 1971. A computer model of transformational grammar. New York: American Elsevier Publishing Company, Inc.
- Halliday, M.A.K. and R. Hasen. 1976. Cohesion in English. London: Longman.
- Hays, D.G. and J. Mathias, eds. 1976. FBIS seminar on machine translation. AJCL microfiche 46.
- Hudson, R.A. 1976. Arguments for a non-transformational grammar. Chicago: University of Chicago Press.
- Katz, J.J. 1972. Semantic theory. New York: Harper & Row, Publishers.
- Klappholz, D. and A. Lockman. 1976. Contextual reference resolution. In Diller, ed. AJCL microfiche 36:4-25.

- Leont'eva, N.N. and S.C. Nikitina. 1973. Semantic relations expressed by Russian prepositions. In *Machine translation and applied linguistics*, vol. 2:319-404. Athenaiion Verlag.
- Lyons, J. 1977. *Semantics*, vol. 1. New York: Cambridge University Press.
- Minsky, M.L. 1975. A Framework for representing knowledge. In *The psychology of computer vision*, ed. by P.H. Winston.
- Morin, Y.C. 1973. A computer-tested transformational grammar of French. *Linguistics* 116:49-113.
- Nagao, M. and J.I. Tsujii. 1976. Analysis of Japanese sentences by using semantic and contextual information. *AJCL microfiche* 41.
- Philips, B. 1975. Judging the coherency of discourse. *AJCL microfiche* 35:36-49.
- Nida, E.A. 1969. Science of translation. *Language* 45:483-98.
- Rieger, C.J., III. 1975. Conceptual memory and inference. In Schank, et. al., 157-288.
- Riesbeck, C.K. 1975. Conceptual Analysis. In Schank, et. al., 83-156.
- Schank, R. and K.M. Colby, eds. 1973. *Computer models of thought and language*. San Francisco: W.H. Freeman and Company.
- Schank R. 1973. Identification of conceptualizations underlying natural language. In Schank and Colby, eds., 187-247.
- \_\_\_\_\_, N.M. Goldman, C.J. Rieger, and C.K. Riesbeck. 1975. *Conceptual Information Processing*. New York: American Elsevier Publishing Company.
- Schank, R. 1975b. The conceptual approach to language processing. In Schank, et.al., 5-21.
- \_\_\_\_\_. 1975c. Conceptual dependency theory. In Schank, et.al., 22-82.
- Simmons, R.F. 1973. Semantic network: their computation and use for understanding English sentence. In Schank and Colby, eds., 63-113.
- Taylor, B. and R.S. Rosenberg. 1975. A case-driven parser for natural language. *AJCL microfiche* 31.
- Wang, W. S-Y., B.K. T'sou, and S.W. Chan. 1971. *Research in Chinese-English machine translation*. RADC-TR-71-211, Final Technical Report.
- \_\_\_\_\_. 1975. *Chinese-English translation system*. RADC-TR-75-109, Final Technical Report.

- \_\_\_\_\_ and P. Robyn. 1976. Chinese-English machine translation system. RADC- TR-76-21, Final Technical Report.
- Wierzbicka, Anna. 1972. Semantic primitives. Athenaiion Verlag.
- Wilks, Y.A. 1972. Grammar, meaning and the machine analysis of language. London: Routledge and Kegan Paul.
- \_\_\_\_\_. 1973a. An artificial intelligence approach to machine translation. In Schank and Colby, eds., 114-51.
- \_\_\_\_\_. 1973b. The Stanford machine translation project. In Natural language processing, ed. by R. Rustin, 243-90. New York: Algorithmics Press, Inc.
- \_\_\_\_\_. 19761. Natural language understanding systems within the A.I. paradigm: a survey and some comparisons. AJCL microfiche 40.
- \_\_\_\_\_. 1976b. Processing case. AJCL microfiche 56.
- Winograd, T. 1971. Procedures as a representation for data in a computer program for understanding natural language. MAC-TR-84, MAC Report.
- Woods, W.A. 1975. Foundations for semantic networks. In Sobrow and Collins, eds., 35-82.