EUR 5116 e

COMMISSION OF THE EUROPEAN COMMUNITIES

THE SLC-II SYSTEM LANGUAGE TRANSLATOR PACKAGE CONCEPTS AND FACILITES

by

S. PERSCHKE, G. FASSONE, C. GEOFFRION,
W. KOLAR and H. FANGMEYER

1974

Joint Nuclear Research Centre
Ispra Establishment - Italy

Scientific Data Processing Centre - CETIS

LEGAL NOTICE

This report is on sale at the addresses listed on cover page 4

at the price of B.Fr. 150.—

Commission of the
European Communities
D.G. XIII - C.I.D.
29, rue Aldringen
L u x e m b o u r g

April 1974

EUR 5116 e

THE SLC-II SYSTEM LANGUAGE TRANSLATOR PACKAGE CONCEPTS AND FACILITIES by S. PERSCHKE, G. FASSONE, C. GEOFFRION W. KOLAR and H. FANGMEYER

Commission of the European Communities
Joint Nuclear Research Centre - Ispra Establishment (Italy)
Scientific Data Processing Centre - CETIS
Luxembourg, April 1974 - 102 Pages - 21 Figures - B.Fr. 150.—

This is the first part of the description of a software system developed at CETIS by the Information Science Research Unit and intended to be the basic support for their R & D activities in Automatic Documentation and Language Translation.

The present volume is a general presentation of the system and should enable the reader to gather an over-all insight into the potential applications and the solutions of the problems found.

For more detailed information, two more publications are in preparation:
— an user's manual which is to enable the reader to generate and run an application system on the basis of SLC-II

—

EUR 5116 e

THE SLC-II SYSTEM LANGUAGE TRANSLATOR PACKAGE CONCEPTS AND FACILITIES by S. PERSCHKE, G. FASSONE, C. GEOFFRION W. KOLAR and H. FANGMEYER

Commission of the European Communities
Joint Nuclear Research Centre - Ispra Establishment (Italy)
Scientific Data Processing Centre - CETIS
Luxembourg, April 1974 - 102 Pages - 21 Figures - B.Fr. 150.—

This is the first part of the description of a software system developed at CETIS by the Information Science Research Unit and intended to be the basic support for their R & D activities in Automatic Documentation and Language Translation.

The present volume is a general presentation of the system and should enable the reader to gather an over-all insight into the potential applications and the solutions of the problems found.

For more detailed information, two more publications are in preparation:
— an user's manual which is to enable the reader to generate and run an application system on the basis of SLC-II

—

EUR 5116 e

THE SLC-II SYSTEM LANGUAGE TRANSLATOR PACKAGE CONCEPTS AND FACILITIES by S. PERSCHKE, G. FASSONE, C. GEOFFRION W. KOLAR and H. FANGMEYER

Commission of the European Communities
Joint Nuclear Research Centre - Ispra Establishment (Italy)
Scientific Data Processing Centre - CETIS
Luxembourg, April 1974 - 102 Pages - 21 Figures - B.Fr. 150.—

This is the first part of the description of a software system developed at CETIS by the Information Science Research Unit and intended to be the basic support for their R & D activities in Automatic Documentation and Language Translation.

The present volume is a general presentation of the system and should enable the reader to gather an over-all insight into the potential applications and the solutions of the problems found.

For more detailed information, two more publications are in preparation:
— an user's manual which is to enable the reader to generate and run an application system on the basis of SLC-II

— a system maintenance manual which is to contain a detailed description of the system modules.

Part of the programming was performed under contract by the software company ITALSIEL S.p.A., Rome.

— a system maintenance manual which is to contain a detailed description of the system modules.

Part of the programming was performed under contract by the software company ITALSIEL S.p.A., Rome.

— a system maintenance manual which is to contain a detailed description of the system modules.

Part of the programming was performed under contract by the software company ITALSIEL S.p.A., Rome.

EUR 5116 e

COMMISSION OF THE EUROPEAN COMMUNITIES

THE SLC-II SYSTEM LANGUAGE TRANSLATOR PACKAGE CONCEPTS AND FACILITES

by

S. PERSCHKE, G. FASSONE, C. GEOFFRION,
W. KOLAR and H. FANGMEYER

1974

Joint Nuclear Research Centre
Ispra Establishment - Italy

Scientific Data Processing Centre - CETIS

ABSTRACT

This is the first part of the description of a software system developed at CETIS by the Information Science Research Unit and intended to be the basic support for their R & D activities in Automatic Documentation and Language Translation.

The present volume is a general presentation of the system and should enable the reader to gather an over-all insight into the potential applications and the solutions of the problems found.

For more detailed information, two more publications are in preparation:

— an user's manual which is to enable the reader to generate and run an application system on the basis of SLC-II

— a system maintenance manual which is to contain a detailed description of the system modules.

Part of the programming was performed under contract by the software company ITALSIEL S.p.A., Rome.

KEYWORDS

INFORMATION RETRIEVAL
INFORMATION SYSTEMS

TABLE OF CONTENTS

Table of Contents (contd.)

Preface

This is the first part of the description of a software system developed at CETIS by the Information Science Research Unit and intended to be the basic support for their R&D activities in Automatic Documentation and Language Translation.

The present volume is a general presentation of the system and should enable the reader to gather an over-all insight into the potential applications and the solutions of the problems found.

For more detailed information, two more publications are in preparation:

- a "User's Manual", which is to enable the reader to generate and run an application system on the basis of SLC-II,
- a system maintenance manual which is to contain a detailed description of the system modules.

Part of the programming was performed under contract by the software company ITALSIEL SpA, Rome.

1. INTRODUCTION

1.1 Motivation and Objectives

SLC-II (Simulated Linguistic Computer) is intended to be the basic software for natural-language data processing. It is based on some ideas which emerged in the context of the Georgetown Machine Translation Project [1, 2] in which A. F. R. Brown conceived and implemented a system called SLC for machine translation. This system was, in fact, never given publicity as an independent software. Rather it was considered to be an integrated part of the translation program. Further, it was closely linked to the basic linguistic concepts of the Georgetown project (symbol substitution approach), so that it appeared of little use for advanced linguistic solutions.

However, the basic idea of SLC is still to be considered valid: language data processing, the R&D and the applications involved (translation, abstracting, indexing, question-answering etc.) imply extremely complex operations and large data bases on one hand, and a high efficiency on the other,motivated by the extreme ease and relative rapidity and economy of the corresponding functions performed by man.

As a consequence, it is observed that in enterprises involving language data processing, the majority of resources (intellectual and economic) is exhausted by the solution of problems to be considered trivial from the point of view of the linguist or the information scientist and there remains little space for the solution of the true problems, or, even worse, the software solution limits the possibility of describing the problems in their proper terms.

SLC, along with some specialized programming languages like COMIT or SNOBOL is to be considered as an attempt to offer the linguist or information scientist a tool for describing his problems in his proper terms, so as to relieve him of complex data and storage management considerations.

While COMIT and SNOBOL chose the classical solution of a higher-level programming language, and, from the point of view of efficiency, were limited to an experimental laboratory environment, SLC kept in mind the over-all efficiency of practical applications like machine translation and chose the approach of simulating the functions of a special-purpose computer.

There is another important aspect in SLC which permitted to increase efficiency: for those phases of the process, which could be considered linguistically resolved, such as dictionary search, invariant algorithms, optimized from the point of view of data and storage management, were developed, for variable dictionaries and grammars, while the algorithmic pro-

gramming language concerns the less stabilized phases, and presents to the linguist the data in conformity with his usual way of working, i.e. as a logical text unit, which can be processed from left to right (or vice-versa).

The principal limitation of the solution was mentioned before: the poverty of the underlying linguistic model. Another limitation concerns the implementation too closely tied to the IBM 7090 computer.

The design of the new system, which, in homage to its precursor, was called SLC-II, put the following objectives:

- independency of a particular application: SLC-II is to be considered as a component of the basic software - the language translation package - of integrated fully automatic documentation, translation and data base management systems. The other components of this package are the software for automatic thesaurus construction and document retrieval and data base management.
- independency of a particular language model: SLC-II applied this principle not so much to grapheme processing and morphology for which one model - retained satisfactory - was chosen, as to the central problem of computational linguistics - syntax and semantics.
- installation independence and transportability: this objective could not be fully realized in the present version. SLC-II is implemented in IBM/360 Assembler language and virtually is only transportable to other computers with byte-organized storage. However, there exists a long-range project - called SLC-III, of re-formulating the system in a higher-level language such as ALGOL or PL/1.
- usability as a research tool in linguistics and information science: SLC-II is a modular system which permits not only application-oriented usages such as translation, indexing, abstracting, etc., but also research oriented ones like parsers, transformational grammars, generative gram-

mar statistics etc.

1.2 Applications

As a whole, SLC-II is a language translation package - languages can be both natural and artificial. The range of applications taken into consideration during system design is:

- machine translation: as a requirement of the system capability, a multilingual reversible translation system of the 2nd-3rd generation was assumed, in which the recognition and the generative parts of the translation are independent one from the other, and the link is established by a metalinguistic representation of the texts. For less advanced solutions, transfer - i.e. symbol equivalence - functions can be introduced, especially for the lexic.

- automatic indexing is to be considered as a particular case of translation from natural language into an artificial language (information retrieval language - IRL). At the present stage of development, most of the IRL practically used, are so-called syntax-free IRL (c.f. coordinate indexing) and the automatic indexing methods which appear most promising for practical applications are rather statistics- than linguistics-based. However, SLC-II is capable of using also advanced IRL with complex syntactic and semantic relation devices.

- automatic abstracting and summarizing: at present, very little progress in this field has been made, and the attempts known of are rather extracting.

- automatic query formulation for information retrieval: this application is very closely linked to automatic indexing. In this context, SLC-II becomes a subset of a larger automatic information retrieval and question-answering system.

- automatic IRL development: a software package is at present being developed at CETIS which uses a subset of SLC-II, and applies statistical

methods on lexeme basis for the definition of the vocabulary of an IRL, and of the paradigmatic relations between the terms.

- machine-aided translation: in the philosophy of CETIS, it should be an interactive post-editing facility with the possibility of access to specialized terminological vocabularies.

1.3 Future Developments

The present version of SLC is operating in batch mode, which is adequate for applications as machine translation or indexing, but unsatisfactory in the information retrieval environment (query formulation) and in research applications. Therefore, a conversational version of SLC-II is being designed at present and implementation will start in 1974.

This conversational version will be added to the information retrieval and data base management package which is being developed at CETIS, so as to permit, for example, interactive query formulation and information retrieval. Another development which was mentioned before, is the formulation of SLC-II in a higher-level language so as to permit full transportability of the package.

This project will possibly be realized in cooperation with an information and computer science research institute of a university.

The question which is to be resolved first is whether higher-level languages which dispose of compilers for different computer models (c.f. FORTRAN, COBOL, PL/1 or ALGOL) are adequate for this class of problems, or, eventually, should one design a new language, for which a set of compilers would be implemented.

2. GENERAL SYSTEM DESIGN

2.1 System Conception

Each application of SLC-II is interpreted as a translation process which starts from the graphic representation of the source text and produces, as a result, the graphic representation of the same text in the target language.

In order to make analysis and design more manageable, the process was broken down into a series of basic functions or cycles, each of which, in principle, has three components:

- an algorithm which, as a final objective, should be invariant with respect to the languages and applications chosen.
- a dictionary which contains the elements of the language handled, and all information about the elements necessary for the process.
- a grammar which is a collection of the rules of the language processed represented according to the language model chosen.

The translation process itself is broken down into three principal phases:

- the recognition phase which has the purpose of transforming the continuous character string representing the source text into the representation of the same text according to the conventions of the language model (meta-lingua).
- the transfer phase which actually is a concession to the difficulty (or impossibility) of fully formalizing the language. In effect, transfer bases on the concept of the equivalence of symbols (beloved in word-for-word translation) and is applied for all elements of the language which in the language model appear just as codes and are not semantically defined. In the 2nd-generation translation projects, transfer is primarily applied to the lexic. There exist a few attempts of fully formalizing also this component of language (as, for instance, Ceccato with differentiation, figuration and

categorization), but their approach is, in general, purely theoretical and speculative, and description and analysis usually apply only to a few selected samples. Large-scale applications never were seriously attempted, and it is even dubious whether the investments and efforts are actually justified in the context of the sole machine translation. They are certainly necessary for advanced solutions in documentation, such as contents analysis and summarizing, but not in short-range projects.

- the generation phase which is the inverse process of the recognition phase, i.e. the point of departure is the metalinguistic representation of the target text, which is to be transformed into a character string according to the grammar of the target language. This phase is indispensable for applications with natural-language output, and might be omitted, when the target language is formalized, i.e. a meta-language.

2.2.1 Recognition Phase

For the time being, the source text is assumed to be written text in machine-readable form, such as hand-coded material, or tapes as a by-product of text-processing and computer-controlled type-setting. Phonetic input was not taken into consideration.

The first cycle, hence, is an input module, which reads the "continuous" character string and transforms it into substrings qualifying them as:

- "word items", i.e. strings which according to the grammar are elements ("words") of the source language and must be looked up in a dictionary, or
- "non-word items", i.e. strings which are not elements of the source language, whose function may be either computable from the string itself (e.g. numerical values or lay-out control symbols), or just strings with an unknown function, as, for instance, foreign-alphabet data.

The model chosen is a finite-state automaton with a context-sensitive immediate-constituent grammar. The technique used in the implementation is that of higher-level programming language compilers with a scanner, parser and semantic interpreter. In the batch version, the input cycle is executed independently, without dictionary control. A subset of the SLC-II programming language is dedicated to the coding of the dictionary and the grammar of this cycle. The algorithm is invariant.

The "word items" identified by the text analysis module are processed by the second cycle of the system, the dictionary search and morphological analysis module. The source language morphological search dictionary is organized as a stem-suffix dictionary. Analysis is performed from left to right. From the linguistic point of view, the following facilities are provided for: word inflection through suffix analysis with morpheme chaining, word derivation analysis, segmentation of compound words, prefix analysis, tentative suffix analysis of unknown words (from right-to-left), homography detection.

The dictionary search and morphological analysis algorithm is invariant. Stem- and suffix analysis are implemented as a finite state context-free grammar with a table-driven parser with the first access to the syntax tree through the dictionary.

A subset of the SLC-II programming language is dedicated to the symbolic coding of grammatical definitions, paradigm tables and dictionary entries. A set of utility programs is provided for the creation and maintenance of the source language morphological search dictionary.

The result of the dictionary search is the replacement of the grapheme representing the "word item" by one (or more in the case of homographs) lexeme identification code (LXN) and the precise description of the morphological form. Each item afterwards is linked to the corresponding

entry of the source language dictionary.

While the first and the second cycle are performed for the maximum possible batch concurrently so as to increase program efficiency by exploiting the phenomenon of the repetition of words, the subsequent cycles are performed on logical text units, which can be defined parametrically depending on the application (e.g. sentence, paragraph, abstract, etc.).

Further, due to the variety of applications and instability of language models, for the time being, no attempt was made for the definition of invariant algorithms for parsers, transfer functions and generative grammars on the syntactic level, but rather a procedural special-purpose programming language was designed which is the central component of the SLC-II programming language. Furthermore, no grammar models and dictionary formats were defined for the single cycles: SLC-II rather enables the application system designer to define his own models and formats and to use his proper nomenclature. A subset of the SLC-II programming language was explicitly defined for coding grammars and dictionary entries and a set of utility programs permits to create and maintain SLC dictionaries.

In the course of development of application systems, (e.g. the automatic documentation and the machine translation projects at CETIS), a set of more or less generally acceptable algorithms, grammars and dictionaries will be defined and implemented and become a generalized application package of the system.

The SLC-II programming language, apart from the general CPU and I/O capabilities, was designed for syntax- and semantics-oriented language processing.

The basic linguistic model underlying the design is the one of the Italian Operational School (Ceccato). However, it is also capable of handling dependency grammars (Chomsky-Hays) and relational grammars (Vauquois). In the latter case, the graphs should be broken down into a set of relations.

The "translation" algorithms can be designed for "bipartite" (algorithm with built-in grammar dictionary) or for "tripartite" (algorithm grammar dictionary) organization. The latter should be more adequate for the design of general-purpose algorithms.

The source data processed by the problem program in the recognition cycle is the internal representation of a logical text unit (e.g. sentence) as a result of dictionary search by means of a four-level tree structure (item - match - segment - form):

- item is a grapheme isolated at input time (word item or non-word item). Eventual input convention inconsistencies which result into reading variants are considered as separate items and must be resolved at problem program time (c.f. period as end-of-sentence symbol and abbreviation, hyphen at the end of a line etc.)
- match is the result of homography detection at dictionary search time. Homography resolution is left to the problem program
- segment is the result of the analysis of compound words, prefixes and word derivation. The distinction between segmentation and derivation is given by the configuration of HWO at the "Form" level. Distinction between prefixes and word segment is given by the associated dictionary entry.
- form is the result of morphological analysis. It is represented by a binary vector, conventionally called Headword O (HWO), whose length is parametrized. Note that dictionary search does not handle morphological homographies. Therefore, the linguist must foresee all possible

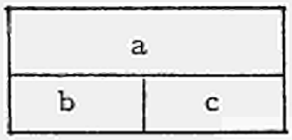
homographs , and represent them by a unique HWO. At problem program time, if one desires to, one can replace the unique description by as many non-ambiguous HWO as there are meanings, e.g. the Russian word "DOROGO1" on one hand produces a homograph on lexical level ("DOROGA" = "WAY", DOROGO1 = "DEAR"). On the level of the form, the variant "DOROGA" is unambiguous: instrumental singular, while the variant "DOROGO1" corresponds to the following forms:

1. Nominative singular masculine (1M)
2. Accusative singular masculine inanimate (4MI)
3. Genitive singular feminine (2F)
4. Dative singular feminine (3F)
5. Instrumental singular feminine (5F)
6. Locative singular feminine (6F).

After the transformation, one can obtain the following representation of the item:

It is up to the problem program to resolve these ambiguities.

At the level "form" the text image is linked to the syntax storage. As was said above, the syntactic model basically is the one of the Italian Operational School. In principle, a syntactic unit is constituted by one operator and two operands, all of which may be either terminal elements or other relations. The graphic representation of a relation used is:

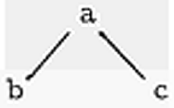


a: operator

b: 1st operand

c: 2nd operand

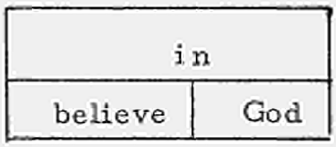
This representation is equivalent to a tree-structure



The interpretation of a relation depends on the level of linguistic analysis. Basically, one can distinguish three levels:

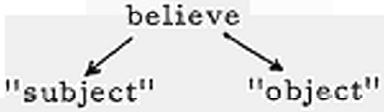
- surface structure syntax
- complement function of relations (deep structure)
- semantic definition of relations.

If one takes the phrase "believe in God", one can construct the relation:



on the 1st level, it is just a relation with the syntactic operator "in", a verb as 1st operand and a noun as second operand, by no way distinguished from phrases like "live in London" or "look back in anger".

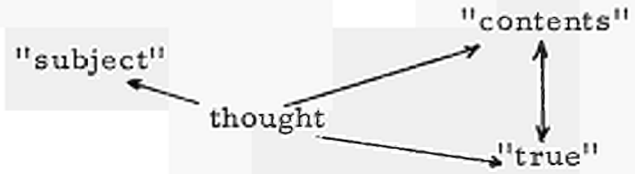
On the second level, the verb "to believe" is defined in view of the complements it may assume: there is a "subject" and an "object":



The "subject" may be defined in different ways on the surface structure level (c.f. I believe, the man believing, he is supposed to believe, I force him to believe, etc.). The "object" may be expressed, in alternative, in the following ways:

(1: "in", 2. "dative", 3. "accusative", 4. indirect clause, 5. "accusative with infinitive etc.). In the example, the first alternative was chosen.

On the semantic level, the meaning of "believe" must be analyzed, and the complements explained in terms of this meaning. "believe" can be defined as "thought" plus the judgement "true" or "right" given to the content of the thought. As a consequence, the situation resulting from the complements is as follows:



In "believe something", "believe that" etc. just the contents of the thought is explained (e.g. I believe that SLC is a good system"); in "believe someone" the situation is somewhat more complicated historically, it is assumed that the "thought" has been communicated to the subject by somebody, and the complement is just the subject of the communication.
In "believe in someone/something" the contents of the thought is an object judged "true" or "right" (c.f. believe in God, in Hitler, in the relativity theory). The concept "true" may be interpreted as "existence" or "correctness".

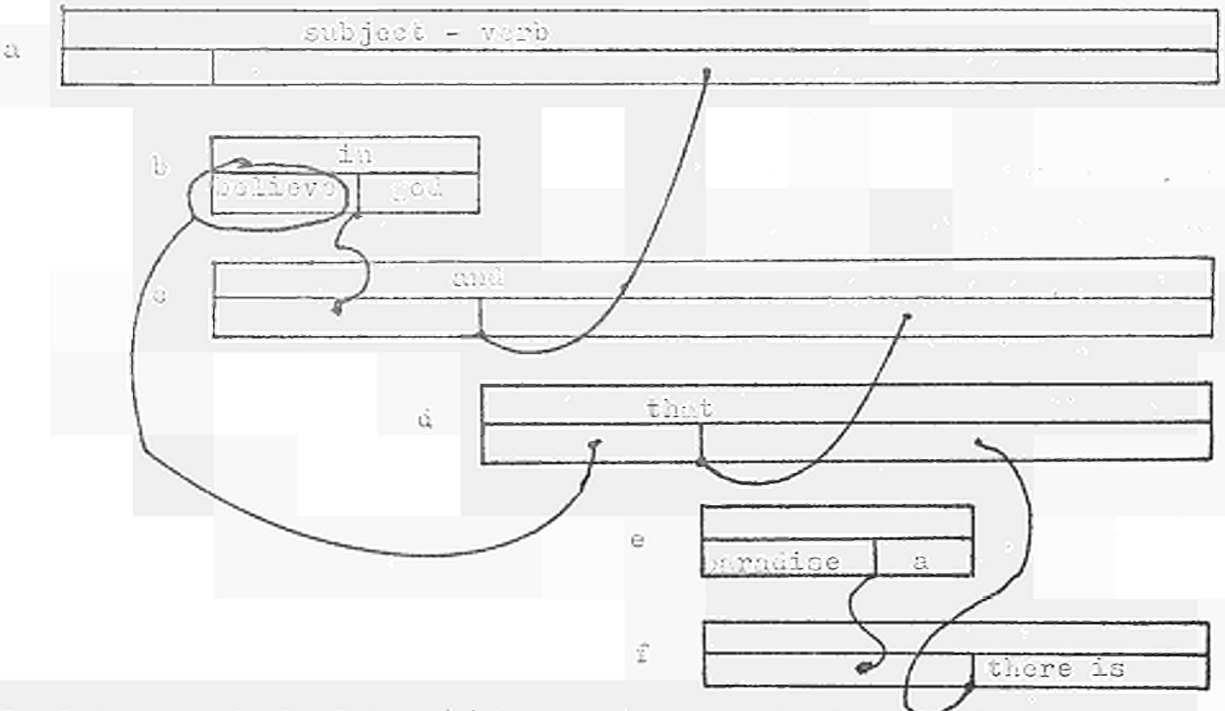
The syntactic model permits to associate the following information to each relation:

1. Word order: i.e. the sequence in which operator and operands are located in the text.
2. The "boundaries" link to the items immediately to the left and right of the items which are part of the relation. This information is useful during analysis especially if "immediate constituent" grammars are used.
3. The "delimiters": punctuation marks not always can be handled as syntactic operators or operands. Therefore, they may be just declared as

delimiters and associated to the left or right of a relation.

4. The links to the elements contained in the relation. The links may be either to the "form level" of the text image (terminal) or to another relation (non-terminal).
5. Classification of the above links: in general, items are inserted directly into a relation. However, there are two other ways of insertion:

   - artificially supplied items in the case of elliptic expressions
   - resumed items: this case occurs primarily in coordinative relations and in comparisons c.f. in the phrase: "I believe in God and that there is a paradise", the structure is as follows

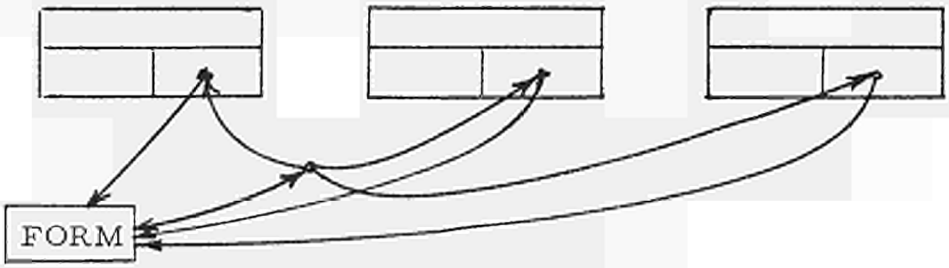


The 1st operand of relation (d) has not been supplied explicitly - in order to understand the relation, one has to resume it from (b). This operation is very similar to some functions in algebra:

$$ab + ac = a(b+c)$$

Another mechanism consists in the handling of syntactic homographies. Each item (at form level) may occur in more than one relation, and in this case the relations are alternative solutions. For this reason, the items and relations are linked to higher-level relations not directly, but

by means of a list element.



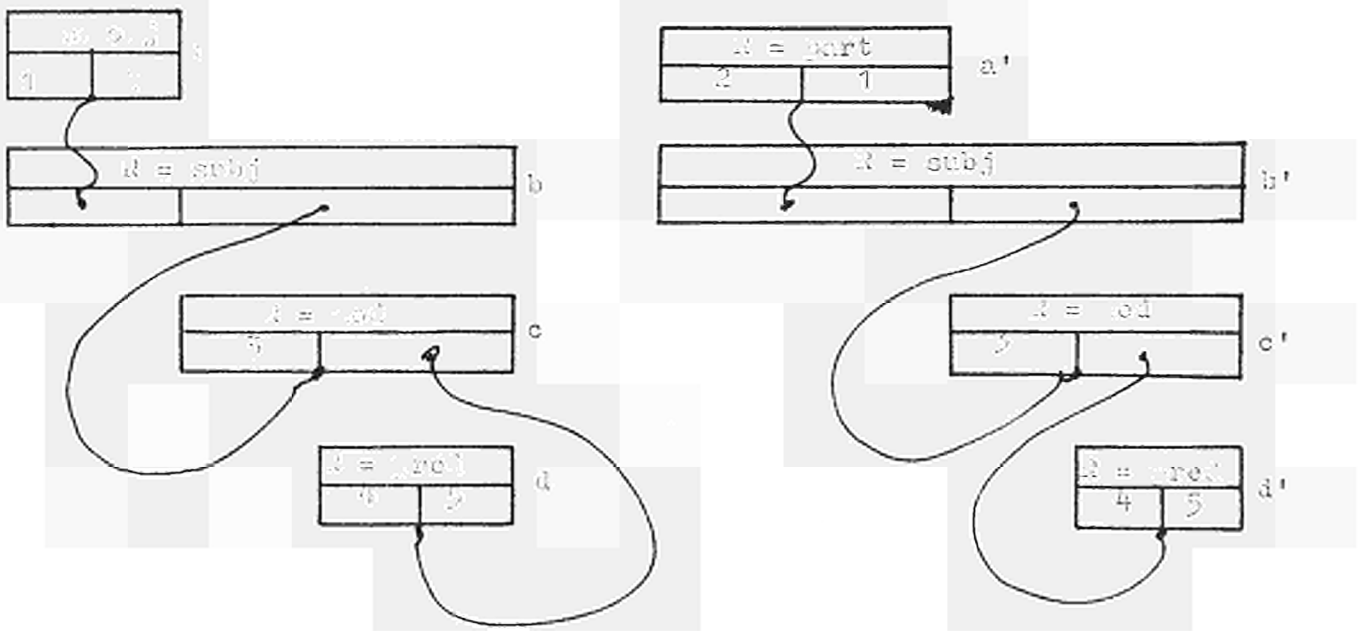
This mechanism permits to handle syntactic homographies with a maximum of economy avoiding the proliferation of identical partial solutions. At the upper level of syntactic structures there is a two-level tree structure.

The first groups the alternative solutions, while the second contains the complementary partial solutions in the course of analysis. Assuming that the sentence:

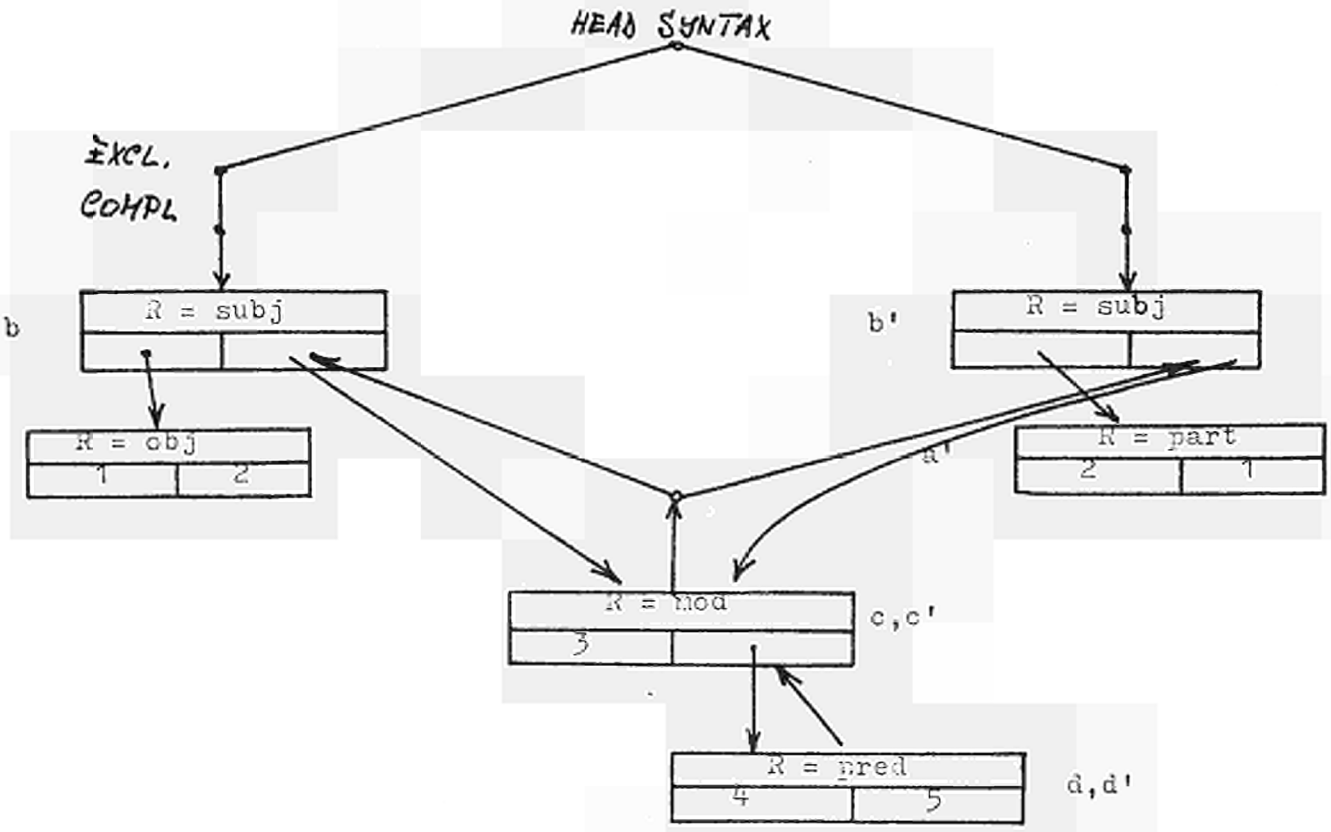
"Flying planes may be dangerous"

1 2 3 4 5

can be analyzed in two ways



The representation of the two solutions would look like the following, given that the relations c, d and c', d' are the same



The instructions which enable one to construct the syntactic representation of the source text, normally work in two phases:

- first, a tentative relation is constructed,
- second, if it is accepted by all grammatical rules, it is set permanent and linked to the other relations.

The way of linking relations may be performed in two ways:

- one either maintains all previous relations unchanged and declares the use of the element or relation which enters as an operand or as operator into a higher-level relation as a new possible use of the relation.
- the second permits to link two substructures at any level so as to produce one sole result. This function, if not performed at the highest level, implies a transformation of the existent structures.

If the use of any one of the relations at a level higher than that of the insertion is not univocal, i. e. it appears as an element in more than one alternative relation, at the moment of insertion the structure involved is copied so as to avoid unpredictable side effects. This kind of transformation is only possible in the course of a top-to-bottom exploration of a structure which keeps track of the passes performed.

The system being syntax-oriented, at the end of the source text analysis cycle, the syntactic representation is interpreted as the meta-linguistic description, and all elements which have no link are deleted from storage.

The second cycle - the transfer - actually is an auxiliary cycle designed for handling those elements, which could not be formalized in the meta-linguistic representation, on the basis of symbol equivalence. This applies, at the present stage of development, principally to the lexical transfer, as the attempt of a completely formalized representation of the meaning of words still appears to be little realistic in application-oriented projects. Of course, this transfer cycle can be omitted.

The transfer cycle may imply structural transformations of the source text image, mainly due to the necessity of replacing single words by expressions and vice-versa.

The result of the transfer cycle is a meta-linguistic representation of the text in the target language, which is used as input to the third cycle - the generative algorithms. The task of this cycle is inverse to that of the first cycle: the syntax-oriented text image is to be transformed into a linear string of items, each of which is constituted by the identification code of the lexeme, the definition of the inflectional form and by lay-out control codes.

The process implies the definition of the surface structure of the target text, with the necessary transformations due to the target language grammar, and the conversion of the syntactic representation into a linear string.

The subsequent cycles - morphological generation and editing, again, for efficiency purposes, are performed on the entire text batch processed in input.

The algorithms for morphological generation and editing use the same type of grammar as the analogical recognition algorithms, and are invariant. As the entries of the target language generation morphological dictionary are accessed by the lexical identification code (LXN), they may have also nil-stems in the case of irregular inflections (c.f. go - went).

Generative morphology permits to attach suffixes and prefixes to the stem and to chain them (suffixes from left-to-right and prefixes from right-to-left).

The editing algorithms are capable of processing information obtained from the text image during text analysis (e.g. lay-out data, capitalization etc.) and that obtained from the target language dictionary and grammar (e.g. capitalization of nouns in German).

Another function consists in the transformation of strings due to phonetic phenomena (c.f. a/an in English; au/à l'in French, etc.).

2.2 System Organization

The SLC-II system is implemented as a set of re-entrant and recursive modules with dynamic task and storage management controlled by a monitor module. The present version is designed for batch mode operation,

but the single modules are all conceived in view of future conversational application in a time-sharing environment.

In the batch version, the single phases of the process are designed for the maximum possible amount of text with a given main storage size available. Only the central part, i.e. the phases programmed in SLC-II, is designed for processing a single logical text unit at a time.

In the over-all program organization, one can, thus, distinguish three separate cycles, which are characterized by the different amount of source text processed:

- the external cycle, which includes text analysis and dictionary search at the input side and morphological generation and editing at the output side. The amount of text depends on the size of core storage and the number of different word items (which may vary depending on the homogeneity of source texts).
  Experience made shows that with 300 kbytes core storage, one can process app. 12 k different word items, which according to our experience, may cover from a minimum of app. 70 k to a maximum of over 250 k current word items depending on the corpus. Non-word items in no way affect system capacity;
- an intermediate cycle, which subdivides the text processed in the external cycle into minor batches, loads the SLC-dictionary entries and organizes, one by one, the logical text units on core storage. The amount of source text which can be processed in one cycle depends on the main storage size and on the number and average length of the dictionaries associated with the three problem program cycles. Precise estimates are difficult to make; as an indicative value one can assume that in a partition of 300 k, app. 120 k may be occupied by the dictionary entries;
- an internal cycle per logical text unit. The definition of a logical text unit is application-dependent. It may be a sentence in machine translation, an

abstract in indexing etc.

For the system, a logical text unit is a string of items terminating in a "delimiter" item to be communicated to the system as a parameter. When defining the logical text unit, one should keep in mind that the maximum number of items which can be processed in one cycle, is app. 800-1,000.

If the problem program logic demands for longer logical text units, one can define "sub-delimiters" which are used if the "delimiter" was not encountered during loading, and ensure communication between the parts of the logical text unit either through the intra-cycle communication storage or I/O operation on temporary data sets.

The system operation is controlled by a set of options and parameters which can be pre-defined, compiled and link-edited as a load module in the system library, or introduced at execution time through control cards. Data sets are defined at J.C.L. level.

The options are, among others, the names of the grammars associated to the invariant algorithms, inclusion or exclusion of optional functions in algorithms (e.g. source text listing, frequency counts, segmentation, homograph detection etc.), the names of the SLC-II main programs associated to the three cycles of the problem program etc.

2.3 System Generation

The SLC-II system is organized as a library of executable load modules, which is to be used as JOBLIB or STEPLIB under OS control. One should keep in mind that SLC-II is a basic software package, and, as such, does not perform any application function, in the same way as operating systems or compilers.

In order to generate an application system, the following operations are necessary:

- analyse the source text formats and coding rules, decide about word- and non-word items, write, compile and link-edit the text analysis grammar. Communicate the name of the grammar to the system;
- analyse the source language morphology, decide about the level of analysis (derivation or not, prefix analysis or not, word segmentation or not, homography detection or not, suffix analysis of unknown words or not) write, compile, link-edit the grammar, communicate the grammar name to the system;
- compile the relative source language morphological search dictionary;
- define the source language recognition grammar and algorithms; compile and link-edit; communicate the name to the system;
- define all information about the lexemes used by the recognition grammar and algorithm; compile the source language dictionary;
- define the transfer grammar and algorithms, etc.;
- compile the transfer dictionary;
- define the target language generation grammar and algorithms, etc.;
- compile the target language dictionary;
- define the target language morphology, etc.;
- compile the target language morphological dictionary.

In the first applications, the preparatory work appears to be immense. One should, however, keep in mind that many of the data bases and grammars produced in different applications, such as text analysis grammars, generative and recognition morphological dictionaries and grammars etc. are virtually application-independent and exchangeable and, hence, need to be compiled only once.

The application of a basic software like SLC-II, in principle, resolves one of the most serious problems hampering the development in computational linguistics and information science: the exchangeability of the data bases and research results between different projects.

3. SYSTEM DESCRIPTION

3.1 Text Analysis

3.1.1 Functions

This is the input module of the SLC-II system. The variety of machine-readable text sources (commercial tape services, hand-coded material, tapes used for computer-controlled type-setting, etc.), made it necessary to create a powerful tool for handling different coding conventions.

For the system, the source text is a continuous string of characters which must be broken down into substrings which are either

- word items, i.e. elements of the source language and are normalized and sent as arguments to the second SLC-II module, the dictionary search, or
- non-word items, i.e. elements alien to the source language (e.g. control characters and codes, foreign-alphabet data, digits, etc.). These data are considered as mere character strings in the further process and not linked with any dictionary data.

The solution chosen is very similar to that of a table-driven compiler. For the description of the specific coding conventions, and the concrete actions to perform upon the strings, a subset of the SLC-II programming language was defined, which has the following components:

- atom definition, which identifies the minimal substrings to be passed by the scanner to the parser,

- syntax, which controls the operation of the parser and invokes the semantic actions. The syntax is a context-sensitive immediate-constituent grammar,
- semantic actions, which are invoked by the syntax and permit manipulation of the strings and their issue to the system as word or non-word items,
- control dictionaries, conversion tables, etc. for decision making and string manipulation,
- error diagnostic and recovery facilities, which can be linked both to syntax and semantics.

The word items are sorted according to the SLC sequence. The amount of text which can be processed in one cycle depends on the number of different word items which can be kept in core storage. Optionally, for statistics purposes, one can demand the frequency count for each word item.

3.1.2 Text Analysis Processor

The text analysis module, according to the general philosophy of the system, has been conceived as an invariant algorithm - the processor - controlled by different grammars.
The processor is constituted of 3 components;
The scanner has the following functions:

1. To read the input records and define start and end of the data fields,
2. Eventually, to link strings over the end-of-record,
3. Optionally, to process header and trailer fields and pass them as strings to a special semantic action routine,
4. To scan the data fields and to pass the strings defined as "atoms" by the associated grammar to the parser ("GETATOM").

The parser has the following functions:

1. To match the atoms passed by the scanner against the syntax tree,

2. To invoke the associated semantic actions,

3. To invoke eventual error diagnostic and recovery actions.

The interpreter executes the semantic action routines and the error diagnostic and recovery functions.

3.1.2.1 - Logical Flow Charts of "Scanner", "Parser", "Interpreter"

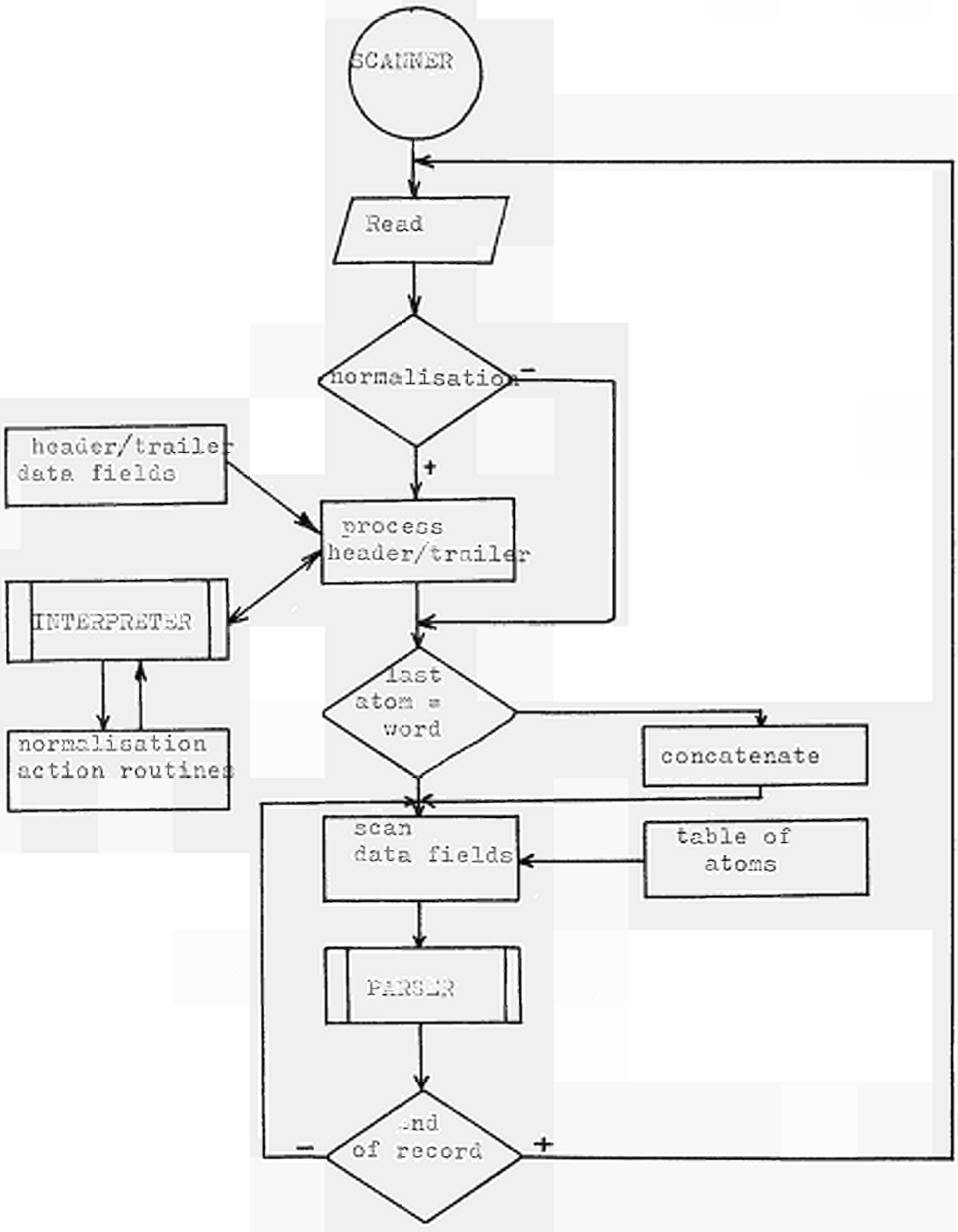


Fig. 1 - Logical Flow Chart of "Scanner"

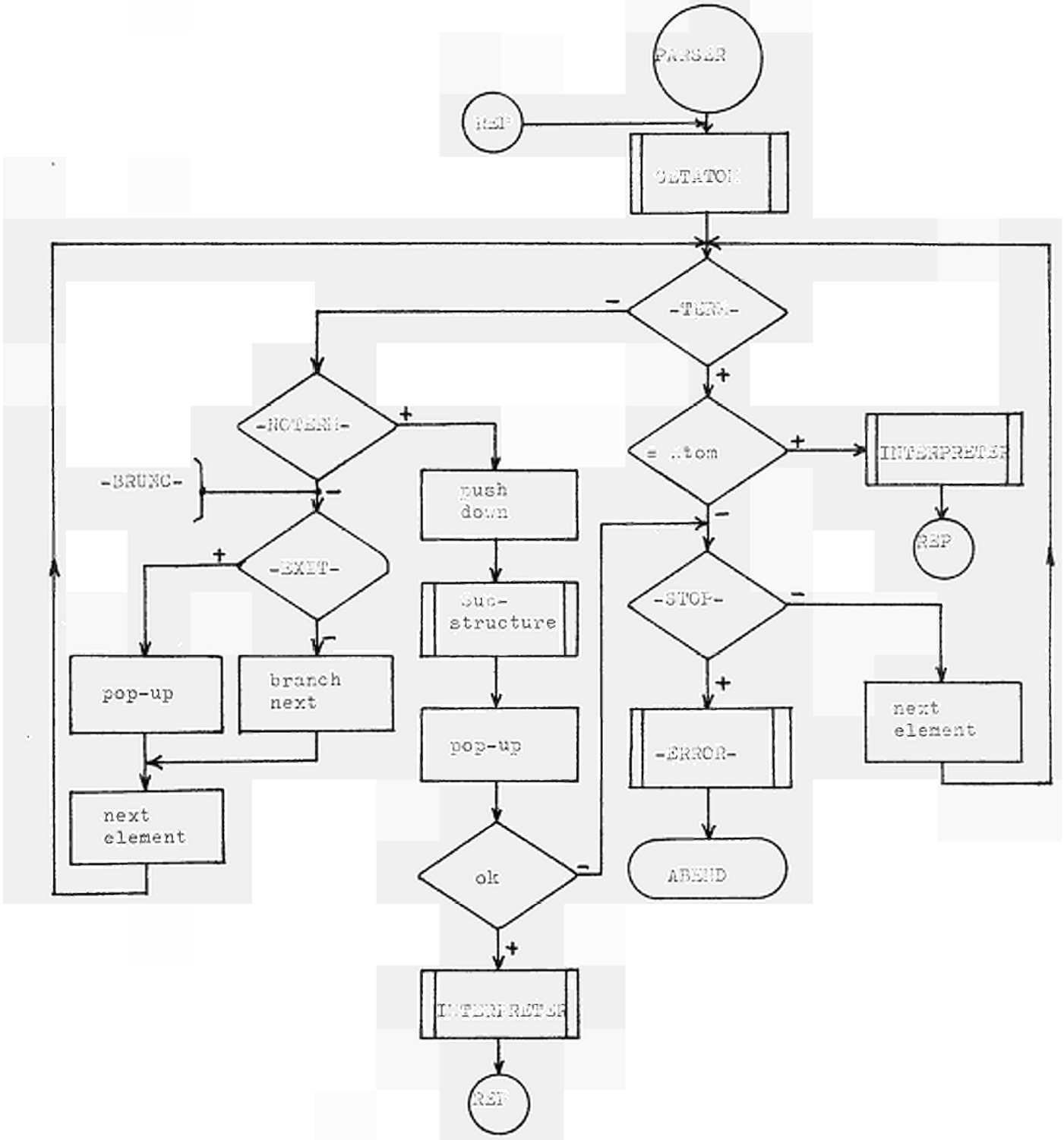


Fig. 2 - Logical Flow Chart of "Parser"

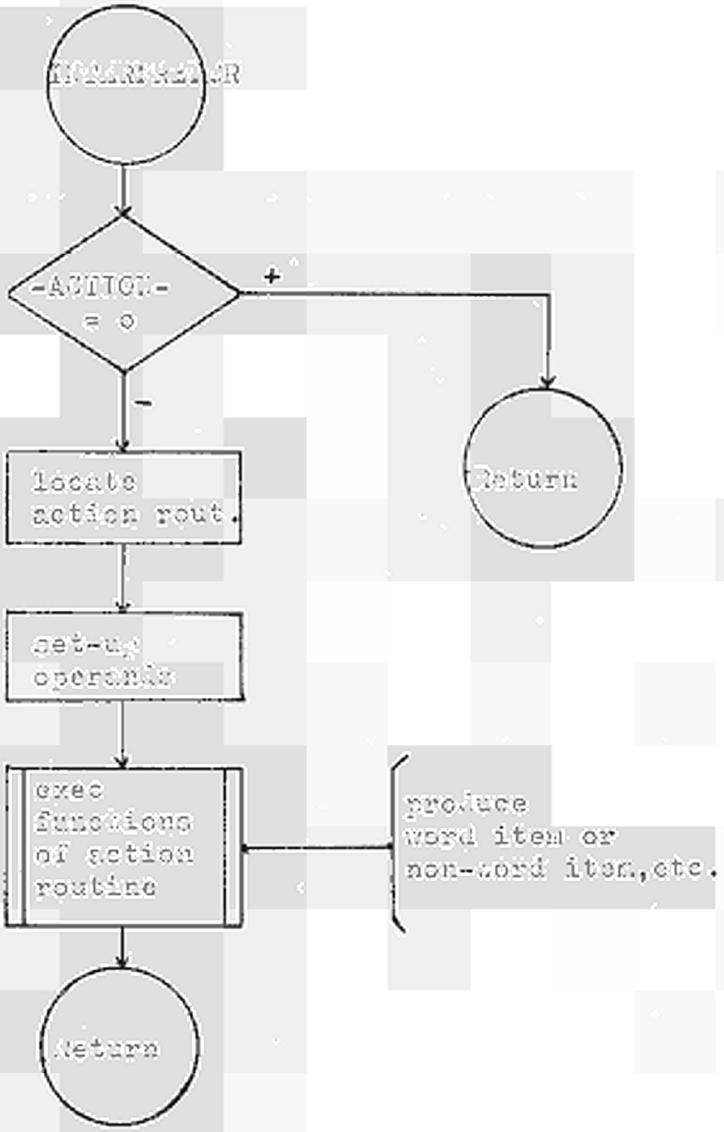


Fig. 3 - Logical Flow Chart of "Interpreter"

3.1.2.2 - Scanner (Fig. 1)

The grammar which controls the scanner has the following components:

1. The identification of the header, trailer and data fields on the record. Processing of header and trailer fields is optional and is invoked asynchronously each time a new record is read. This facility was created to enable the system to process text collections coded according to traditional punched-cards convention with text identification fields (like document number, category code, card sequence number etc.).
   The grammar describes these fields as one or more fixed-lengths strings. The scanner fills each substring and invokes the header/trailer processing action routine.
   The data fields are considered to be a continuous string (ignoring header and trailer fields). In certain conditions, the scanner must connect the rest of the precedent record with the beginning of the new record.

2. The Table of "atoms"

   The grammar can define up to 256 different atoms which correspond to the possible values of a byte. Functionally, four classes of atoms are distinguished:

   a) Individual atoms: single characters which have a special meaning for the parser and appear in the syntax tree. The scanner passes them as one-character-long strings to the parser.
   b) End of record atom: This special character which is placed outside the data field is never passed to the parser. It is used internally by the scanner and invokes, according to the status, reading of a new record, eventual header/trailer field processing and word concatenation.
   c) Blank atom: This atom functionally is very similar to individual atoms. The only difference consists in the fact that the scanner processes immediately all consecutive blank atoms and passes to the parser only one atom with its length.

d) Word atom: This atom is defined by default, and comprises all character strings which do not contain any of the other atoms. As the "word atoms" are the only ones with variable length, the concatenation of end-of-record to beginning-of-record only takes place, if the data field terminates on a word atom.

3.1.2.3 - Parser (Fig. 2)

The grammar which controls the parser represents the syntax of the strings being processed, and conceptually can be represented as a tree structure. Each node of the tree structure may be represented either by a "terminal", i.e. an atom issued by the scanner, or by an entry point to a substructure ("non-terminal"). All nodes at the same level of the syntactic tree concern the same atom. This means that the "GETATOM" function of the scanner is invoked only, if a "terminal element" of some level has been matched and one goes down to a lower level in the path.

One can associate an action routine to each node of the tree structure. The action routines are activated each time the conditions set by the terminal or non-terminal are met. If one arrives at the last element of the tree at some level without matching the conditions required, there are two possible alternatives:

- to branch to some other node of the tree structure,
- to activate an error diagnostic and recovery routine.

The syntax tree is represented by a table of elements with the following structure:

TYPE	ACTION	NEXT	STOP	ERROR	EXIT	TERM/NOTERM

TYPE: TERM - the node is considered matched if the atom issued by scanner is the one defined in the last field,

NOTERM - pointer to a substructure defined in the last field,

BRUNC - branch unconditional to element defined in NEXT

ACTION: action routine to be activated by the interpreter if the conditions of the element are met,

NEXT : address of the next node to enter if the conditions of the present node are met,

STOP : flag: last element in the node,

ERROR : error code which appears in the diagnostic message,

EXIT : flag which causes the return from a substructure which did not meet the conditions,

TERM/
NOTERM: atom identification code if type = TERM
substructure entry point if type = NOTERM

As the ERROR exit at the syntactic level only causes the printing of a message with the error code indicated and an abnormal termination of the job step, it is advisable to limit its use to unrecoverable errors and to provide error recovery action routines on the semantic level for less serious errors.

3.1.2.4 - Interpreter (Fig. 3)

Each time the conditions described by an element (terminal or non-terminal) are met, and the action field of the element is non null, the parser passes the code of the action routine to the interpreter which locates the action routine and executes the operations requested.
The action routines have the function of processing the atoms obtained from the scanner and parser, to manipulate them if necessary, to qualify them and to pass them as word items or non-word items to the system. Further, one can program particular error diagnostic and recovery routines which do not terminate the job. The instruction set prepared for the action routines, hence, is subdivided into the following classes:

- string manipulation (set string, concatenate strings, divide a string, transcode, etc.),
- control of strings (table look-up, compare, etc.),
- qualification (set word item/non-word item, send, etc.),
- general program control (return, branch, counter operation, switch operations, etc.),
- error recovery (error message printing),
- debugging aids (trace).

3.1.3 - Output

The elements established by the text analysis module are processed as follows. The text image is represented by a list which contains an entry for each element: for word items, the list contains a pointer to the next-level list, while non-word items are recorded in place. (TEXT TABLE).

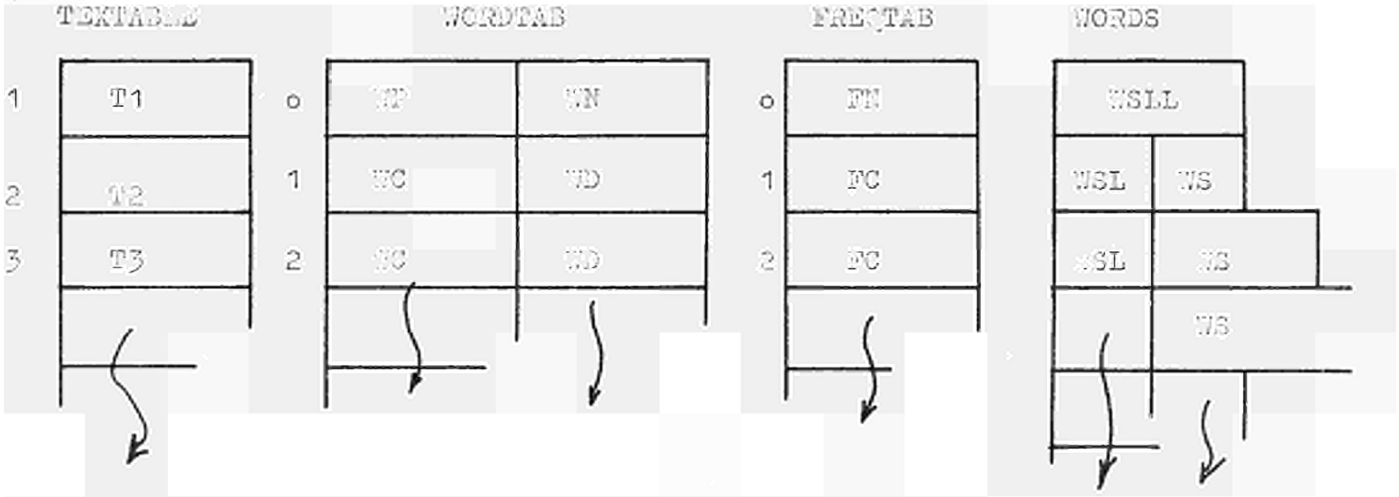
A second list is built for each different word item. It is arranged physically according to the order of occurrence of the items, and at the end of the input phase each entry contains the sequence number of the word in the SLC-alphabetic sequence (WORDTAB). (1st character in ascending order - length in descending order - alphabetic order of rest of equal length).

Optionally, a frequency count for all word items can be requested. In this case, a table of the same dimension as WORDTAB is built and contains the frequency of each word (FREQTAB). The single word items are recorded in the SLC sequence and constitute the input to the dictionary search.

3.1.4 - Examples

3.1.4.1 - Example of use of tables

Lay-out of tables:



where:
- T1,T2,.. = Pt(WORDTAB)/Non-word item
- WP = Pointer to first entry in chain
- WN = Number of used entries
- WC = Chain field
- WD = Disp(WORDS)
- FN = Number of used entries
- FC = Frequency counter
- WSLL = LL-1(space occupied)
- WSL = LL-1(word)
- WS = Word

Input text:

THE WORD ITEMS ARE SORTED ACCORDING TO THE SLC SEQUENCE. THE AMOUNT OF TEXT WHICH CAN BE PROCESSED IN ONE CYCLE DEPENDS ON THE NUMBER OF DIFFERENT WORD ITEMS WHICH CAN BE KEPT IN CORE STORAGE.

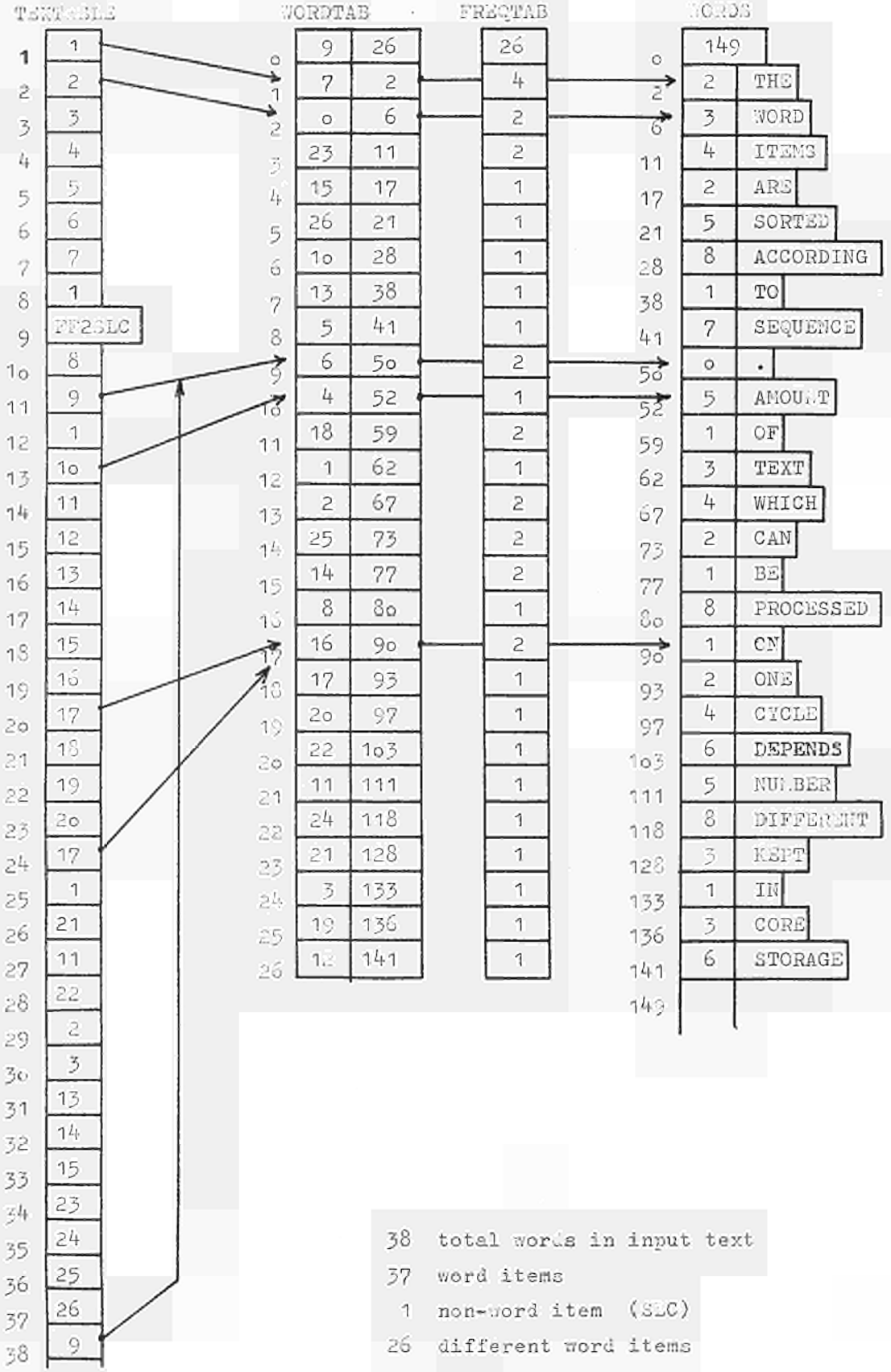


Fig. 4 - Example of use of tables

3.1.4.2 - Examples of syntax

INPUT: Texts written in Russian

USED NOTATION: BACKUS-NAUR FORM (B-NF)

< Input text > : : = < WORD > | < DOLLAR > | < HYPHEN > | MBLANK

< WORD > : : = WORD { MBLANK | $ MBLANK | $WORD |
- WORD | -MBLANK WORD [MBLANK | $ | -] }

< DOLLAR > : : = $ [WORD | MBLANK | $ | -]

< HYPHEN > : : = - [MBLANK | $ | WORD]

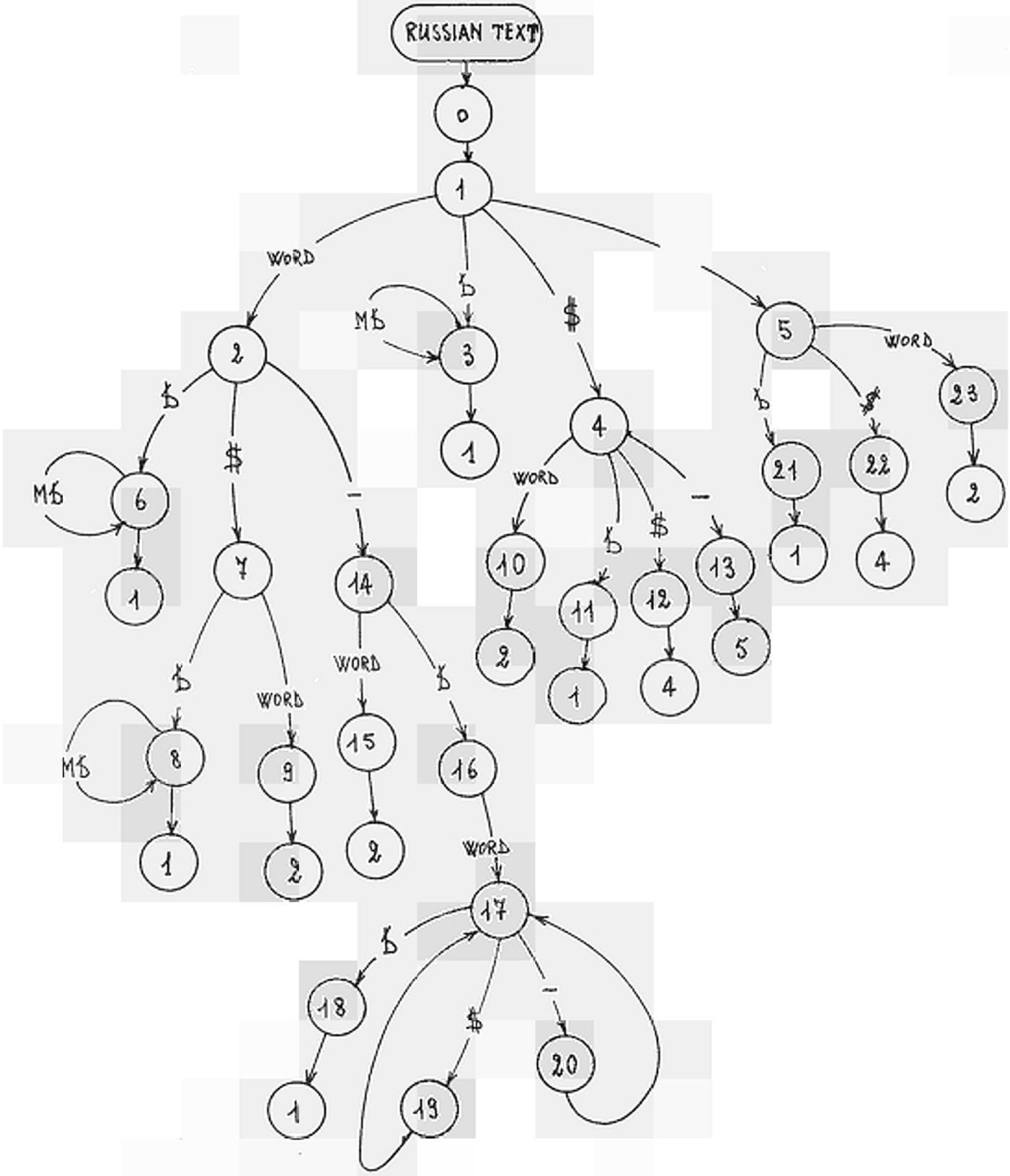


Fig. 5 - Syntactical tree for the analysis of Russian texts

3.1.4.3 - Example of action routine

AR8	SORT	UP, NAMSTR	sort characters of string NAMSTR in ascending order (UP)
	TRANSC	NAMSTR, TAB	substitution of characters of NAMSTR with string pointed at in TAB - table name
	SUB	NAMSTR, COMMA	if name string terminates in string defined by COMMA - delete sub-string in NAMSTR
	SETSTR	C, NSTR, NAMSTR	construct the string NSTR using the CHARACTER STRING (option C) of NAMSTR
	SETWJ	STR	set ON a flag of string STR
	OFFSW	SWITCH	set OFF the switch SWITCH
	LOOKTAB	NSTR, WITAB, AR92	search in a table the address of which is in WIDTAB a string equal to NSTR. If one finds it, the control passes to AR92
	SETNWI	STR	set ON a flag of string STR
	ONSW	SWITCH	set ON the switch SWITCH
	LOOKTAB	NSTR, NWITAB, AR92	see the precedent comment
	ERROR	4	print the 4th label of "Message Dictionary"
	RETURN	ER11	INTERPRETER passes the control to PARSER that continues with scan of ER11 element

3.2 Dictionary Search

3.2.1 Functions

This is the second cycle in the SLC-II System. Its purpose is to look-up the graphemes identified as "word items", i.e. as elements of the source language in a dictionary, and to perform the morphological analysis. The result of this operation is the description of the word item as a lexeme and its "inflectional" form.

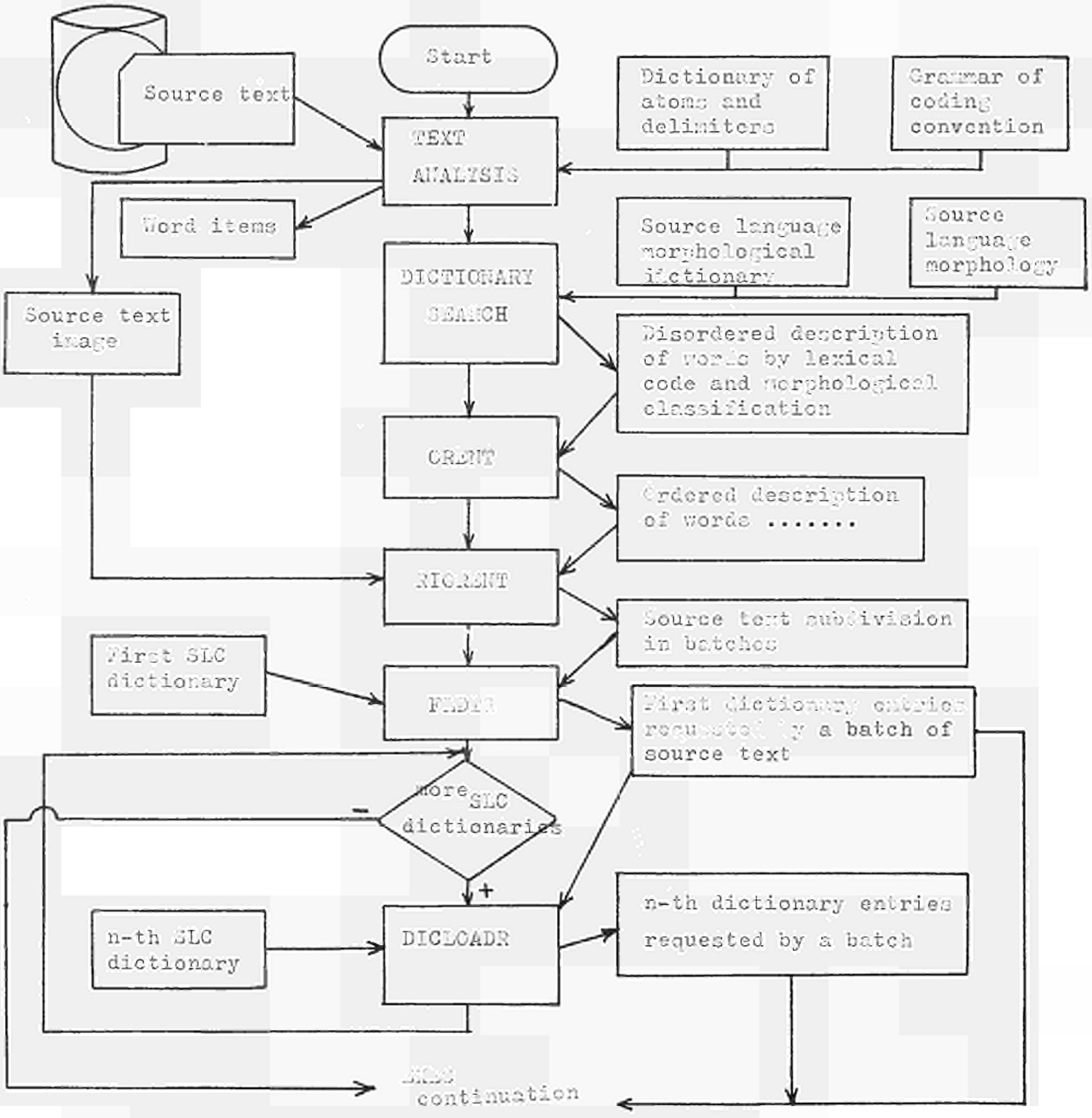


Fig. 6 - Logical Flow Chart of Dictionary Search

The dictionary used - the "source language morphological search dictionary" - has been conceived as a stem-affix dictionary. The stem, in this context, is the invariant part of a lexeme respect to the morphological forms it can assume. In the case where (with irregular inflections, c.f. go - went) no invariant portion can be found, in order to avoid nil-stems, more than one entry of the dictionary can be associated with the same lexeme. This is one of the reasons why, in the system, the different logical parts of the dictionary have been physically separated and logically linked to each other through the lexeme identification code (LXN).

The grammar used is the source language morphology organized as paradigm tables. These paradigm tables, for more or less regular inflections, can be "general", i.e. grouped in a common grammar module. For irregular inflections, they may be directly associated to the respective dictionary entry ("built-in paradigms").

No particular efforts were made to formalize even regular phenomena of the transformation of the root due to inflection (such as palatalization in slavonic languages or "Ablaut/Umlaut" in germanic languages). It was felt rather that, from the point of view of search efficiency and coding effort, it was preferable to use the facility of built-in paradigms or to introduce several entries to cover one sole lexeme.
From the functional point of view, the dictionary search module was conceived to handle the following problems:

- word inflection through suffix analysis (c.f. puella, puellae, puellam, etc.),
- word derivation through suffix analysis (c.f. Marx - Marxism - Marxistic, etc.),
- analysis of compound words (c.f. Bahnhofsvorsteherwitwenpensionsantragsformular),
- analysis of prefixes (c.f. scientific - unscientific - pseudoscientific),

- detection of homographs (c.f. "can"),
- tentative suffix analysis of words not found in dictionary (NID).

It is up to the system designer to decide which of the above functions are actually to be used in some particular application. It should be remembered here that the detection of homographs is limited to the lexeme level. Morphological homographs are not handled by the program, and the ambiguity of an inflectional form (c.f. put - put - put) must be foreseen by the linguist and associated to one sole paradigm entry.

3.2.2 The Source Language Morphological Search Dictionary and Access to It

The dictionary is accessed by the stems which are sorted, as the word items, in SLC sequence: 1st character - length - rest. This sequence was chosen primarily in view of word segmentation and prefix analysis to avoid nonsense segmentations.

To accelerate the first access to the dictionary, a three-level directory is placed in front of it: 1st character - length - end of block. Further, to accelerate search in groups of entries with the same 1st character and length, a list of pointers is placed in front of each separate group to permit a binary search (entries are variable in length).

For the subsequent accesses, each entry contains two pointers:

- the first pointer is used if no stem contained completely in the word item was found in a particular group. To limit search time, each entry contains the address of a subsequent entry of lower length, which is smaller than or equal to the present, but larger than or equal to the precedent entry. Match between word items and dictionary entries is from left to right for the length of the stem, and in principle the search is sequential. When, during the scan, the stem becomes larger than the corresponding

portion of the word item, scan is discontinued and resumed at the entry pointed at;

- the second pointer contains the address of the next entry completely contained in the present one (same length or shorter). It is used, when a full match between a stem and a word item already has occurred. In the case where the morphological analysis has been successful, it is used to detect eventual homographies. If it has failed, to retry analysis.

If the reference (1st or 2nd depending on the status of search) is nil, and no match has occurred, the word in question is unknown - not in dictionary (NID).

3.2.3 The Source Language Morphological Paradigms and Morphological Analysis

Each dictionary entry contains either a pointer to a general paradigm or a built-in paradigm. The paradigms contain the suffixes which a stem can be given and the description of the meaning of the suffix. For economy purposes, suffixes can be subdivided into a chain of morphemes, each of which can be looked up in a separate paradigm. Thus, in highly inflecting languages like Russian, a word like KRASOVAVWIMIS4 can be analysed as KRAS-OVA-VW-IMI-S4

OVA - Infinitive stem formant,
VW - past active participle,
IMI - instrumental plural,
S4 - reflexive.

The description of the form is given by a binary vector (HWO), whose contents and length can be defined symbolically by a subset of the SLC-II programming language.

If morphemes are chained, the vectors associated to each suffix are

added logically.

The remaining functions of morphological analysis depart from particular situations created in suffix analysis.

Word derivation analysis is started, if the binary vector associated to the suffix (HWO) has a particular configuration, communicated to the system as an input parameter, and there is a chain reference in the entry to another paradigm. In the derivation algorithm, a binary vector is associated to each level of derivation.

Word segmentation is started if the match is incomplete, the paradigm entry has no chain reference and the binary vector associated has a particular configuration communicated to the system as an input parameter. Segmentation takes place only if no full match has previously been found. As it is uncertain that it will also be found, an artificial homography between the full word as NID and the segmentation is created, which is resolved at the end of dictionary search (see ORENT). The rest of the word is considered to be an autonomous word and looked up in the dictionary separately.

Prefix analysis is a particular case of word segmentation. In the dictionary search program no distinction is made between compound words and prefixed words. It is up to the source language dictionary and the problem program to distinguish these cases.

Multiple Matches (homographs) are detected when the chain of the entries linked to each other by the second pointer is followed to the end. The detection of homographies can optionally be excluded by an input parameter. In this case search is interrupted after the first full match.

Analysis of words not found in the dictionary (NID). If the chain of the first or second pointers terminates without any full match, the word is consi-

dered to be unknown, and, subsequently, is treated in the same way as are non-word items. Optionally NID can be analyzed with a special paradigm table which might permit, especially for highly inflected languages, to determine the grammatical properties of the word. This option may be useful especially in applications as machine translation, as the unknown word, at least, may be correctly inserted at the surface syntax level.

3.2.4 Realization of the Dictionary Search Program

Especially with large dictionaries and many word items to look up, dictionary search becomes a prevalently I/O bound problem. The design of the over-all search strategy aimed at an optimal exploitation of the resources in a batch-processing environment, but took into account also the future destination of the modules for interactive operation.

The solution chosen ensures a good balance between the I/O operations and the dynamic main storage management and CPU occupation.

The following chapters describe concisely the program organization and the general program and data organization logic.

The problem of balancing the resources is given by the presence of two large data sets which, in a certain sense, control each other's I/O operations: on one hand the words which request the input of certain dictionary entries, on the other hand the dictionary which is read according to the requests issued.

As both words and the dictionary are sorted by the same sequence, their I/O should, in principle, be sequential. However, for the dictionary, there may be two sorts of exceptions:

- blocks of entries may not be requested at all, and should be skipped;

- especially the word segmentation option may request the return to a previously processed point of the dictionary.

For the words, one must keep trace of those already processed in a block so as to be able to read the next block when all requests have been treated.

The problem was resolved by the definition of a series of control lists which are inter-dependent.
The words are read sequentially into core storage and scanned one by one against the dictionary master directory (resident during dictionary search) to determine the initial request for scanning the dictionary (see Fig. 7).

The buffers for input words are controlled by a list called AERTAB in the program, which contains the buffer address, the status of I/O and request operations (I/O operation termination checked or not, all requests for search issued or not), and a counter of requests still to be satisfied. Each time when search for a word is terminated, the counter is reduced by one, and, eventually, a new request is issued. When all requests relative to a word buffer are satisfied, it is free for the next block. The amount of buffers and relative AERTAB entries is dynamically allocated depending on the main storage available and the space occupied by the words.

The outstanding requests are controlled by a list called REQTAB which contains one entry for each word processed. It contains the word address, the number of the buffer, the request (in terms of track and offset), the word number and a code which permits to chain homographs, segments etc. after search. Further it contains the search status information flags which are consulted during search. The entries are sorted by ascending track number.

Each time a new request has been issued (see Fig. 8), the program checks whether the relative track has already been requested by means of a list called TABRT, which controls all such requests and records the address of the first REQTAB entry which issued it. Each time a new track is requested, it is inserted into TABRT, and one checks whether it permits to continue sequential reading. The sequential reading of the dictionary and the relative buffers are controlled by a list (RDDCTB). As there may be more sequential read requests than there are buffers, an auxiliary list (RQDCTB) handles all outstanding requests. Direct access operations are only performed if no sequential I/O is possible. When a read operation of a track requested is completed, one locates the REQTAB entry which first requested the track and processes subsequently all entries which requested it. The links to the subsequent accesses are processed immediately if they point to the same track, and queued for later processing if the track changes.
When all requests for a given track have been processed, the buffer is free for input of the next track.

The offset field in the request may point either to an individual entry or to the binary search directory (see Fig. 9).

In the latter case, a binary search is performed before starting sequential scanning, locating the first entry equal to or lower than the word.

In the sequential scan, one compares the beginning of the word with the stem by the length of the stem. If the stem is smaller, one continues to search at the next sequential entry. If the stem is larger, one uses the first pointer and either continues scan or queues the request depending on whether the track changes or not. If the stem is equal to the beginning of the word, one considers the rest of the word (the portion beyond the length of the stem) to be the potential ending and calls the morphological analysis routines. At the return, one uses the second pointer to continue

search if morphological analysis failed, or, optionally, to detect homographies. When the chain of pointers is interrupted and no match has occurred, one calls the routine which handles unknown words (NID).

The morphological analysis routine (MORPHAN) first locates the paradigm pointed at in the dictionary entry (general or built-in) and scans the entries contained in it sequentially trying to match from left to right the word ending with the suffixes contained in the paradigm (see Fig. 10). If a full match between a word ending and a suffix is found and there is no link to another paradigm in the entry, search is terminated and no attempt is made to detect morphological homographs.
If the match is partial, or there is a link in the paradigm entry, the following options are examined:

- morpheme chaining: one locates the paradigm pointed at and continues search. A full push-down mechanism is provided for the case that the morpheme chaining fails at a certain point.

- word derivation: it is detected through a particular configuration of HWO, given as a parameter. This parameter is a double representation of HWO. The first part contains a mask indicating which bits of HWO must be checked and the second, which must be on. The mechanism of word derivation is quite the same as in morpheme chaining, with the only difference that the HWO are not OR-ed to represent the result, but linked to each other as if they represented word segments.

- word segmentation: it is detected in the same way as word derivation with another input parameter (see Fig. 11). It takes place only if no full match has occurred before to avoid pseudo-homographs. The particular sort sequence makes it quite sure that a full match will not occur after the first segmentation has been performed. As it is not sure that the segmentation will be successful, an artificial homography with the full word as NID is created which will be resolved in the subsequent cycle. If prefix analysis is required, segmentation can be forced so as

to detect possible homographies (c.f. German "gehört"). For the rest of the word a search request is set up, as if it were a new word to be looked up in the dictionary. The word segments are linked to each other by conventional pointers.

The analysis of unknown words (NID) first moves the word into a special file and inserts its relative position in the file into the search result instead of the lexical number (see Fig. 11). Optionally, a list of unknown words may be produced.

As an option, which can be useful particularly in machine translation, the unknown word can be analyzed with a special paradigm table quite in the same way as in morphological analysis, with the only difference that suffix matching and morpheme chaining take place from right to left. (N.B. At attempts operating with a null-dictionary, the result is that of a suffix analysis program).

The results of dictionary search are written on a data set and contain in addition to HWO and LXN an identification of the entry given by the word number and a conventional match counter, and the eventual pointers to segments and homographs.

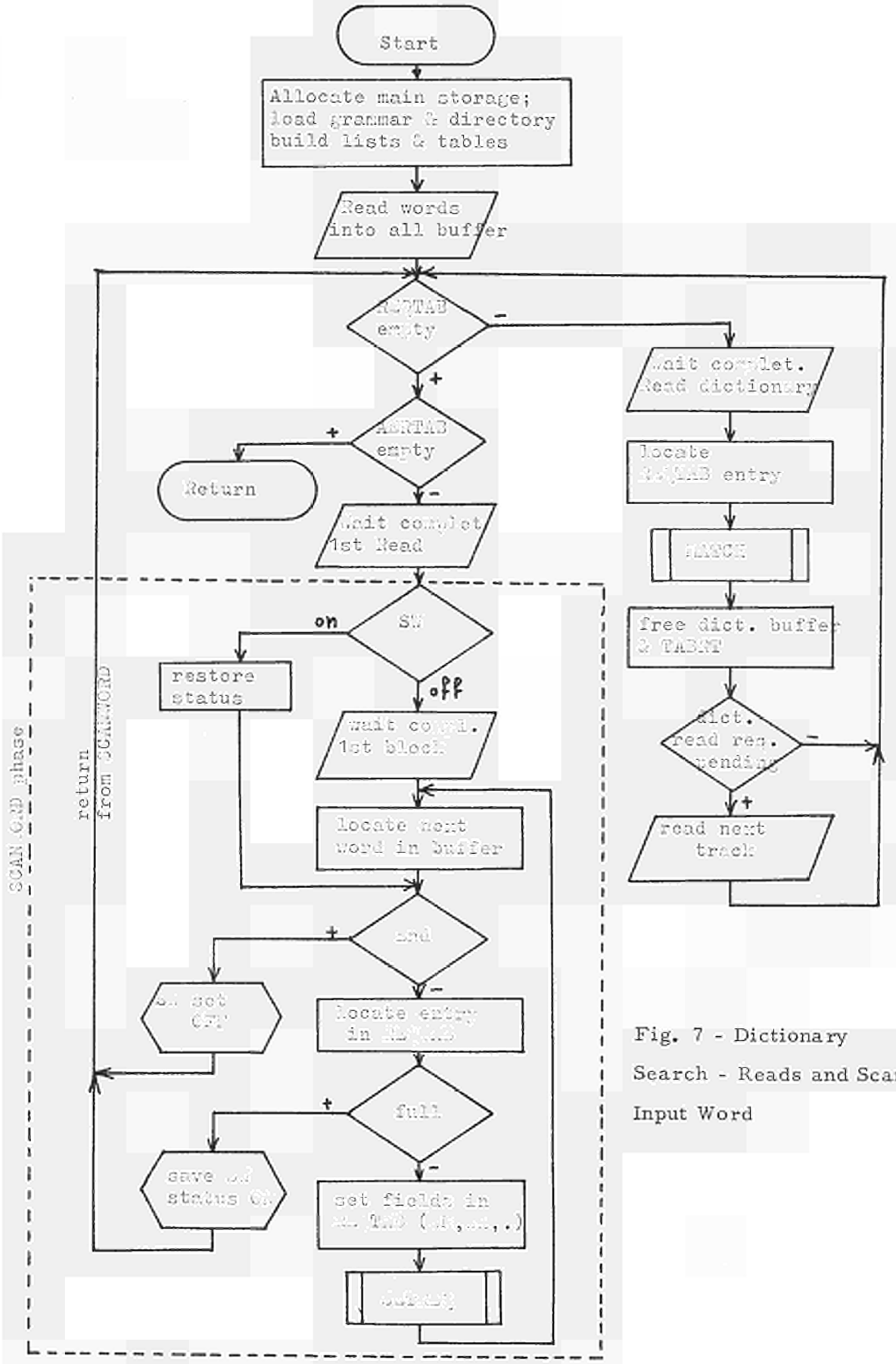


Fig. 7 - Dictionary Search - Reads and Scans Input Word

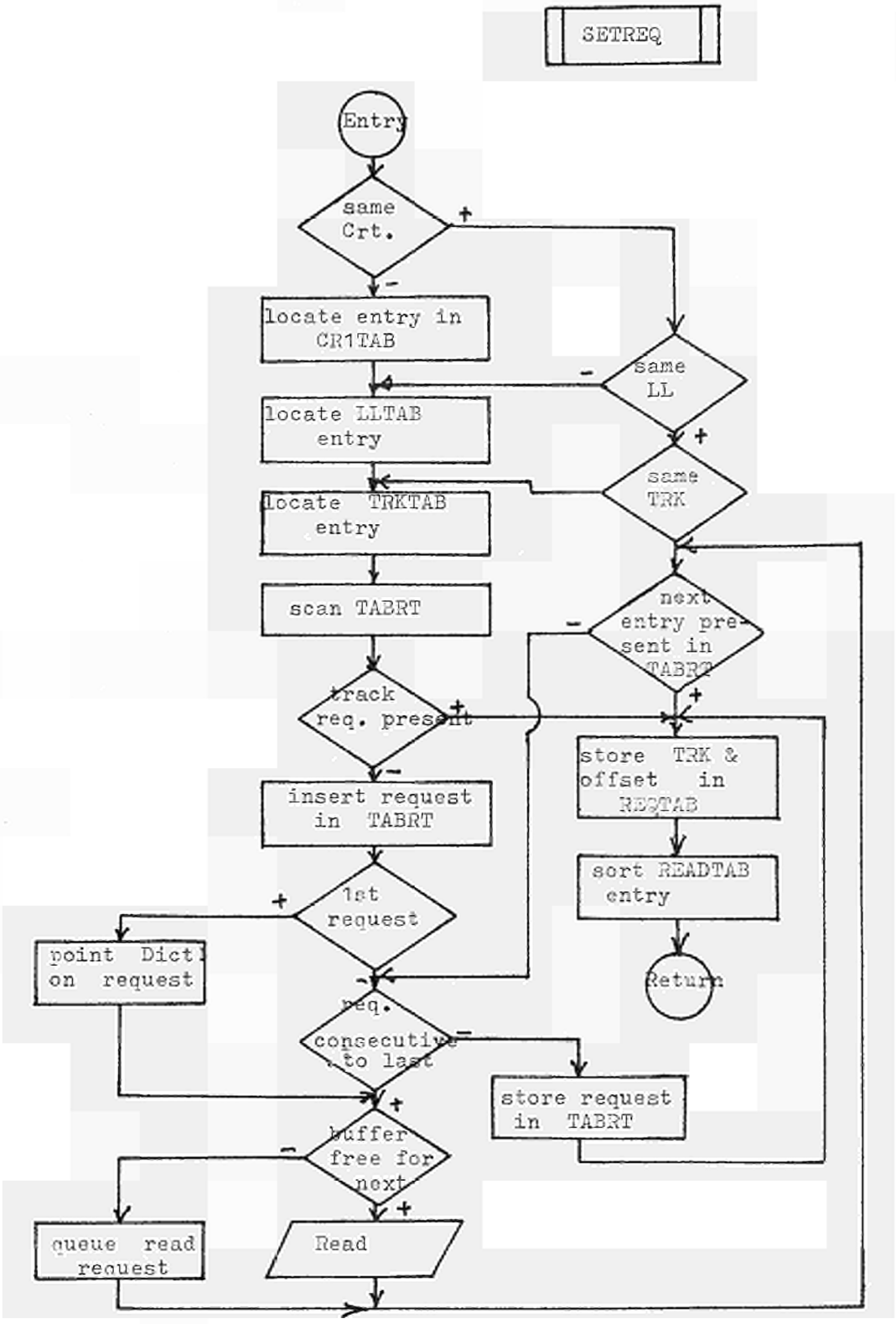


Fig. 8 - Dictionary Search - Sets Request of Words and Reads Dictionary Tracks

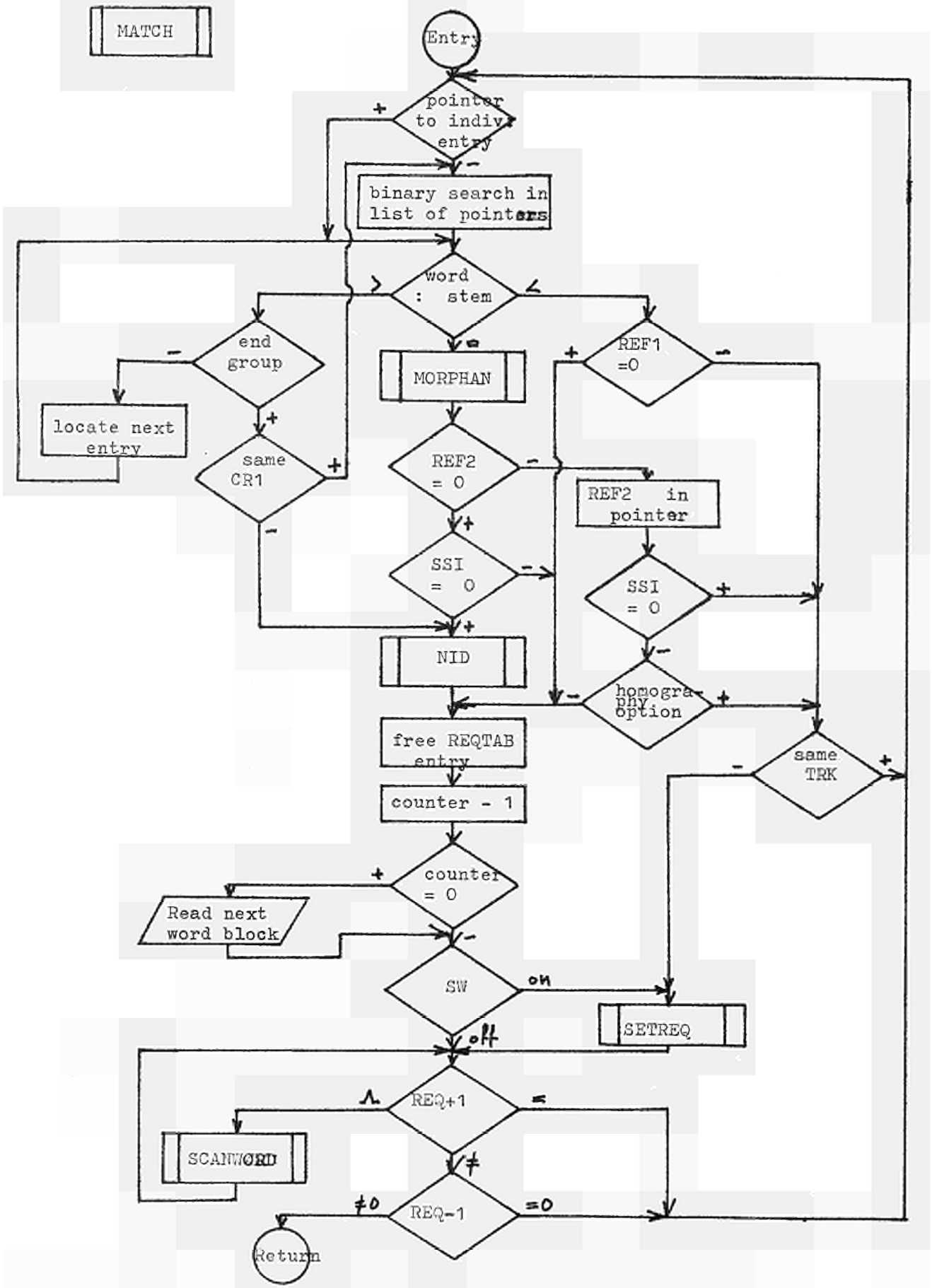


Fig. 9 - Dictionary Search - Search in Dictionary of Input Word Item

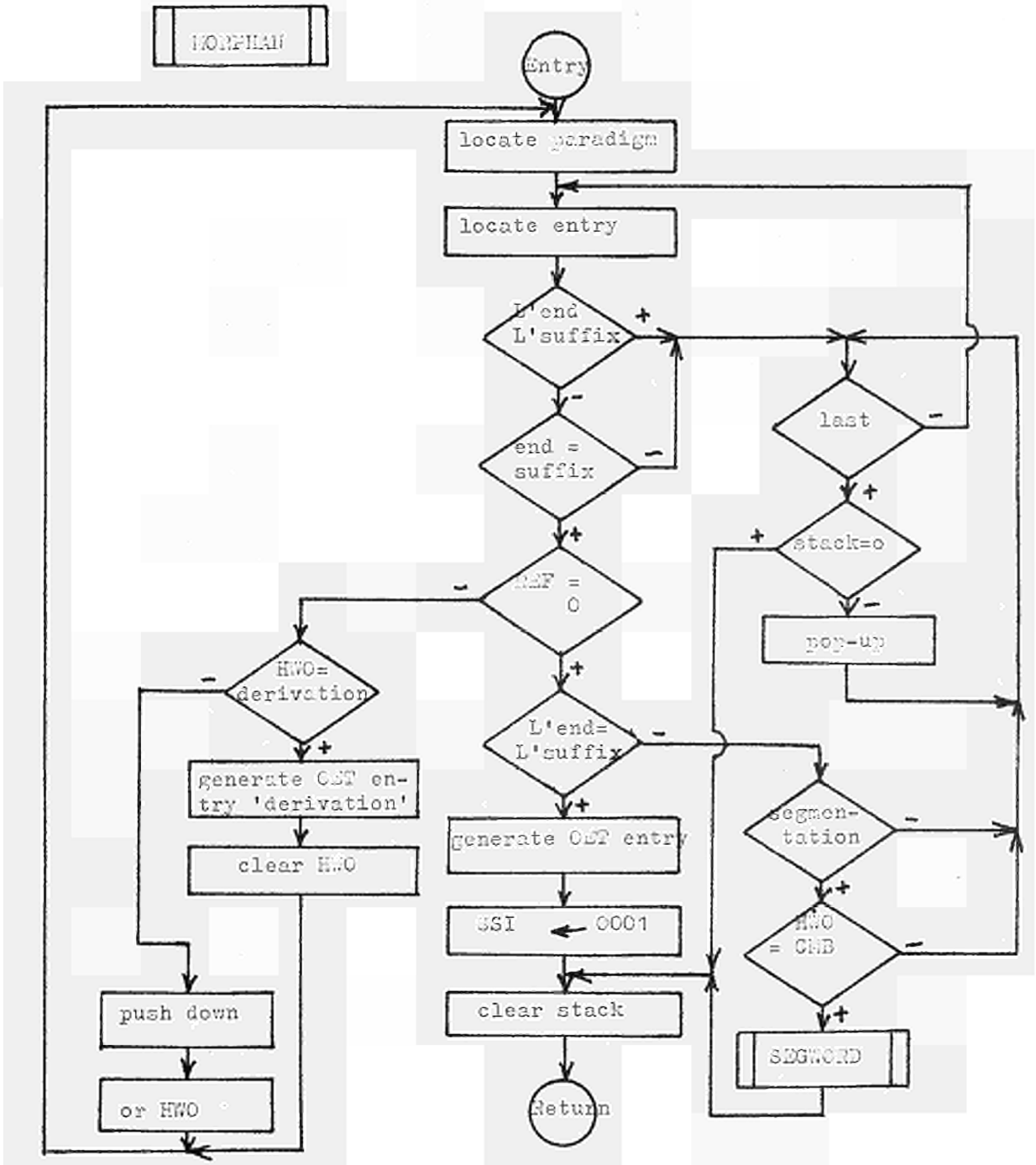


Fig. 10 - Dictionary Search - Morphological Analysis on Input Word Item

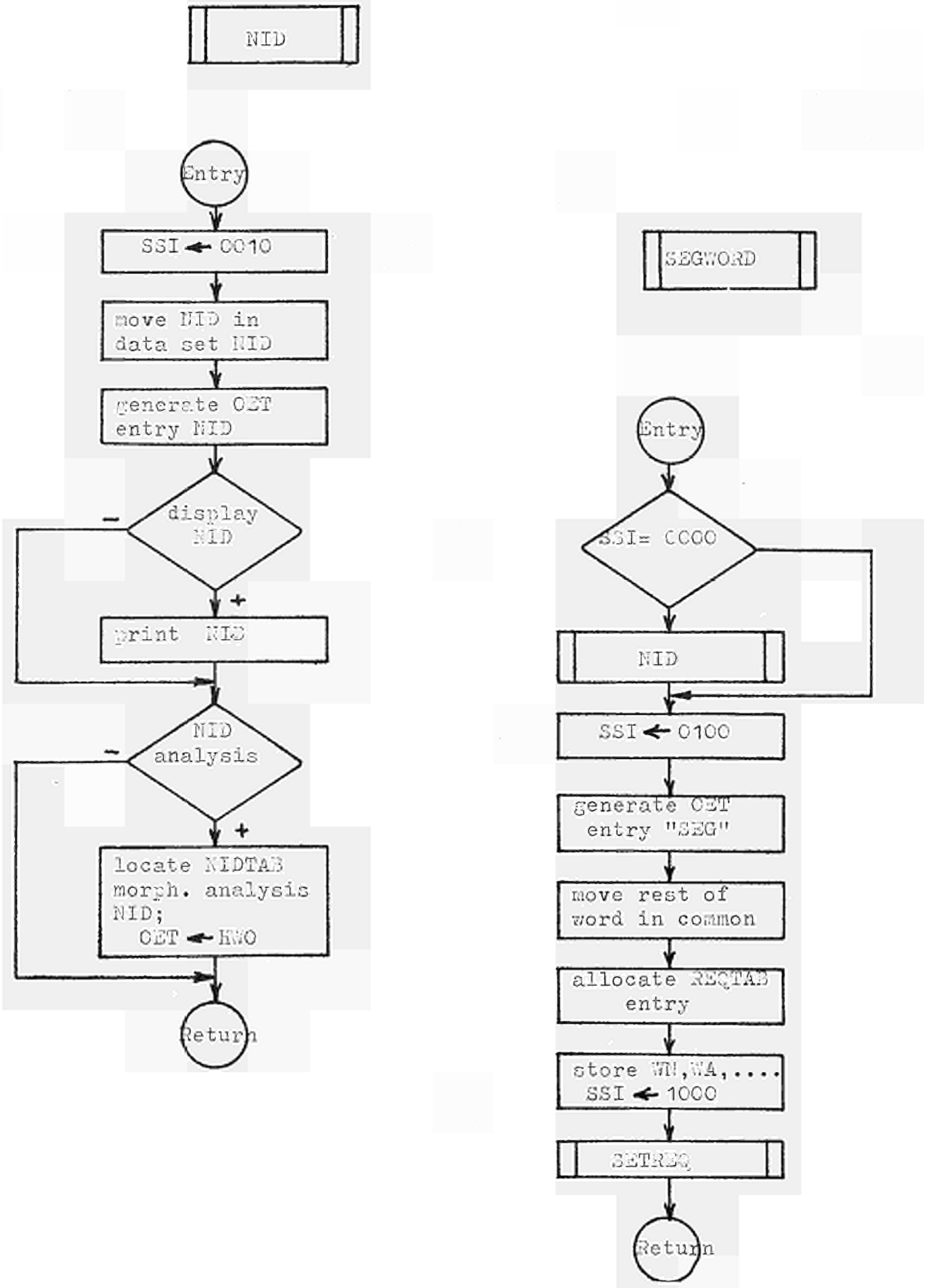


Fig. 11 - Dictionary Search - Analysis of Unknown Words and Input Word Item Segmentation

3.2.5 Organization of Dictionary Entries (ORENT)

This module is called at the end of the dictionary search and examines its results for inconsistent solutions which primarily may come from word segmentation. The results of dictionary search have been collected on a data set (OET) which for each entry has the following format:

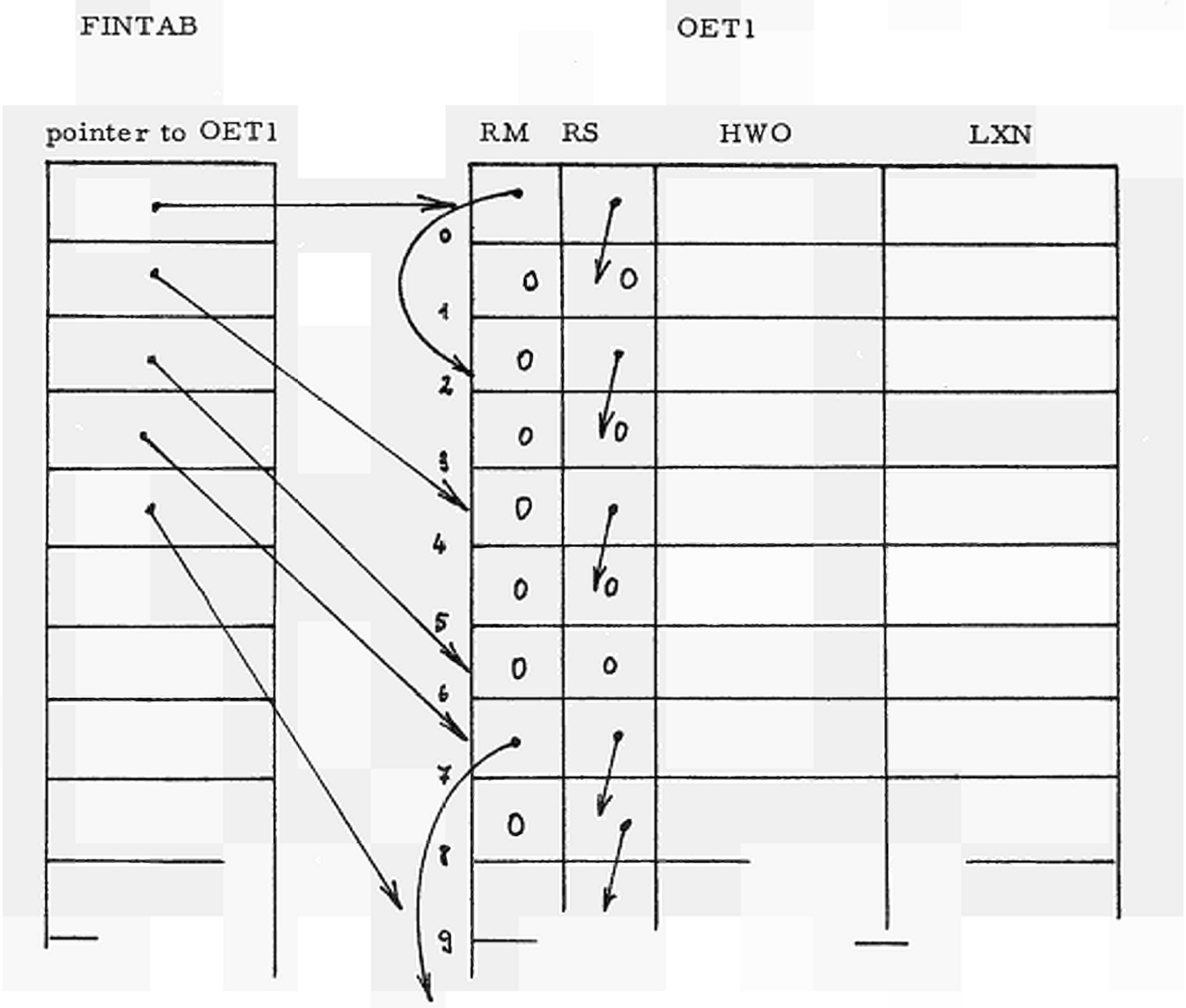
WN	CN	RM	RS	HWO	LXN

where:

WN = word item number
CN = conventional number
RM = reference match
RS = reference segment
HWO = head-word zero
LXN = lexical number

The sequence of the entries depends on the moment in which the match has occurred. The task of this module is to group physically all entries which refer to the same word item and to eliminate all inconsistent solutions.

The output of the module is a bipartite list (FINTAB & OET1) with the following structure:



The program first reads all OET entries and links them to FINTAB and by RM and RS as far as they exist. Afterwards, all possible solutions are scanned and the inconsistent ones eliminated.

The example here-below is to illustrate the process. Assume that the word was

JELEZOXROMOALHMINIEUY1

with WN = 1253, CNCT = 1560. The complete word could not be found in the dictionary, but the segmentation facility produced the following result:

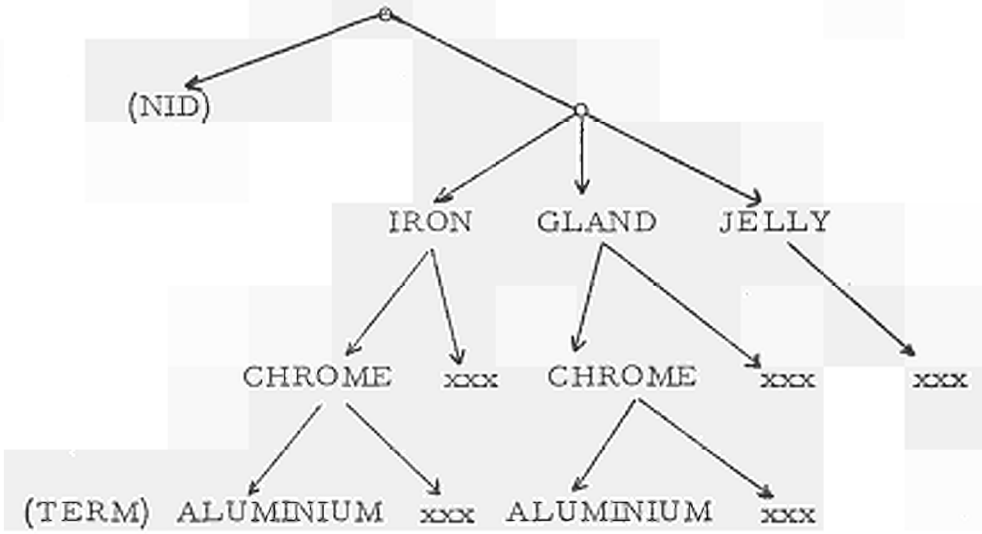
STEM NUMBER	STEM	HWO	LXN
1	JELEZ + (O)	CMB	(IRON)
2	JELEZ + (O)	CMB	(GLAND)
3	JEL + (E)	CMB	(JELLY)

REST NUMB.	REST	WN	CN	STEM	HWO
1	XROMOALHMINIEVY1	1253	1563	XROM + (O)	CMB
2	XROMOALHMINIEVY1	1253	1563	XROM + (O)	CMB
3	ZOXROMOALHMINIEVY1	1253	1567	----------	----
4	ALHMINIEVY1	1253	1569	ALHMINIEV+ (Y1)	----
5	ALHMINIEVY1	1253	1571	ALHMINIEV+ (Y1)	----

The OET corresponding to the above matches is:

	WN	CN	RM	RS	HWO	LXN
1	1253	0	1561	0	± NID ±	± NID ±
2	1253	1561	1562	1563	CMB	(IRON)
3	1253	1562	1564	1565	CMB	(GLAND)
4	1253	1564	1566	1567	CMB	(JELLY)
5	1253	1563	1568	1569	CMB	(CHROME)
6	1253	1565	1570	1571	CMB	(CHROME)
7	1253	1569	1572	----	----	ALUM....
8	1253	1571	1573	----	----	ALUM....

Logically, it can be represented by the following tree structure:

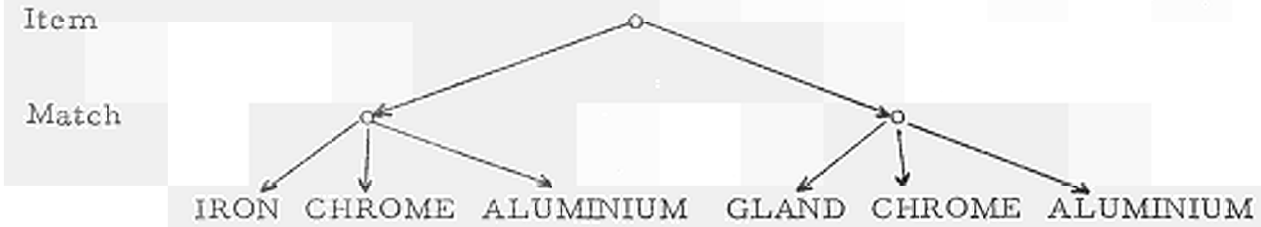


The result, after transformation, is shown here below. As one can see, the OET entries (1) corresponding to the preliminary NID analysis and (4) corresponding to "JELLY" could be eliminated, but the homography between "JELEZO" (IRON) and "JELEZA" (GLAND) persists and it is up to the problem program to resolve it.

pointer

	RM	RS	HWO	LXN
0	3	1	CMB	IRON
1	0	2	CMB	CHROME
2	0	0	CMB	ALUMINIUM
3	0	4	CMB	GLAND
4	0	5	CMB	CHROME
5	0	0	CMB	ALUMINIUM

The corresponding tree structure is:



3.2.6 Subdivision of the Source Text into Minor Batches and Dictionary Loading (RIORENT - FLDTG - DICLOADR)

3.2.6.1 - RIORENT Module

The philosophy of the input and dictionary search modules was to process in one cycle the maximum possible amount of source text so as to increase the efficiency by exploiting the repetition of words. However, if one wants to associate the dictionary information to all of the words processed in the precedent cycles, the core storage available may turn out to be insufficient to hold them.

For this reason, the source text is subdivided into a series of batches each of which can be entirely processed in core storage. The input parameters are the average length of the dictionary entries to be loaded and the amount of core storage available. These parameters are initially given by the user, but after the first cycle they are computed internally.

The amount of text which can be processed in one cycle depends on the number of different dictionary entries which must be loaded (each entry is loaded once only).

For this purpose, the TEXT TABLE is read, scanned against WORD TABLE, FINTAB and OET1, and all entries (LXN) contained in OET1 are inserted into a list with a hash code technique (TLXN).

A subset of OET1 is produced for each new word item requested (OET2); in TEXT TABLE, the pointer to the WORDTAB entry is replaced by the entry number in OET2 (a list TEXT1 is produced) and the LXN in OET2 is replaced by the entry number in TLXN (Fig. 12).

In the following example the first sentence of text presented in Text

Analysis is considered:

THE WORD ITEMS ARE SORTED ACCORDING TO THE SLC SEQUENCE.

One assumes that the result of dictionary search is:

THE	two meanings:	article, adverb
WORD	" " :	substantive, verb
ITEMS	one meaning :	substantive
ARE	" " :	verb
SORTED	" " :	verb
ACCORDING	three meanings:	adverb, verb, preposition
TO	two meanings:	preposition, adverb
SEQUENCE	one meaning ;	substantive

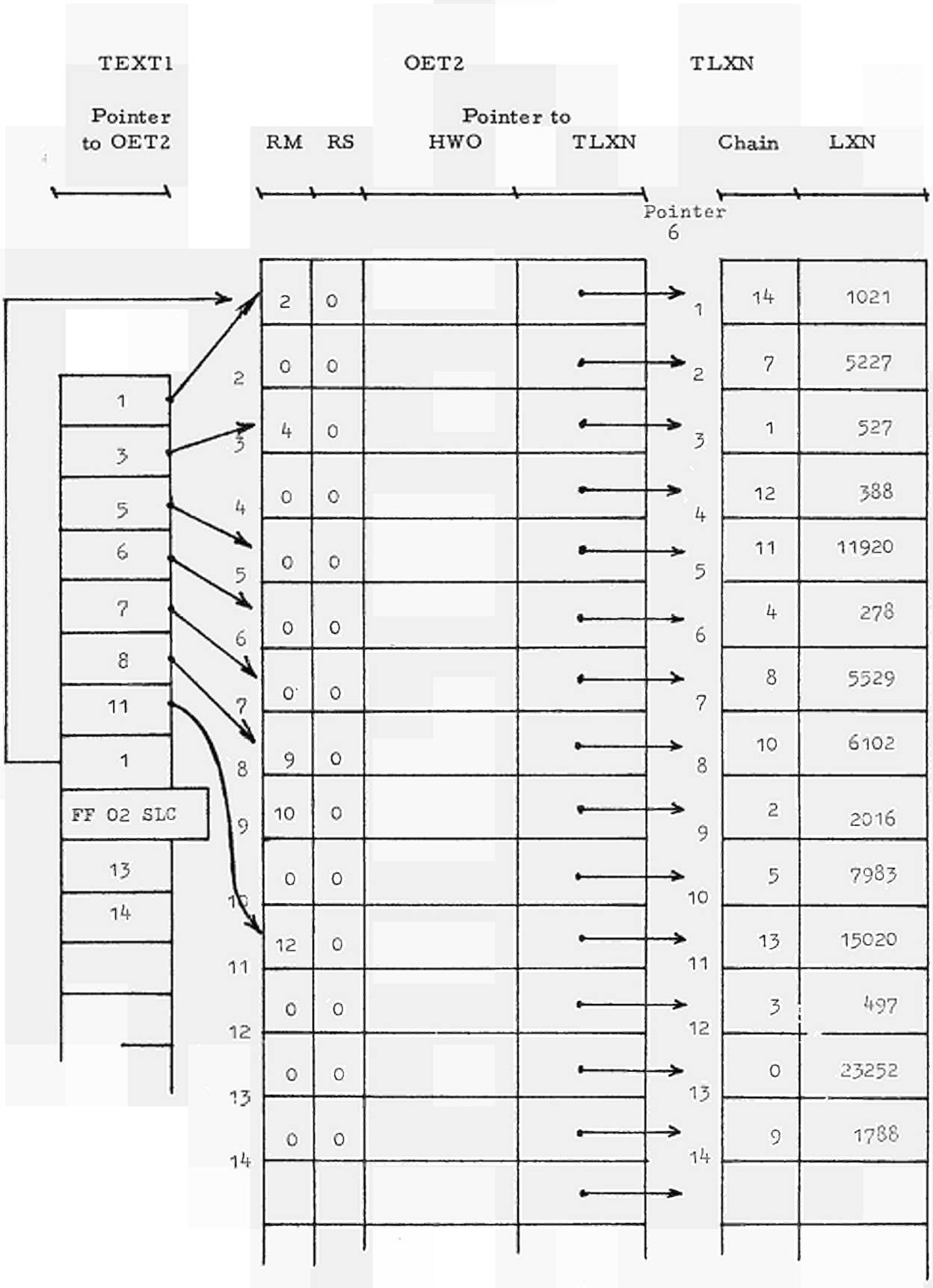


Fig. 12 - Data Structure before Dictionary Loading

Each time a new entry is requested, the length of the OET1 entry and of the dictionary entries are added. When they reach the amount of core storage available, the process is interrupted. The hash table is sorted by LXN and control is given to the next module - FLDTG.

3.2.6.2 - First Dictionary Loader (FLDTG)

The program scans the hash table (TLXN) produced in the precedent cycle, accesses the First SLC-Dictionary and loads the entries requested into core storage.

The SLC dictionaries are all structured in the same way: the first block (all dictionaries must be recorded one block per track of DASD-Disc, Drum, etc.) contains a master directory which on the first entry indicates the number of tracks occupied by the dictionary and in the subsequent entries the LXN of the last entry on the corresponding track. Dictionary entries are sorted by ascending LXN (lexeme identification code). During loading the directory is resident in core storage.

FLDTG scans the LXN contained in TLXN, locates the track number of the corresponding entries and requests input of the tracks involved in direct access mode. When a read operation has been terminated, the entries requested are moved into the core storage area reserved for them, and the LXN in TLXN are replaced by the address of the entry.

When loading has terminated, the pointer to TLXN in OET2 is replaced by the address of the entry in core storage. As a result the following data structure is produced:

N. B. One continues with example used in RIORENT

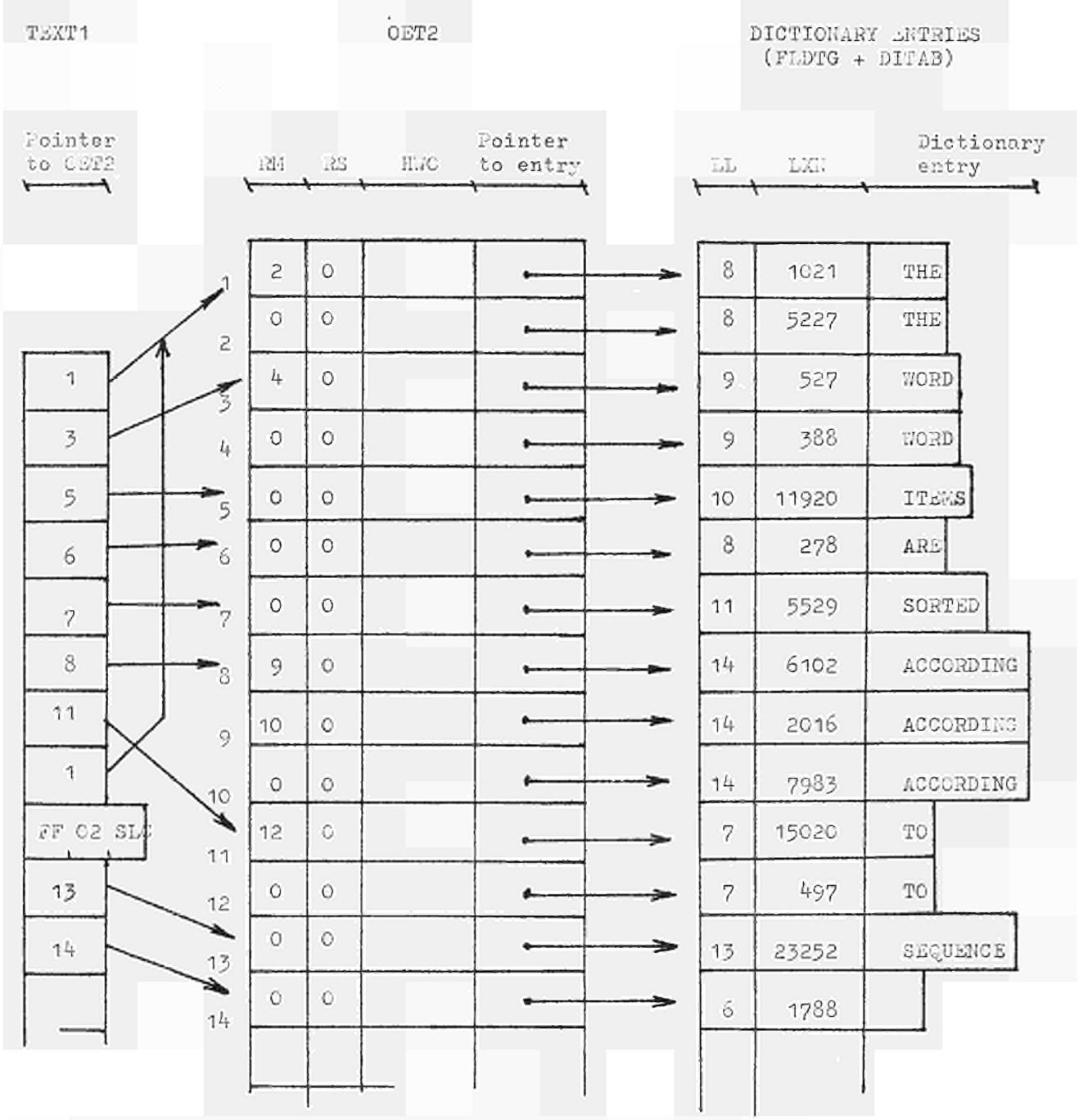


Fig. 13 - Data Structure after Loading of Dictionary Entries

3.2.6.3 - DICLOADR Module

The translation process (indexing, query formulation etc., also can be considered as special cases of translation) is, generally, subdivided into three distinct cycles - source text analysis, transfer, target text synthesis - for each of which one must define an algorithm, a grammar and at least one dictionary. In the batch mode version one of the principal objectives consists in optimizing the I/O functions. Therefore, all dictionaries requested must be loaded into core storage before control is given to the SLC-coded problem program.
SLC dictionaries, for the processor, have all the same structure:

LL	LXN	BINARY VECTORS (HW)		
		ID.C.	LL	DATA
		ID.C.	LL	DATA

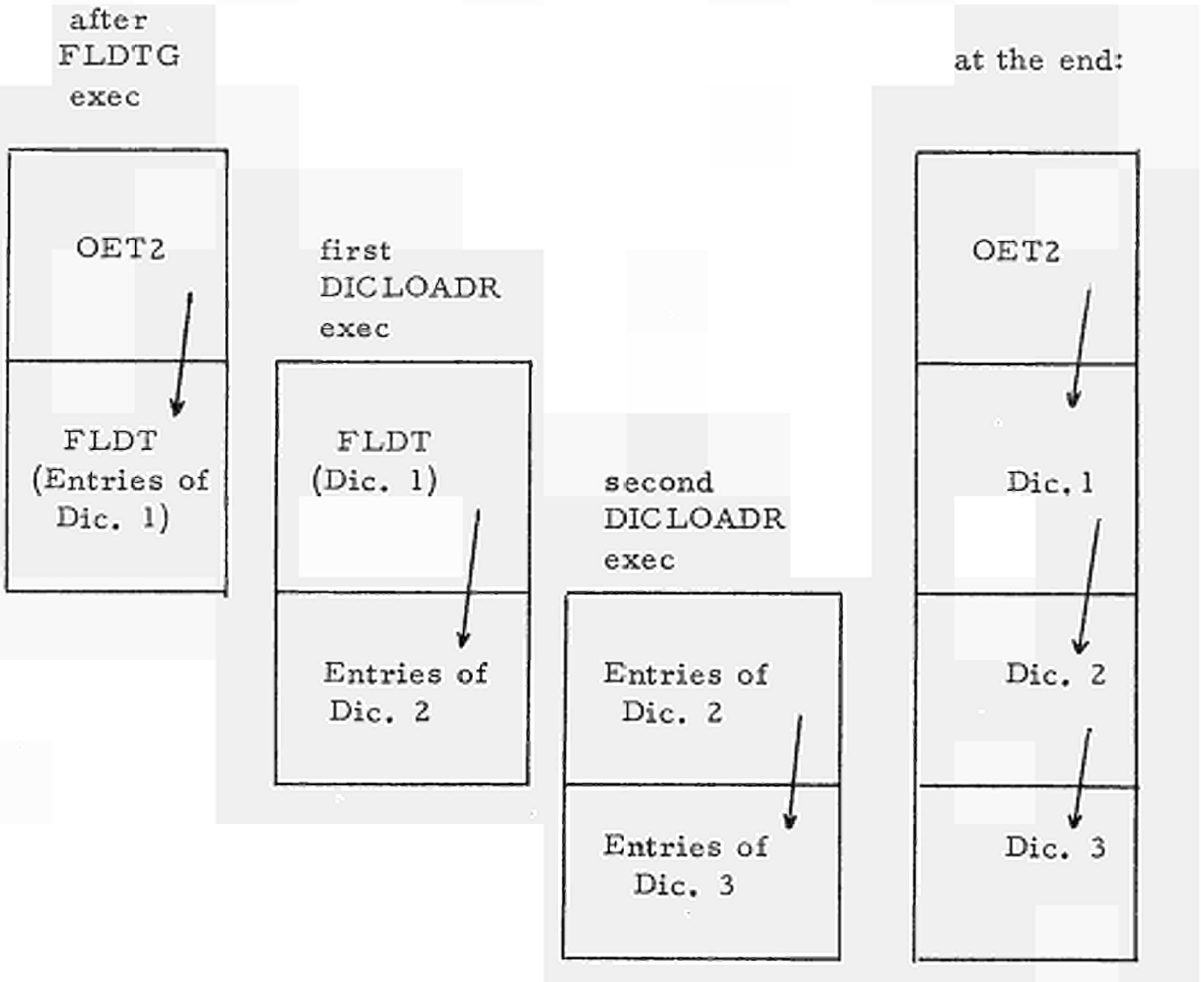
Apart from the length field and the LXN by which an entry is accessed, there may be a series of binary vectors (HW1 - HW14) whose length and identification code are parametrized at system generation time, and a series of "variable fields" whose function is identified by their identification code.

If a dictionary entry is to be linked to a subsequent dictionary, the LXN of the entry requested may be either contained in the field of binary vectors or in one or more variable fields. In the latter case, the LXN must be located in the last four bytes of the variable field.
A set of input parameters indicates in which position(s) of the binary vectors there are contained LXN, and which variable fields contain them.

The DICLOADR program assumes as input data a set of entries already

in core storage. The entries are scanned one by one and the LXN requested are processed exactly in the same way as in FLDTG. After loading of the entries requested, the LXN in the requesting dictionary are replaced by the addresses of the relative entries.
The DICLOADR is cyclic and may process any number of dictionaries. The number is communicated through an input parameter.

Schematically the function of DICLOADR can be represented as follows, assuming 3 dictionaries to be loaded:



3.3 Problem Programs Execution

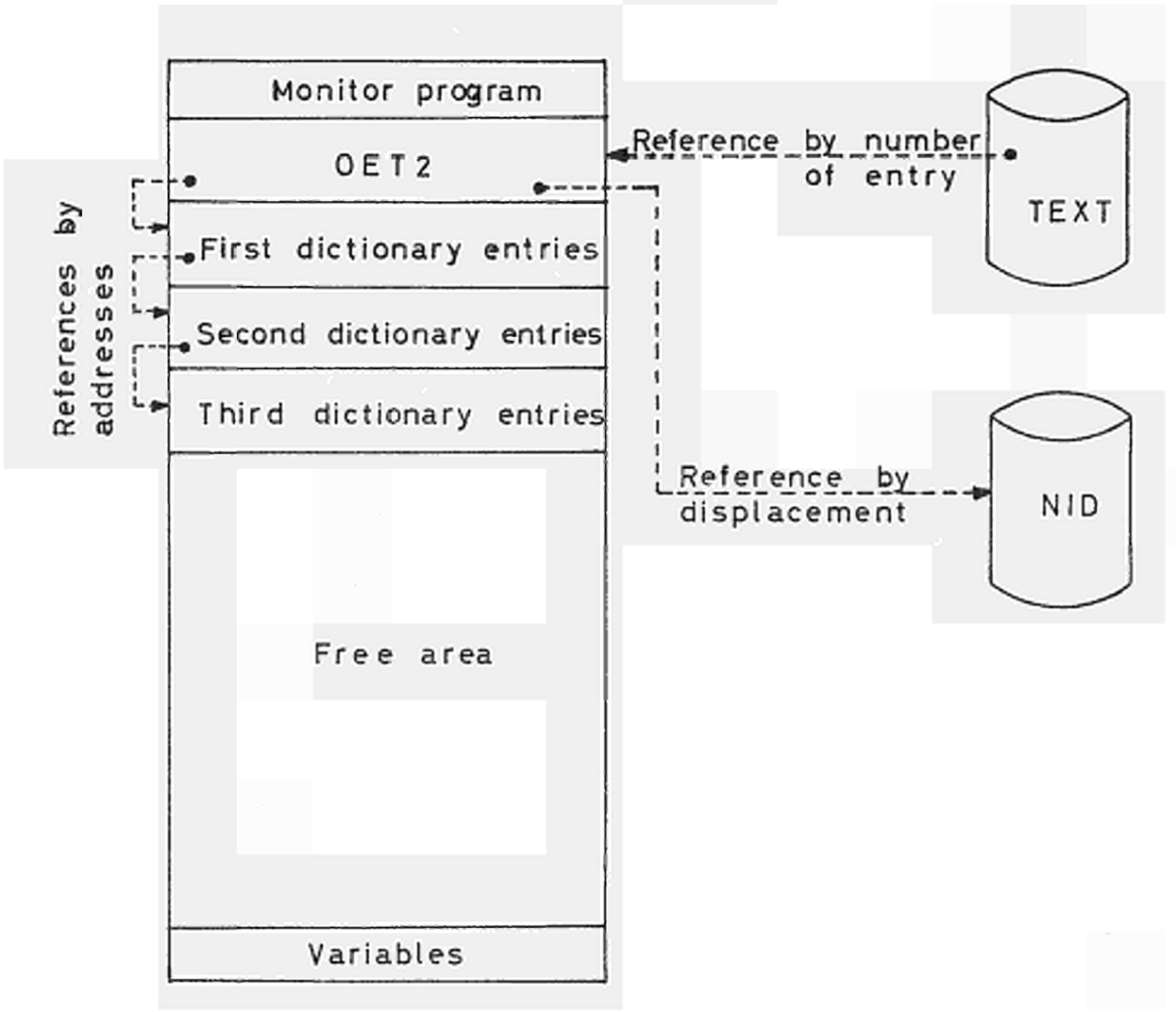
3.3.1 - Data Organization in Input

The precedent phases make available all data necessary for the execution of the problem program, in particular:

- The definition of the source text batch which can be processed in one cycle, depending on the central-storage size. The source text is recorded, with conventional codification, in the TEXT file.
- The definition of the properties of each element of this text batch (as word item or non-word item) according to the grammar assigned to the text analysis program. This classification is also codified in the TEXT file.
- The assignment of all possible meanings to each word through dictionary search and morphological analysis. The dictionary search module is able to handle multiple matches, word inflection, word derivation, segmentation of compound words, prefix separation and a tentative suffix analysis of words not found in the dictionary. All these informations are codified in OET2 table.
- The loading into core storage of the associated dictionary entries (in principle source language dictionary, transfer dictionary and target language dictionary).

The data organization after dictionary look-up phase is shown in Fig. 14, and Fig. 15 shows the contents of TEXT file and OET2 table for a source text sample.

As one can see, all these data are scattered over the main storage and it would be difficult to handle them as they stand. Therefore, the text is subdivided into "logical unit" (sentences, paragraphs, abstracts according to



Legend: TEXT Sequential data-set that contains, for each item of source text, a reference to OET2 table (word item) or the non-word item.

NID Sequential data-set containing , in alphabetic order, the unknown words.

OET2 Table, each entry of which contains:

- reference to match
- reference to segment
- HWO of segment
- address of dictionary entry for word or relative address of unknown word

Fig. 14 - Data and Storage Organization After Dictionary Look-Up Phase

SOURCE TEXT : I HAVE BOUGHT 12 PENHOLDERS .

Result of text analysis :

Word-items : I , HAVE , BOUGHT , PENHOLDER , .

Non-word-items : 12

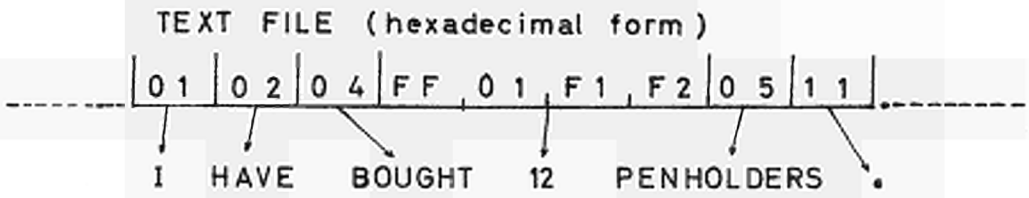
Result of dictionary look-up :

HAVE Two meanings (author,verb)

PENHOLDER Has been divided in PEN and HOLDER

PEN Three meanings (paddock,verb,for write)

HOLDER Two meanings (of a bet,device)



OET2 TABLE

N	RM	RS	HWO	Dict.
1	0	0		
2	3	0		
3	0	0		
4	0	0		
5	7	6		
6	7	0		
7	9	8		
8	9	0		
9	11	10		
10	11	0		
11	13	12		
12	13	0		
13	15	14		
14	15	0		
15	0	16		
16	0	0		
17	0	0		
18				

Dictionary entries

I	
HAVE	1
HAVE	2
BOUGHT	
PEN	1
PEN	2
PEN	3
HOLDER	1
HOLDER	2
.	

Fig 15 _ Contents of TEXT file and OET2 table for sample sentence

the application), and each one of these units is organized in core storage as a four-levels tree structure (item - match - segment - form), called TEXT IMAGE, the nodes of which are linked to all the informations obtained from the precedent phases. Part of the functions of SLC-II programming language was specially designed to handle this tree structure and to access the different data.

3.3.2 - Organization of Logical Text Units (ORGUL Module)

The organization of logical text units and the construction of the respective tree structures are performed by the module named ORGUL.

Firstly, this module loads the data-set of unknown words (NID). Then it scans sequentially the data-set TEXT. Each record of TEXT file contains a reference to the associated entry in OET2 for word items, or the non-word item. The order of the records corresponds to the order of source text items.

For each record of TEXT it allocates an entry at the highest level of the tree structure (ITEM). Subsequently, the lower levels are created according to the structure of the results of dictionary search, recorded in the list OET2 as follows. Each entry of OET2 contains:

- a reference to another match or 0 for the last match,
- a reference to another segment or 0 for the last segment,
- a binary vector, called HWO, that represents the morphological classification of the segment,
- a pointer to the associated dictionary entry.

Depending on the TEXT file contents, there are two possibilities for creation of tree structure:

- The element is a non-word item.

In this case only one entry at MATCH, SEGMENT and FORM levels res-

pectively, are created. The field HWO in the level FORM is set to zero.

- The element is a word item.
  In this case, the number of the entries at the levels MATCH and SEGMENT depends on the number of homographs, and on the segments within each homograph (inclusive derivations), detected during dictionary search.
  Note that the number of entries at the level FORM is always equal to one due to the strategy of dictionary search, and therefore, the eventual morphologic homographs must be represented within only one headword O.

The logical flow of ORGUL program is shown in Fig.16 , the Fig. 17 shows the general structure of TEXT IMAGE tree, and Fig.18 shows this structure through a sample sentence.

The construction of TEXT IMAGE is interrupted when a delimiter element is encountered in the text. The delimiter elements are conventional word items (e.g. period, semicolon, end-of-paragraph symbol, etc.) which have been communicated to the system through a parameter list.

When the construction of TEXT IMAGE is completed, control is given to the SLC executor module which initializes the SLC environment and controls the execution of the problem programs written in the SLC programming language.

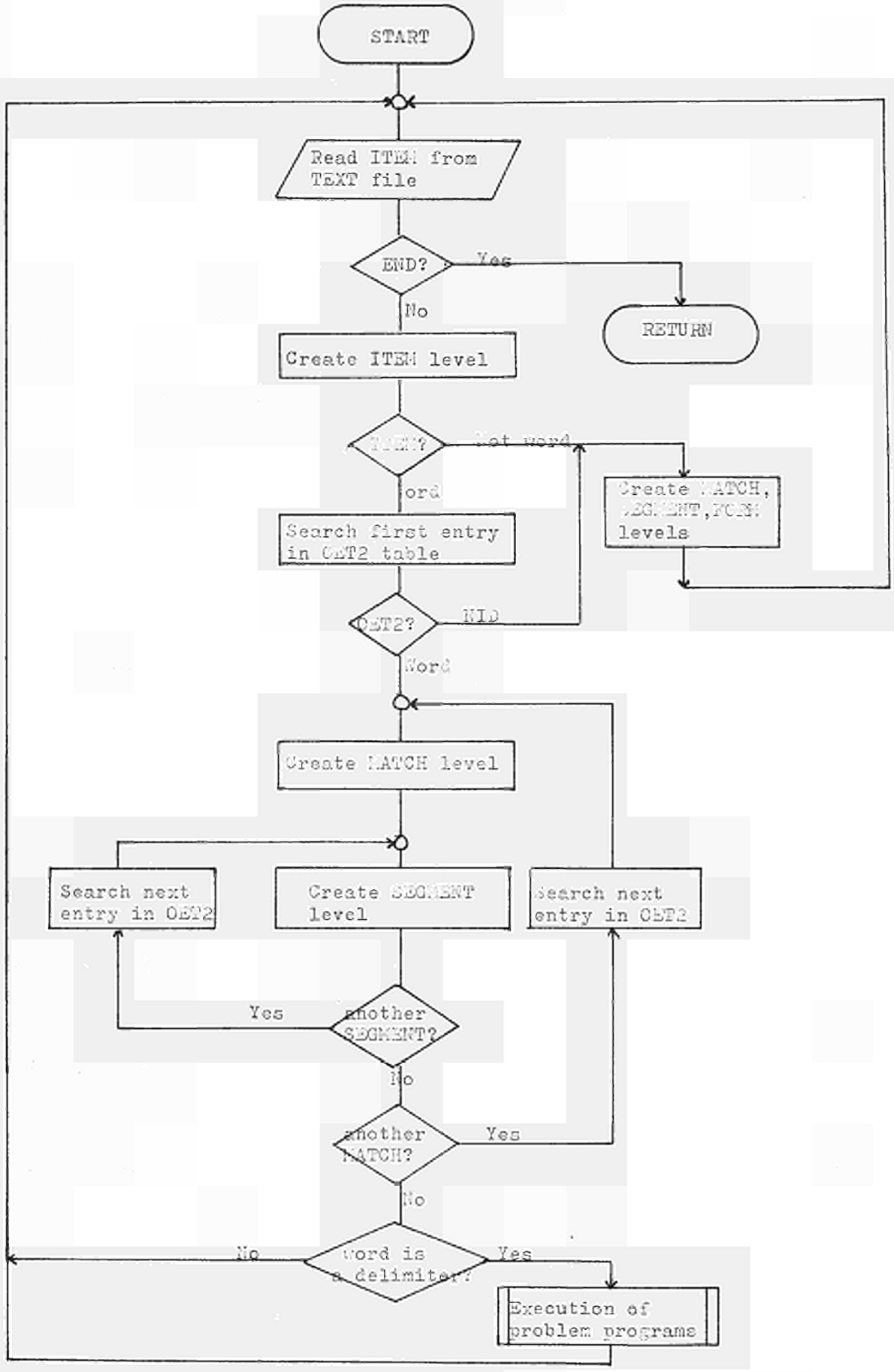
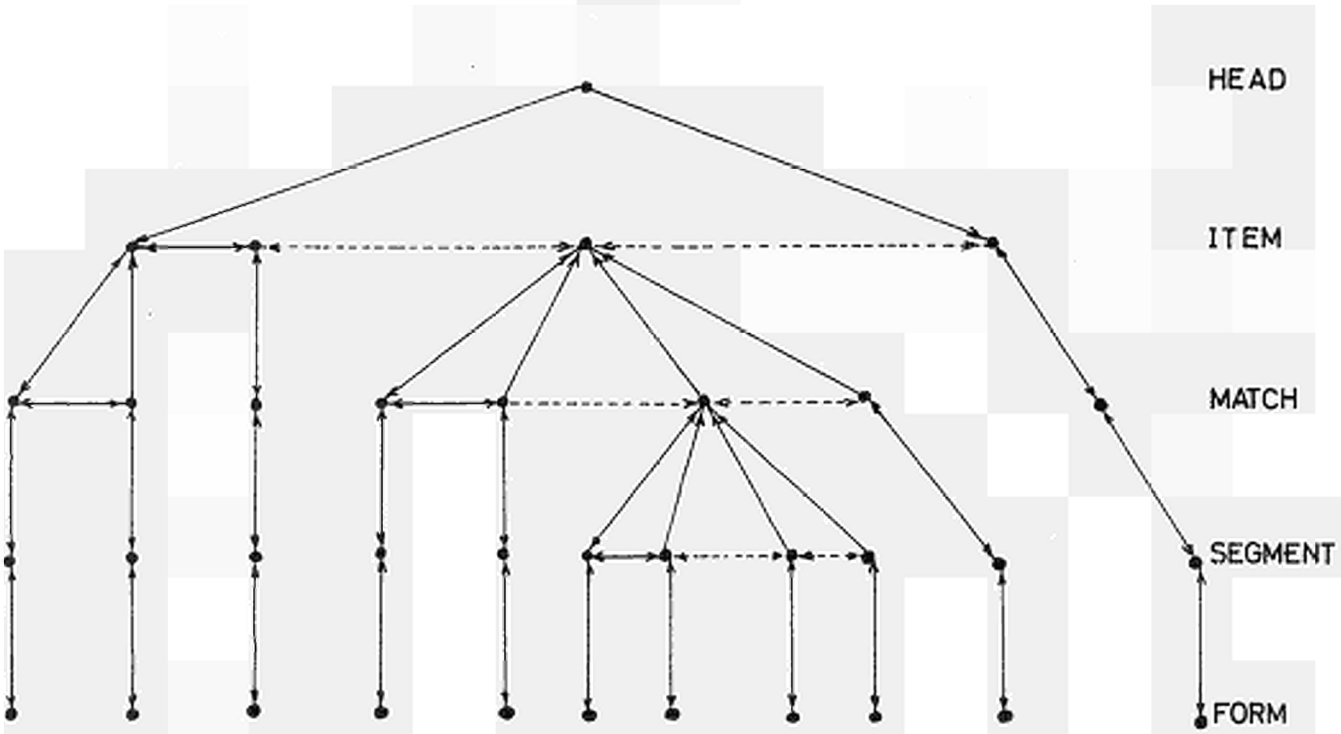


Fig. 16 - Logical flow of text image construction.



The arrows show direction of chains

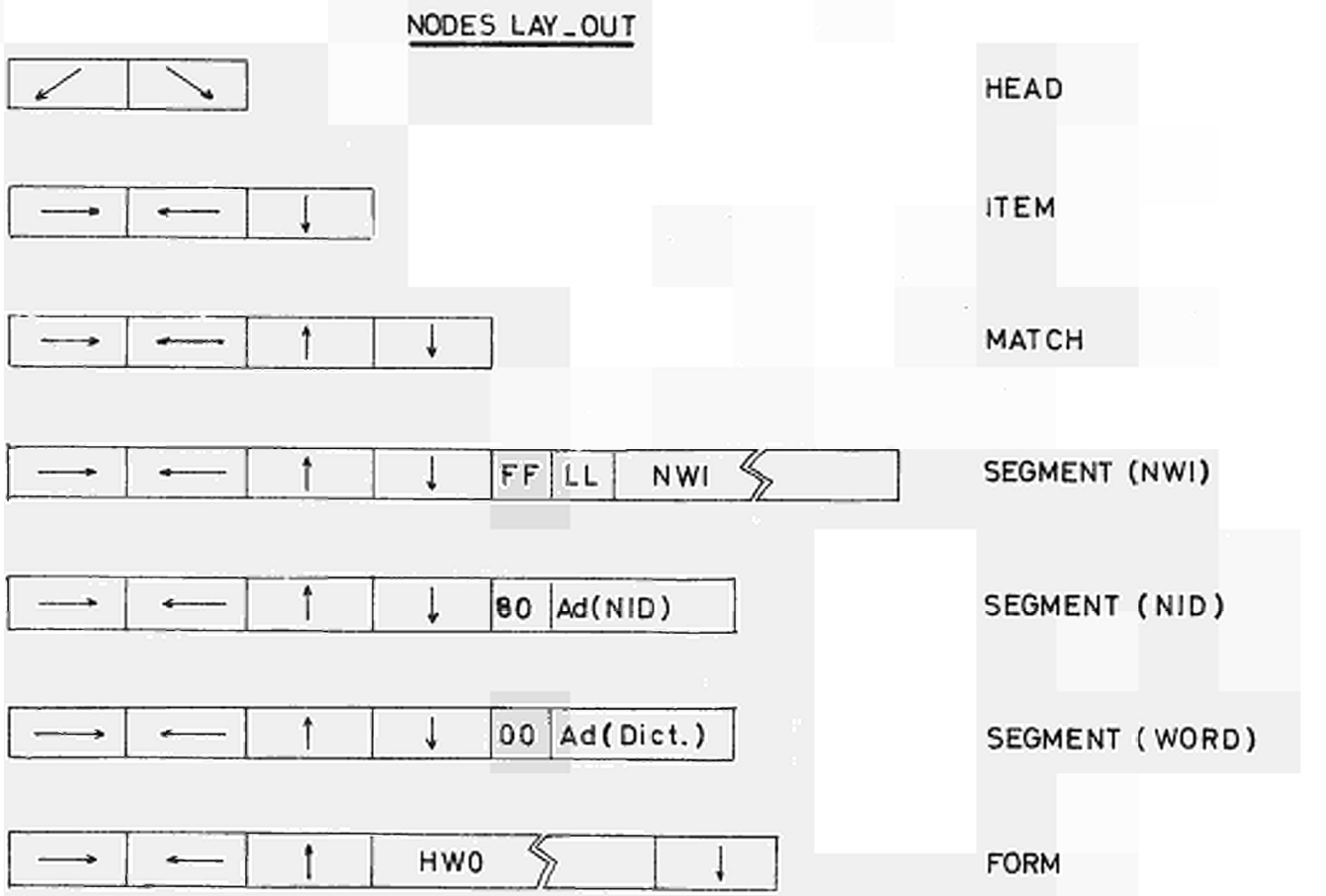


Fig. 17 - Text Image Structure and Lay-Out of Nodes

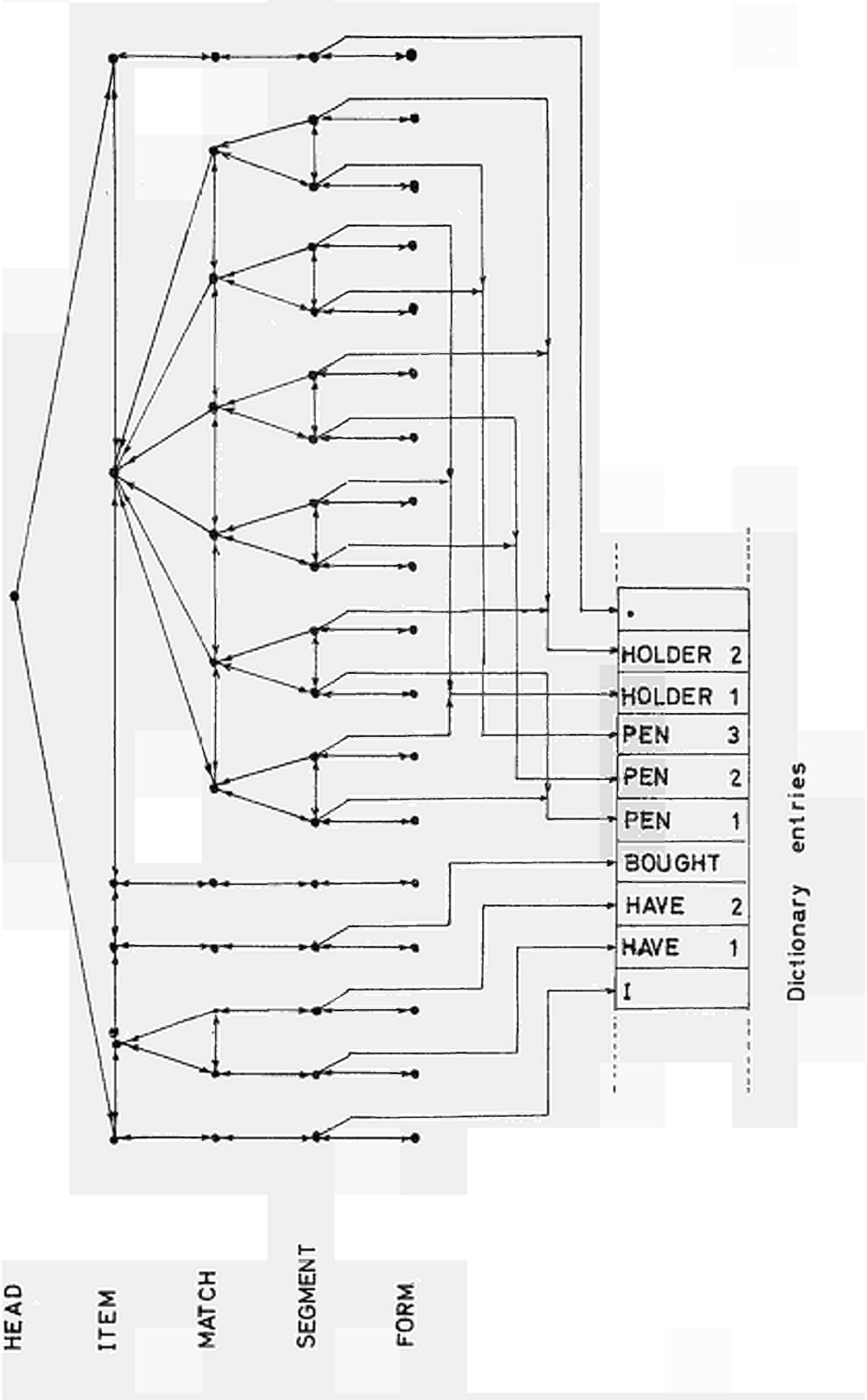


Fig. 18 - Text Image Configuration for Sample Semtence of Fig. 15

3.3.3 - The SLC Executor Program (Fig. 19)

This program controls the execution of the programs written in SLC programming language. In principle, it simulates the CPU (Central Processing Unit) and the OS (Operating System) of a computer.

The execution of SLC-written programs is articulated into three cycles which represent the three main functions in translation (applications as indexing query formulation etc. also can be interpreted as translation problems):

- source text analysis which produces a formalized description of the source text according to the linguistic model adopted;
- transfer - which transforms the results of the first cycle into a formalized representation of the target text (this function can be considered auxiliary for all those parts of the process which cannot be completely formalized and therefore rely on the equivalence of symbols in two languages);
- target text synthesis which generates the text in the target language.

Each one of the cycles is constituted of three components:

- an algorithm realized as the problem program written in SLC-II language;
- a dictionary which contains all information about the words necessary for processing;
- a grammar which is a set of rules according to which the algorithm processes the text unit.

In principle, in partial applications, the number of cycles can be reduced to two or one.

Within each cycle, a series of independent SLC-II written programs is

executed according to their priority which is assigned to the program at request time and can be modified during execution. A priority scheduler controls the program execution requests and gives control to the highest-priority program.

All programs executed can be linked to each other through a call mechanism which permits a recursive and re-entrant use of the single modules. The compiler facilitates the design of recursive and re-entrant modules.

The single SLC-II instructions are executed in interpretive mode: i.e. the program is a binary string in core storage, and a routine simulates the functions of the CPU which is subdivided into

- a FETCH which loads the next instruction according to the instruction counter, interprets the binary string as an operation code and operands and locates the operands,
- an EXECUTE which mainly consists in the branch to the function subroutine,
- a TRACE which is optional and is to facilitate program debugging.

3.3.4 - Organization of SLC-II Programs

A SLC-II problem program, normally, is subdivided into three cycles. TEXT IMAGE for each one of the cycles ensures the link to the relative dictionary information. Hence, in principle, there exist three dictionaries, which are linked to TEXT IMAGE at the SEGMENT level. However, all dictionary entries are linked to each other from the first cycle on (in batch mode) and therefore, one could also design an SLC-II program with one sole cycle and several dictionaries.

The link between the cycles is ensured by special modules which rearrange TEXT IMAGE and purge all elements which are not used in the subsequent cycle.

At the end of the third cycle, the results of the SLC-II program are represented as a string of items consisting of the lexical code of a word and a binary string (HWO) which contains the definition of the inflection form and lay-out information. This string is processed by the programs for morphological generation and text editing at the end of an input batch.

Each SLC-II program is recorded as a load module in a library and can call other programs through the instruction CALL or schedule the execution through the instruction REQUEST with a certain priority. Control is given to the latter by the priority scheduler when a return to the executor is performed.

3.3.5 - SLC-II Language Environment

One part of the SLC-II programming language is a subset of the standard IBM 360 instruction set (the decimal and floating point arithmetic are excluded). SLC-II, hence, simulates the central organs of the 360 computer with the general-purpose registers and the byte storage.

A second subset permits operations on TEXT IMAGE and the associated dictionary entries (addressing of particular entries, transformations, etc.).

A third subset permits to construct and handle syntactic structures. The SYNTAX storage is linked to TEXT IMAGE at the lowest level (FORM), and permits to describe the source text in the form of relations according to various linguistic models.
The basic elements of the syntax store are relations consisting each of one operator and two operands.
All SLC-II language instructions take into account this environment, either explicitely or implicitely.

3.3.6 - Brief Description of the SLC-II Programming Language

The SLC-II language can be subdivided into two parts:

The first part is exactly a subset of the Assembler 360 programming language, and permits the control of the general registers and of the byte-structured storage.

The second part is specific to the SLC-II system and permits the control of the remaining elements of the SLC-II environment:

- addressing of elements of the text-image storage,
- creation, changes and deletion of elements of this storage,
- control of the program queue,
- input/output operations,
- access to dictionary data,
- access to data modules,
- etc.

In its present form, the SLC-II programming language is an elementary assembler-type language (i. e. each statement produces one binary string which consists of an operator and operands). There exists no dedicated SLC-II compiler. Instead, one uses the IBM 360 Assembler H compiler with an associated special macro-library which contains the operation codes and their expansion. This solution, of course, is not computer-independent. There exists, therefore, the intention of re-formulating the compiler and executor in a transportable high-level programming language (like Algol or PL/1).

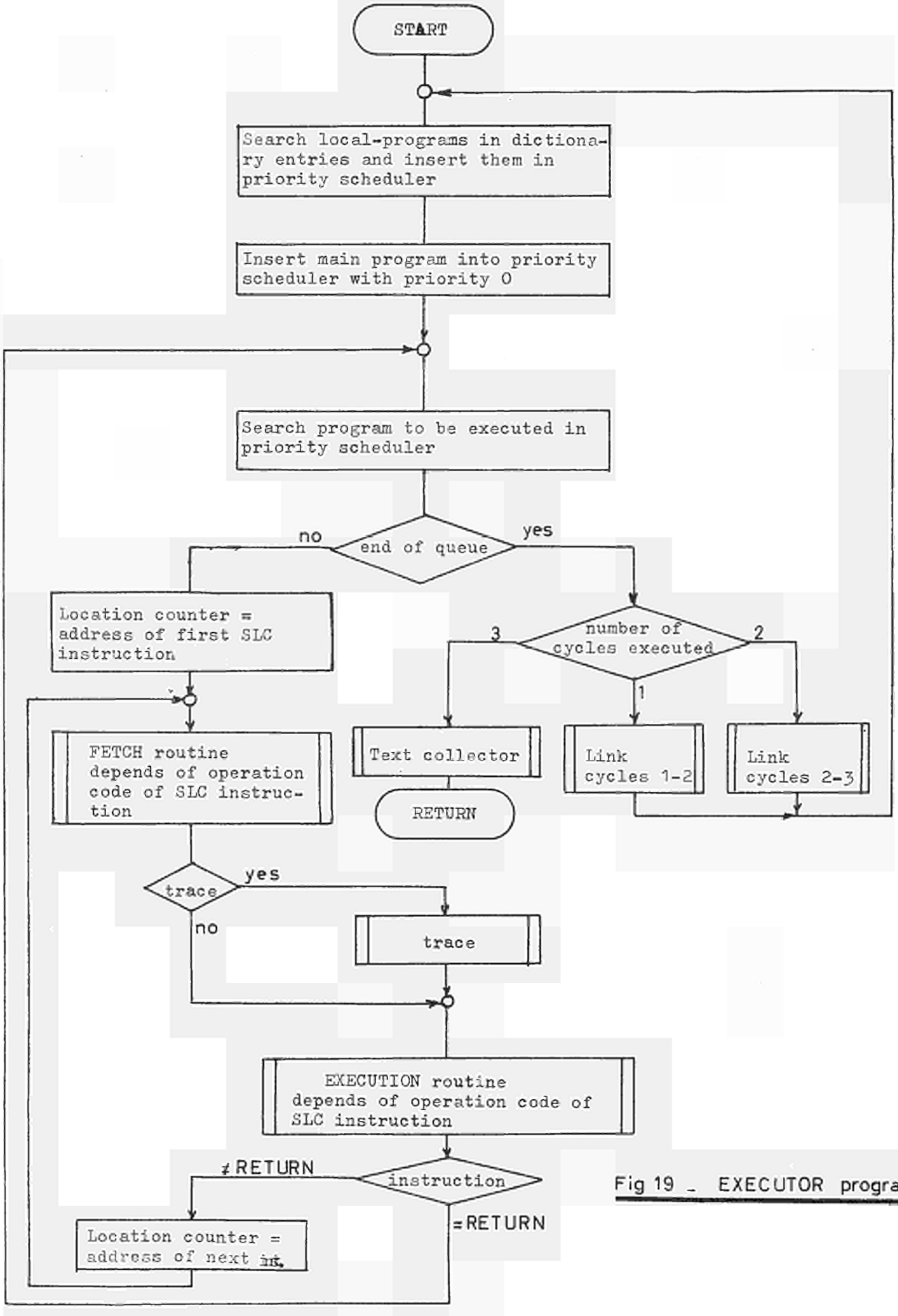


Fig 19 _ EXECUTOR program

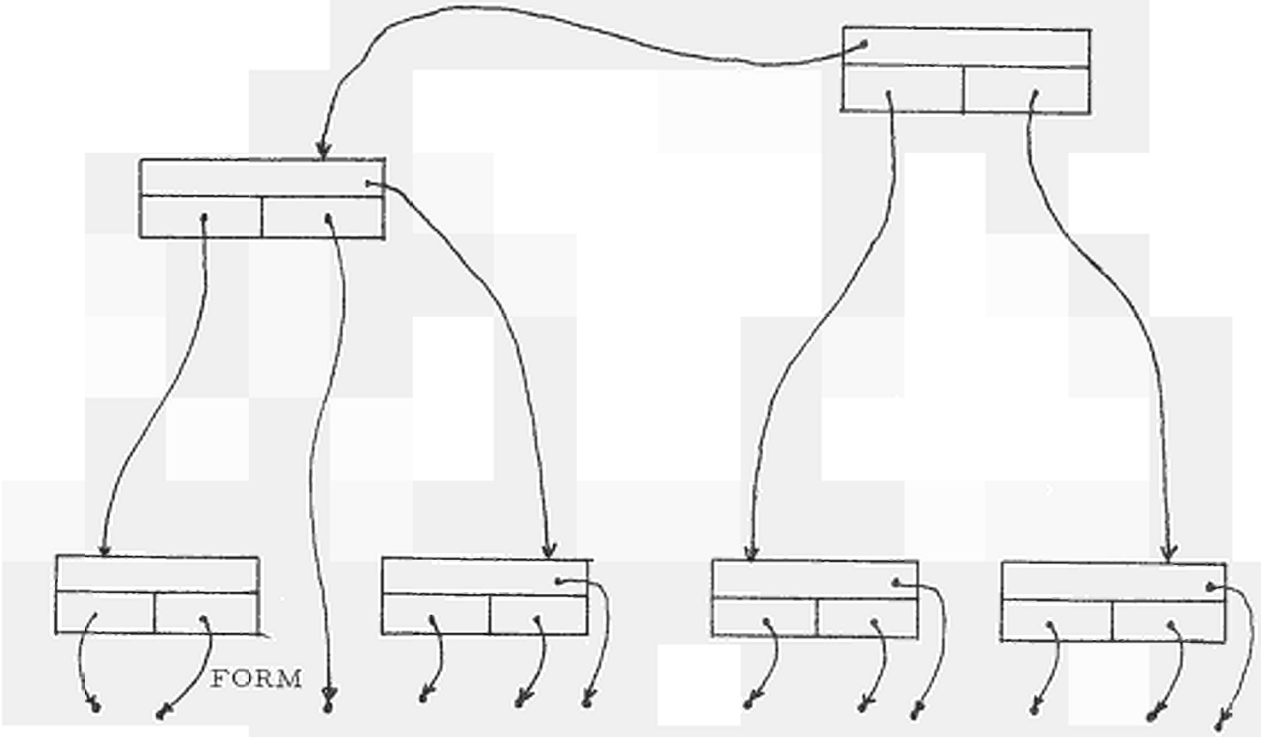
3.3.7 - Syntax

3.3.7.1 - Organization

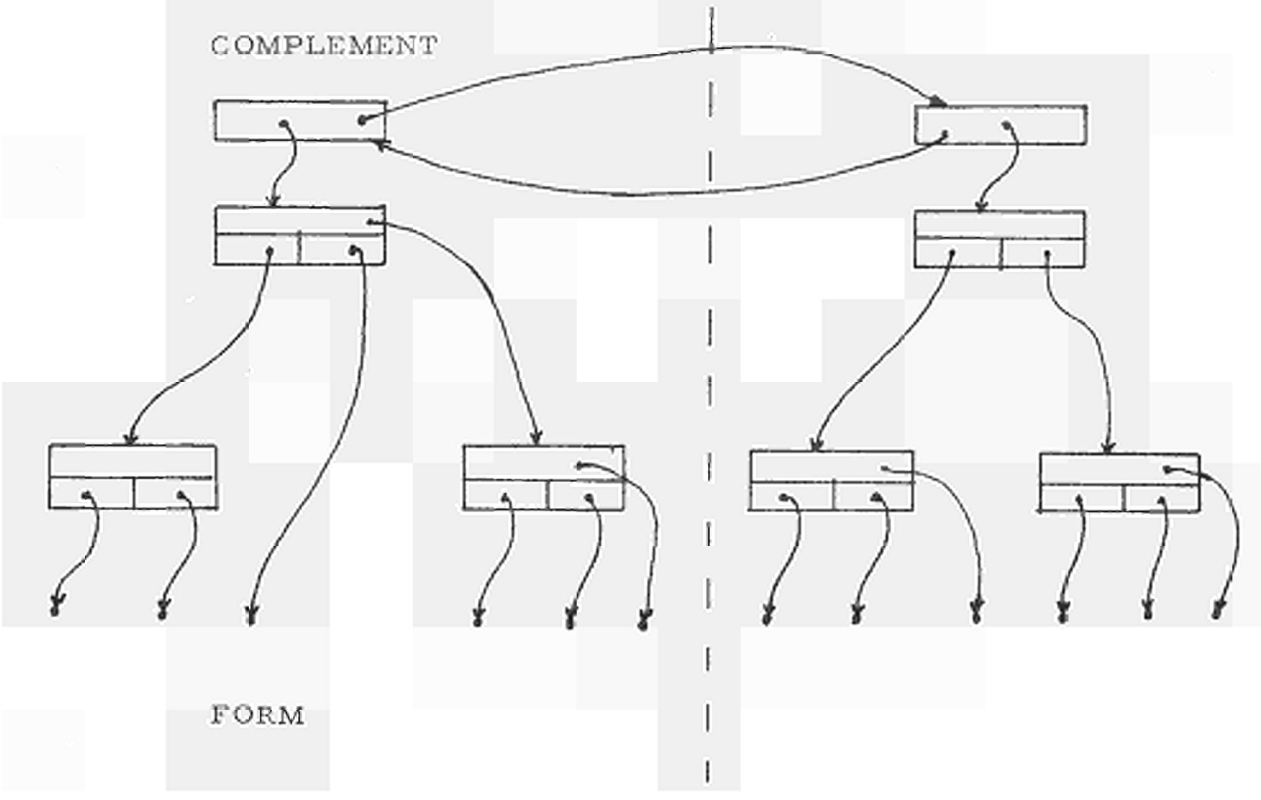
As it was stated above, SLC-II was designed primarily for syntax- and semantics-oriented language processing. Therefore, particular care was dedicated to the definition of a syntactic model and system functions, which permit on one hand to use more or less easily any existent linguistic model, and to construct, explore and transform syntactic structures on the other hand. A subset of the SLC programming language is specifically concerned with syntax which is to be interpreted as a particular organ of the hypothetic computer being simulated.

The basic syntactic model, as it was stated above, is relational, and a syntactic unit consists of one operator and two operands (very similar to arithmetics or logic), which, depending on the level of analysis, may be interpreted as surface structure or as some kind of semantic deep structure.

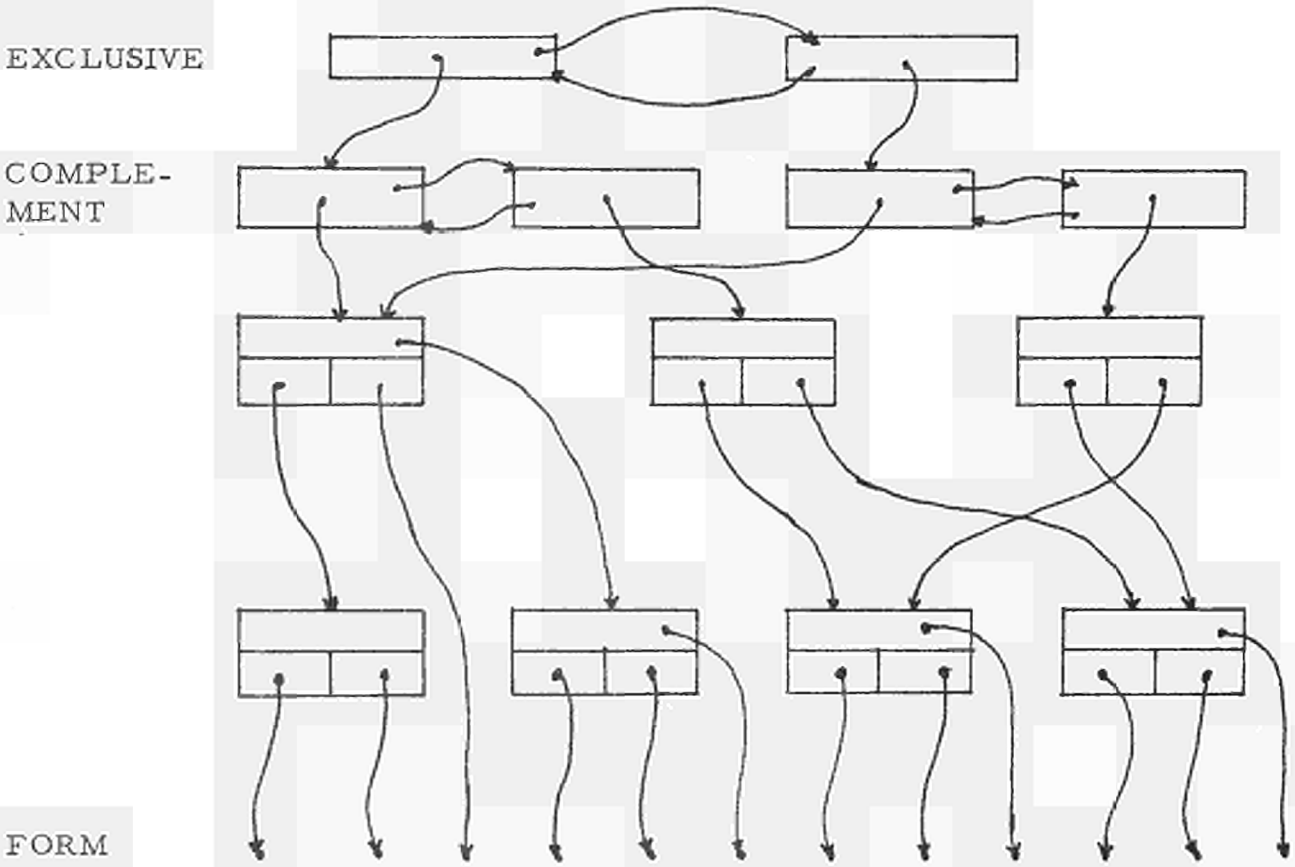
Each relation represents a node of a tree structure which, at the lowest level, is connected to the "form" level of the text image and on the highest should comprehend, at the end of analysis, the entire logical text unit. Actually, the syntax orientation of the system is so strong that all elements of the text image not related to the syntactic representation, are deleted from storage at the end of the first cycle of the problem program.



During analysis, the syntactic representation of the text is frequently partial and comprises but parts of the logical text unit for each sub-structure. To facilitate,in particular,top-to-bottom analysis, all substructures, which could be put in relation with each other, i.e. are "complementary" are linked to each other by a further node of the tree structure, in the order of their linear occurrence in the text



All alternative solutions, which, in principle, cannot be put into relation with each other, are grouped together at the highest level of the syntactic text representation ("exclusive level").

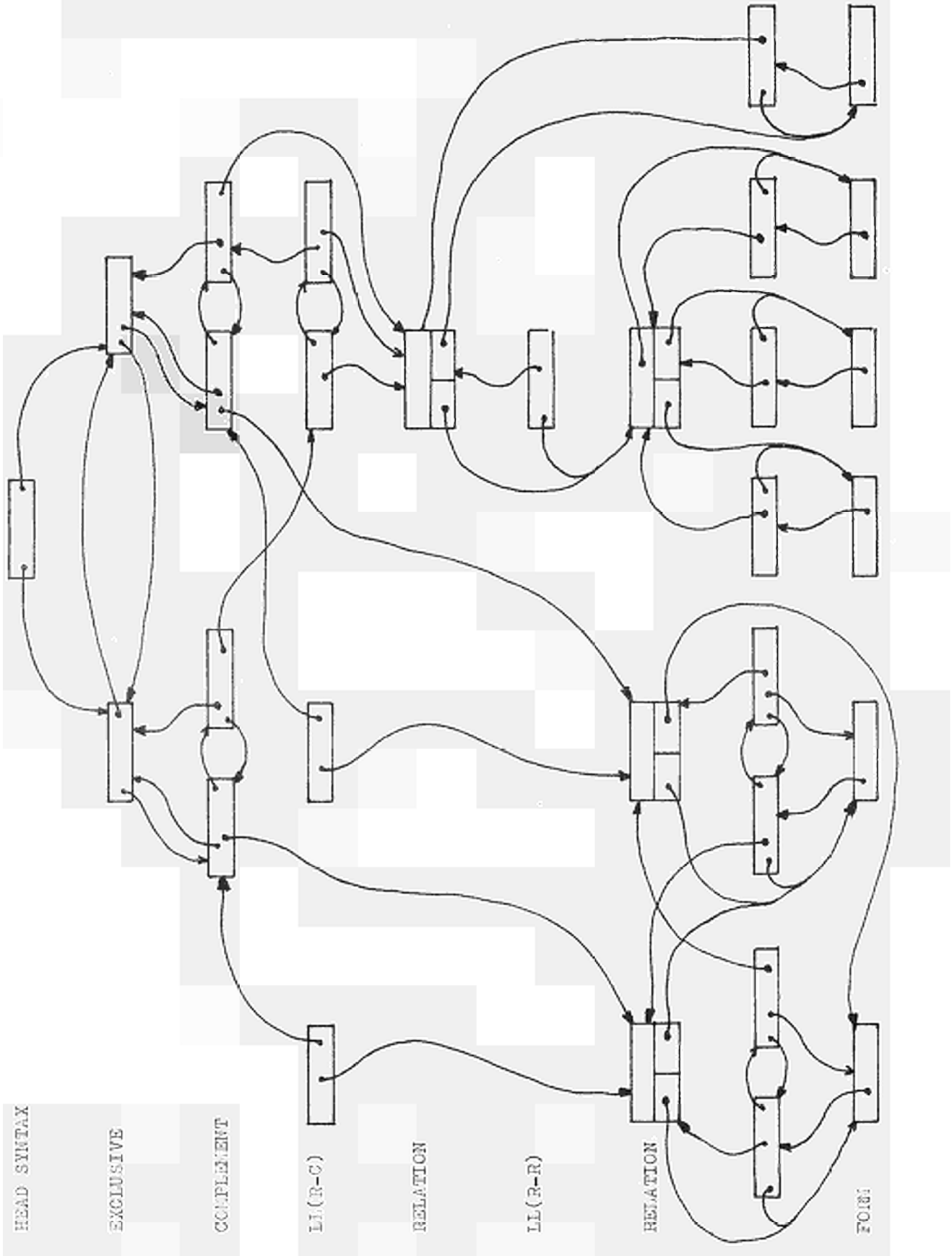


One more facility was provided for the handling of syntactic homographs (total or only partial during analysis). Elements (at the form level) and relations are linked to the upper level not directly, but by means of a nodal point ("link-list") which administrates the alternative use of each of them.

In this way, one can avoid the proliferation of identical sub-structures due to their different use at the higher level. The next figure shows an example of a syntactic structure along with the internal connections between the elements.

In order to preserve all possibilities as "text image", the syntactic representation permits to record information like word-order, delimiters,

the way of connections (direct, resumed, artificially supplied), etc.



3.3.7.2 - Construction and Exploration of Syntactic Networks

In principle, the construction of a relation is performed in two steps: The first step consists in creating a provisional ("transient") relation which contains the links to the pre-existent structures, but does not affect them. The principal SLC instructions are CREATE, INSERT, CLEAR, DELETE with the operand TRANSIENT. Note that only one transient relation can exist at a time.

If, after the establishing of a transient relation all conditions of the grammar are satisfied, the relation can be set "permanent", i.e. definitively inserted into the existent structures. The "SET PERMANENT" automatically explores the exclusive and complementary attributes of the new relation established and creates the relative connections.

For the exploration and management of the syntactic structures, the SLC programming language provides a set of instructions enabling to:

- move up and down the structure,
- to interrogate and modify the single fields of a relation,
- to delete single relations or entire sub-structures (up to the complementary and exclusive level),
- to perform top-to-bottom explorations keeping track of the path.

Actual transformation of a structure must be performed in two steps:

- first one deletes the single relations to be modified,
- second one creates new transient or permanent relations and inserts the elements in the desired way.

Algorithms for syntactic recognition and generation, and the relative grammars are to be prepared in the context of the machine translation project at CETIS. Particular care is being taken in this context, to make

both the algorithms and the grammatical model, in principle, language- and application-independent so as to ensure the transportability to other languages and applications.

3.3.8 - Communication Between the SLC-II Programs and the System

All communications between the SLC-II programs and the system are provided for by the EXECUTOR module.

The SLC-II programs must correspond to one of the following rules:

- There must be, for each cycle, a main program, edited as a load module in the associated job or step library. The names of these main programs are communicated to the EXECUTOR as parameters.
- The main programs may call sub-programs, which, themselves, must be organized in the same way as the main programs. Their names are communicated to the EXECUTOR as operands of the SLC-II instruction CALL (see Fig. 20).
- A program can insert another program into the request queue, or remove it, or it can change its priority. The name of this program is operand of the relative instructions (see Fig. 21).
- Finally, a SLC-II program can be part of an entry of one of the dictionaries. The EXECUTOR scans the dictionary entries and inserts the "local programs" with their priority into the request queue. Programs located in the dictionary entries are conventionally called local programs, while programs edited in the job or step library are called "general programs". Local programs can refer to general programs, but not vice-versa. The only way of accessing local programs is the priority scheduler.

In its present version, the system loads the modules into core storage each time they are called and deletes them after execution. This may in-

crease somewhat the number of I/O operations, but permits to reduce the main storage requirements for the programs and, consequently, to allocate more space for data. The maximum dimension of the SLC modules must be communicated to the system as a parameter.

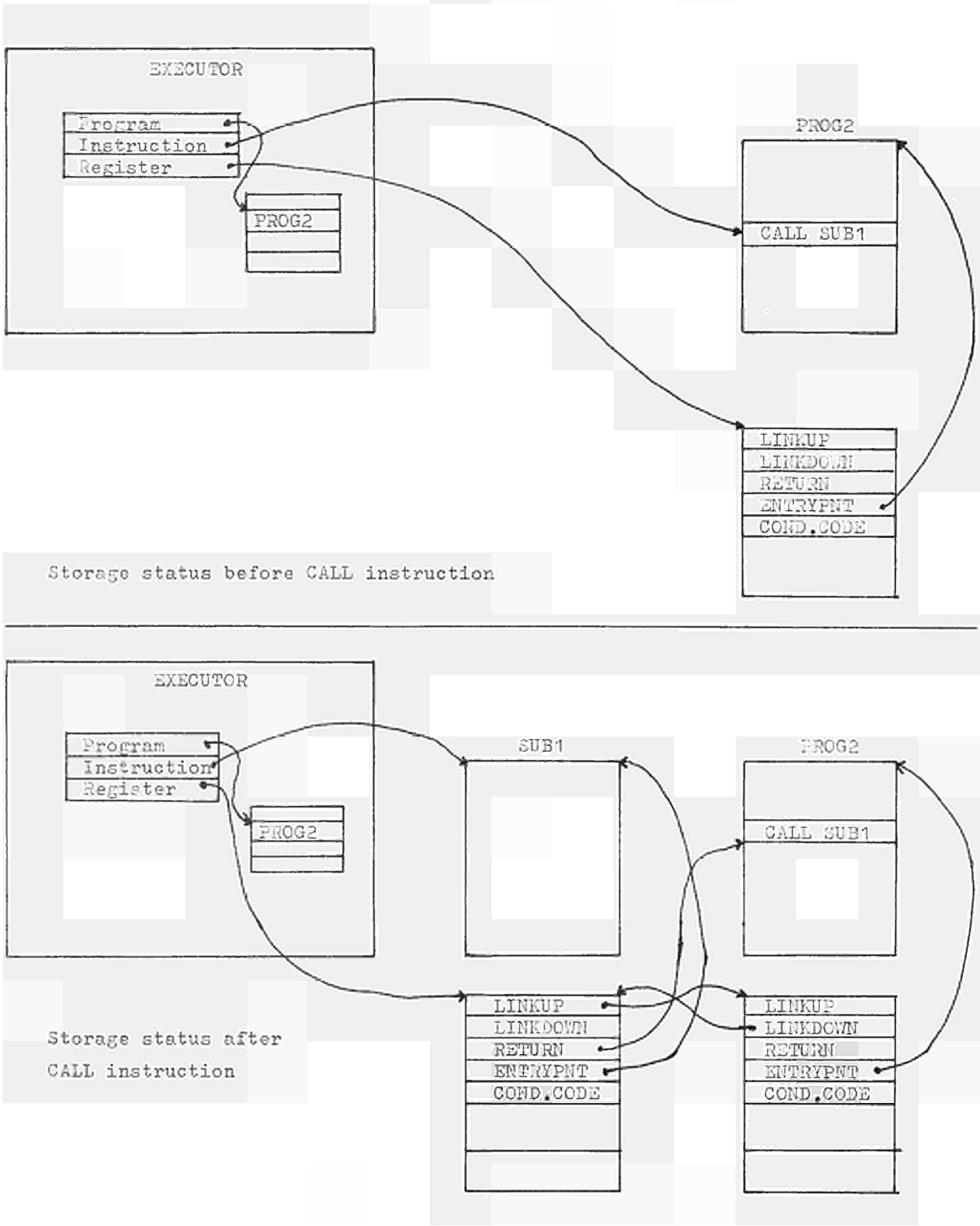
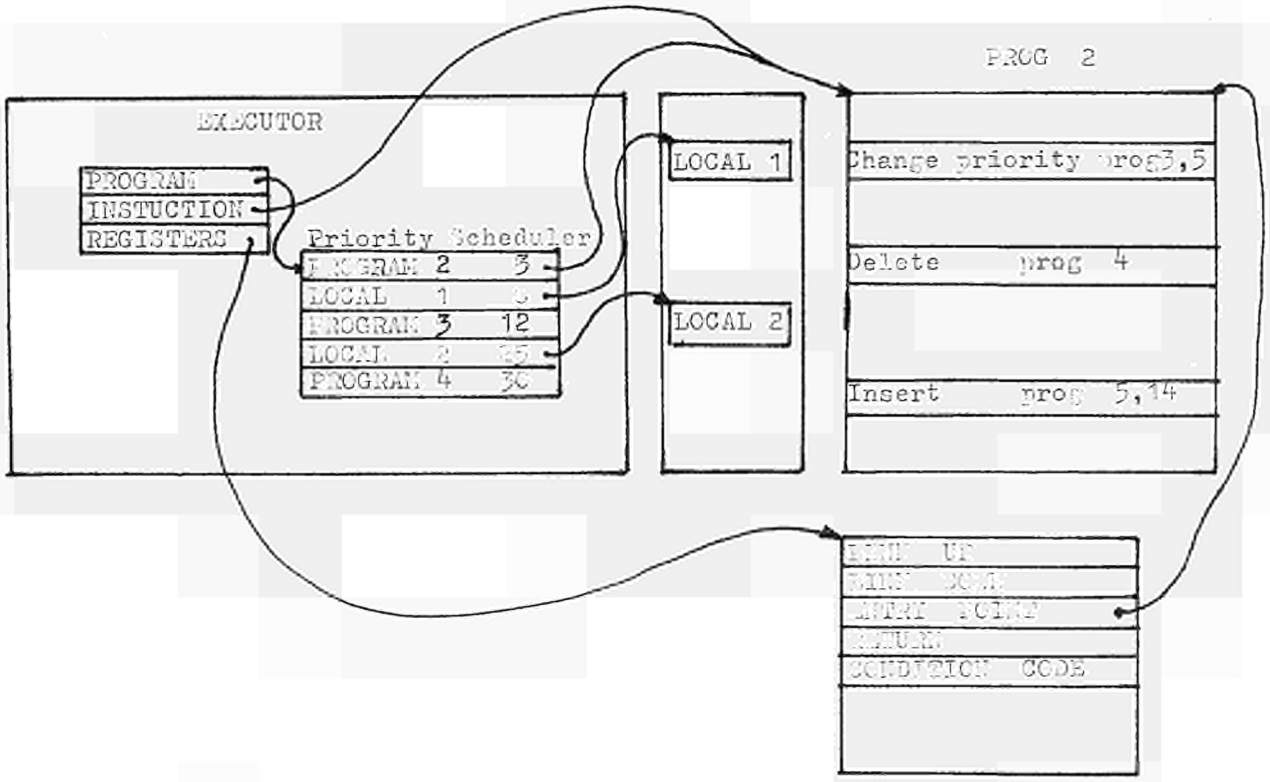
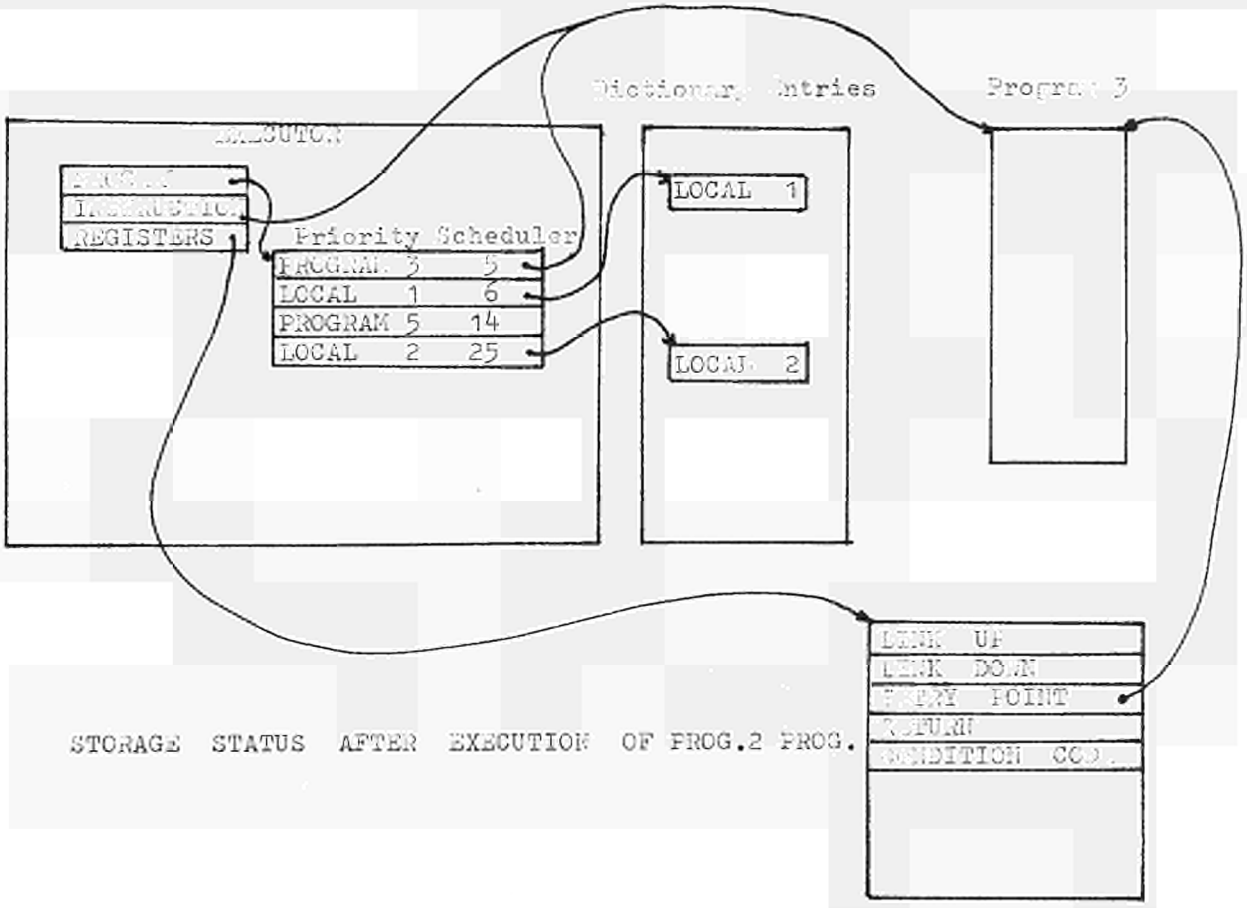


Fig. 20 - Call Mechanism



STORAGE STATUS BEFORE EXECUTION OF PROG. 2 PROGRAM



STORAGE STATUS AFTER EXECUTION OF PROG.2 PROG.

Fig. 21 - Request Queue Mechanism

3.3.9 - Data Modules

As SLC-II programs may need data beyond those contained in the dictionaries, e.g. grammars, one can code and compile so-called data modules with any structure desired. These, non-executable modules are managed by SLC-II instructions. Data modules must be link-edited as load modules in the SLC-II library. The names of the data modules are communicated to the executor as operands of the instructions involved.

Basically, these modules are used for coding grammars. Should one want to use them as auxiliary programs, the programmer must provide the calling sequence and the return. This facility may be useful in case one designs a grammar as a finite-state automaton. In this case the data module is bi-partite and contains non-executable parts which point to executable action routines.

3.3.10 - Intra-Cycle Communication Storage

In general, SLC-II programs process logical text units independently one from another. If data must be communicated from one logical text unit to another (e.g. in automatic abstracting, or in machine translation for the resolution of problems like pronouns or articles), one has the option of allocating a certain amount of intra-cycle communication storage, which in principle has byte structure and can be accessed only by SLC-II instructions. The storage initially is cleared and remains un-affected by any system operation during the whole job.

3.3.11 - Input/Output Facilities of SLC-II Programs

The standard input to SLC-II programs is constituted basically by the source text, the associated dictionary entries and, eventually, by the data modules and the intra-cycle communication storage.

The standard output is provided for by the TEXT COLLECTOR module which is given control at the end of the third cycle and collects the final result of the "translation" process. It consists basically of:

- lay-out and editing codes,
- the lexical identification code and the binary vector defining the morphological form of word items,
- the character sequence representing the non-word items and the words not found in the dictionary.

These data are collected as a data set and used as input to the morphological generation and editing modules.

As optional features two additional I/O facilities have been provided for:

- handling of "permanent data sets" which on one hand can be used to communicate data from cycle to cycle which cannot be kept in intra-cycle communication storage because of their volume, on the other hand to produce special outputs (as for instance the direct file entries in automatic indexing),
- message handling: in input, in the batch version, input of messages is not provided for and will be introduced in the conversational version. In output one can print messages, lay-out of text image, syntactic structures, dictionary entries etc., dumps, and, as a debugging aid, trace the SLC-II instructions executed. In the batch version, messages appear on the printer, but will be sent to terminals in the conversational version.

3.4 Editing

3.4.1 - Morphological Generation

3.4.1.1 - Input

After the execution of the SLC-II-coded problem program, the TEXT COLLECTOR creates a sequential data set which contains for each item to be edited the identification codes (LXN), the morphological information (HWO) and the lay-out control codes. This data set is the input to morphological generation program, needed for operating the target language morphological dictionary and the paradigm tables as grammar.
The target language morphological dictionary is accessed by the lexical identification code (LXN) and contains the invariant part of the grapheme (stem) and a pointer to the paradigm.

The morphological generation program permits to attach suffixes and prefixes to the stem (which may also be nil) and to chain them (suffixes from left to right, prefixes from right to left).

3.4.1.2 - Functions

For each different lexical number of the input data set, one accesses the corresponding dictionary entry, attaches the suffixes and/or prefixes to the "stem" and transmits the grapheme,thus completed,to the editing program.

The morphological generation program is articulated into three parts:

- Analysis of the input data set. The program scans the entries containing a lexical number and inserts the lexical numbers into a table

with a hash code technique, similar to that of the DICLOADER module. At the end the entries are sorted by lexical number;

- Loading of the dictionary entries. One uses exactly the same technique as described in paragraph 3.2.4;
- Morphological generation. The program re-reads the data set modified in phase a, and analyses all entries. Those containing non-word items, unknown words and null items (items which contain only editing information) are transmitted to the editing program without any processing.

Entries containing word items are connected through the hash table with the dictionary entries. One moves the stem into a work area,locates the corresponding paradigm and looks them up for the correct prefixes and/or suffixes. The location of the affixes is performed by logical operations on the binary vector HWO. The program permits to chain several affixes.

3.4.1.3 - Output

The output of the morphological generation program is constituted by:

- The character string to edit,
- A binary vector containing the control codes for editing.

This information is transmitted directly to the editing program, as the last phase of morphological generation and the editing program co-exist in core storage.

3.4.2 - Final Text Editing

The program analyses the binary vector with the control codes transmitted by the precedent phase and processes correspondingly the character string associated, or sets permanent switches.
The editing control codes have two functions:

- permanent control, i.e. valid until another code of the same type is encountered. It concerns primarily the general lay-out data as tabulation, the length of the printed line and of the margins, the character type and the inter-line;
- control concerning only the item the code is associated with. It defines the selection of upper- or lower case, the left or right justification of the items, spaces between items, insertion of blank lines or pages etc.

The set-up of the control codes is performed by the last cycle of the SLC-II coded problem program, and inserted in the last two bytes of HWO. Therefore, the description of the target language word form must be exactly 2 bytes shorter than the one of the source language. This fact must be kept in mind, when one defines the paradigm table for the morphological generative dictionary, and sets-up the HWO in the last cycle of the problem program.

4. SYSTEM USAGE

4.1 Application System Generation

In order to use the system, it is necessary to provide for a certain number of data:

- The source text: it must be recorded on a machine-readable support (cards, paper tape, magnetic tape, etc.), according to the conventions defined by the input grammar associated.
- Options: the use of options gives the system an extreme flexibility. One can subdivide them into three classes:
  - options which permit to select a particular application, as, for instance machine translation, automatic indexing, thesaurus construction, query formulation for IR or SDI, etc.;
  - options which permit to select certain particular services of the system, as, for instance error messages, word lists, listing of the source

text, dictionary search options, etc.;
- options which permit to change certain standard values as, for instance data set attributes, program and grammar names, average length of dictionary entries etc.

- The description of the coding conventions of the source text which comprehends a dictionary of the elements with a special function in the text (delimiters, control codes etc.), and a grammar, which permits to fragment the text into substrings and to qualify them with the attribute of word items or non-word items. Dictionary, grammar and the associated semantic action routines are one load module of the system, invoked by the text analysis program.

- The source language morphological search dictionary, associated with the relative paradigm tables, which permit to describe each word item by means of a lexeme identification code and the definition of the inflectional form. The dictionary is a direct access data set.

- The paradigm tables are a system load module invoked by the dictionary search program.

- The generative morphological target language dictionary and the relative paradigm tables. They are organized in the same way as the corresponding source language dictionary, and are requested in all applications which provide some output in natural language.

- Up to three dictionaries and relative grammars, associated with the SLC problem programs, whose number and structure, essentially, depend on the application. For instance, in translation, one needs a source language dictionary, a transfer dictionary and a target language dictionary.

4.2 Data Base Creation and Management

The SLC system disposes of a set of programs, to be executed off line, which permit to create and maintain the different data bases necessary for some concrete application. The utility programs are executable load mo-

dules of the SLC system library.

A subset of the SLC programming language is dedicated to the symbolic coding of dictionaries and grammars.

4.3 Environment

At present, the SLC system is operational in batch mode with IBM 360/370 series OS. Minimum storage requirement is app. 150 K bytes (excluded OS), while a region of app. 300 K bytes is estimated to give optimal performance. The SLC-II programming language, necessary for symbolic coding of algorithms, dictionaries and grammars, requires the Assembler H compiler as system support, which needs a minimum of 200 K bytes.

All data sets, except the source text and the options, are recorded on direct access devices (discs) as well as the program library and the macro library which has the function of the SLC-II compiler. The peripheral storage requirements can be estimated as follows:

macro library	600,000
program library	1,000,000
source language morph. dict.	app. 25 bytes/entry
target language morph. dict.	app. 20 bytes/entry
other dictionaries	variable

4.4 Sample of a SLC-II System Application

The data in input and output of SLC-II system are shown for a current use. The use chosen for the example is automatic indexing of nuclear abstracts with EURATOM thesaurus.

Only one abstract has been processed in this example, but it is obvious

that the system permits to process an unlimited number of documents during one job step at the rate of some 500,000 words/hour.

The control listing includes three parts:

- The input data (options and text),
- The intermediate results (list of words, unknown words, utility messages),
- The final result (keywords).

OPTIONS

```
THE FOLLOWING OPTIONS WERE CHOSEN
LLHWTAB         01   0100030100006
LLHWTAB         02   1000010200006
LLHWTAB         03   1000010300006
OPTIONS         047F0200
NUMDICT         03
DDNAMDIC        01   DDIC1      450
DDNAMDIC        02   DDIC2      040
DDNAMDIC        03   DDIC3      020
NUMCYCLE        02
SLCMAIN1        INFMAIN1
SLCMAIN2        INFMAIN2
PARTABIN        PARTABFM
PARTABOU        PARTABWG
NMDELIM         01
DELIMTAB        01   00012000
BLKSIZE         7200
LLDELIMS        06
DELIMSTR        ==RIF==
```

This part of output listing shows the options transmitted to the system for the execution of this job.

Each statement is structured as KEYWORD-VALUE, the functions of which are explained below.

LLHWTAB — Defines for each non-standard headword its length, the dictionary number in which it is contained and the level of text image at which the dictionary entry is connected.

OPTIONS	Specifies up to 32 options that can be defined by one binary position. In this job the specified options were: display of input text, display of word-items and display of word-items not found in the morphological dictionary.
NUMDICT	Number of dictionaries used by SLC problem program.
DDNAMDIC	Specifies for each dictionary the reference to a statement that defines the data-set and the estimated average length of one entry.
NUMCYCLE	Number of SLC cycles to be performed for the execution of problem programs.
SLCMAINn	SLC main program name associated to the n-th cycle.
PARTABxx	Paradigm table name used for input (IN) and output (OU).
NMDELIM	Number of delimiters used for separating logical text units.
DELIMTAB	Hexadecimal value of headword 1 of each delimiter.
BLKSIZE	Standard block size of utility data-sets.
LLDELIMS DELIMSTR	Length-1 and configuration of character string used as end of text unit symbol.

All the options are recorded on punched cards and are part of job input stream, that contains, in addition, the control cards for the definition of data-sets.

INPUT TEXT

```
**************************************** INPUT TEXT **********************
   ==RIF==1812270 =$CAT=$001
 1 41812270$S$IMPROVEMENTS $S$IN $S$OR $S$RELATING $S$TO $S$THE          NSA010
 2 41812270$S$RECOVERY $S$OF $S$CAESIUM .                                NSA020
   =$CAT=$002
 3 71812270$S$A METHOD IS GIVEN FOR SEPARATING CESIUM FROM AN            NSA030
 4 71812270AQUEOUS SOLUTION , ESPECIALLY $M$CS$E1$137 FROM A             NSA040
 5 71812270SOLUTION OF FISSION PRODUCTS . $M$THE METHOD                  NSA050
 6 71812270COMPRISES CONTACTING THE SOLUTION WITH A                      NSA060
 7 71812270WATER-IMMISCIBLE SOLVENT FOR CESIUM POLYBROMIDES IN THE       NSA070
 8 71812270PRESENCE OF A BROMIDE SALT AND BROMINE , WHEREBY THE          NSA080
 9 71812270CESIUM FORMS POLYBROMIDES AND IS EXTRACTED , AND              NSA090
10 71812270REMOVING THE EXTRACTED CESIUM FROM THE SOLVENT                NSA100
11 71812270BY STEAM STRIPPING OF THE FREE BROMINE . $M$THE               NSA110
12 71812270SOLVENT PREFERABLY IS NITROBENZENE WITH A HEAVY INERT         NSA120
13 71812270DILUENT ( $S$C$M$ER$I1$4 OR $S$C$I1$2$S$H$I1$2$M$BR$I1$4 )   NSA130
14 71812270ADDED TO INCREASE ITS DENSITY . $M$THE METHOD IS              NSA140
15 71812270HIGHLY SPECIFIC TO CESIUM  AT 90( EXTRACTION ,                NSA150
16 71812270ONLY 10( OF THE RUBIDIUM IS EXTRACTED AND LESS THAN 1(        NSA160
17 71812270OF ANY OF THE OTHER METALS . $S$A FLOW SHEET IS               NSA170
18 71812270INCLUDED . ( $S$D.L.C. ) .                                   NSA180
   ===
```

Each statement of input text is structured as HEADER LABEL - TEXT. The first character of header label indicates the class of text part following the label. The other characters of the label indicate the reference number of the document.

The text has been recorded with conventional codes written as x. Example:

- S All the word is written with capital letters,
- M The first letter of the successive word is a capital,
- $E1$ Exponent 1,
- $I1$ Index 1.

RESULTS OF TEXT ANALYSIS PHASE

After the text analysis phase, the following messages are printed and can be used for statistics:

```
NUMBER OF CYCLE PROCESSED IN INPUT                                    1
NUMBER OF CURRENT WORDS PROCESSED IN INPUT                          166
NUMBER OF NON-WORD ITEMS PROCESSED IN INPUT                          32
NUMBER OF DIFFERENT WORDS PROCESSED IN INPUT                         80
NUMBER OF ERRORS ENCOUNTERED DURING ANALYSIS                          0
EODAD CONDITION ON INPUT TEXT - NO RESTART
```

When the corresponding option has been specified, the list of input words is also displayed as follows:

```
LIST OF DIFFERENT WORD-ITEMS SELECTED BY TEXT ANALYIS PROGRAM
.                                                    RELATING
(                                                    REMOVING
$E1$                                                 RUBIDIUM
$I1$                                                 SEPARATING
$M$                                                  STRIPPING
$S$                                                  SOLUTION
)                                                    SPECIFIC
                                                     SOLVENT
,                                                    SHEET
=$CAT=$                                              STEAM
==RIF==                                              SALT
AQUEOUS                                              THAN
ADDED                                                THE
AND                                                  TO
ANY                                                  WATER-IMMISCIBLE
AN                                                   WHEREBY
AT                                                   WITH
A                                                    1)(
BROMIDE                                              1(
BROMINE                                              9J(
BY
CONTACTING
COMPRISES
CAESIUM
CESIUM
DENSITY
DILUENT
D.L.C.
ESPECIALLY
EXTRACTION
EXTRACTED
FISSION
FORMS
FLOW
FREE
FROM
FOR
GIVEN
HIGHLY
HEAVY
IMPROVEMENTS
INCLUDED
INCREASE
INERT
ITS
IN
IS
LESS
METALS
METHOD
NITROBENZENE
OTHER
ONLY
OF
OR
POLYBROMIDES
PREFERABLY
PRESENCE
PRODUCTS
RECOVERY
```

RESULTS OF DICTIONARY LOOK-UP PHASE

When the corresponding option has been specified in input, the list of word-items that have not been found in the morphological dictionary is displayed as follows:

```
LIST OF WORD-ITEMS NOT FOUND IN DICTIONARY
$I1$
$E1$
,
)
(
ADDED
CONTACTING
CAESIUM
COMPRISES
DILUENT
E.I.C.
EXTRACTED
INCLUDED
IMPROVEMENTS
INCREASE
POLYBROMIDES
PRESENCE
PREFERABLY
RELATING
REMOVING
TO
WATER-IMMISCIBLE
WHEREBY
1(
10(
90(
```

For the entries requested and not found in the successive dictionaries, only the lexical number is displayed.

The average length of dictionary entries is computed and displayed. These messages are useful for a subsequent job that uses the same dictionaries.

```
          MESSAGES FROM FIRST  DICTIONARY LOADER
THE DICTIONARY ENTRY WITH THE LXN                1228      WAS NOT FOUND
THE DICTIONARY ENTRY WITH THE LXN                6192      WAS NOT FOUND
THE DICTIONARY ENTRY WITH THE LXN               13000      WAS NOT FOUND
THE DICTIONARY ENTRY WITH THE LXN               13018      WAS NOT FOUND
THE DICTIONARY ENTRY WITH THE LXN               13021      WAS NOT FOUND
THE DICTIONARY ENTRY WITH THE LXN               13025      WAS NOT FOUND
THE DICTIONARY ENTRY WITH THE LXN               13044      WAS NOT FOUND
THE DICTIONARY ENTRY WITH THE LXN               13065      WAS NOT FOUND
THE DICTIONARY ENTRY WITH THE LXN               13072      WAS NOT FOUND
THE DICTIONARY ENTRY WITH THE LXN               13077      WAS NOT FOUND
THE DICTIONARY ENTRY WITH THE LXN               13087      WAS NOT FOUND
THE DICTIONARY ENTRY WITH THE LXN               13089      WAS NOT FOUND
THE DICTIONARY ENTRY WITH THE LXN               13115      WAS NOT FOUND
THE DICTIONARY ENTRY WITH THE LXN               13116      WAS NOT FOUND
THE DICTIONARY ENTRY WITH THE LXN               13137      WAS NOT FOUND
THE DICTIONARY ENTRY WITH THE LXN                1317      WAS NOT FOUND

          MESSAGES FROM SECOND DICTIONARY LOADER

          MESSAGES FROM THIRD  DICTIONARY LOADER

THE NEW VALUE FOR LENGTH OF DIC3 ENTRY IS  13
```

RESULT OF THE PROBLEM PROGRAM

The list of the keywords selected by the program and assigned to the document is displayed. The number associated to one keyword indicates the relative weight assigned to it.

DOCUMENT NUMBER 1812270

004 - CESIUM
003 - CESIUM BROMIDES
003 - RECOVERY
002 - SOLUTIONS
002 - SOLVENTS
002 - BROMINE
002 - FLUID FLOW
002 - FISSION PRODUCTS
002 - FLOW SHEET
001 - NITROBENZENE
001 - WATER
001 - GAS FLOW
001 - SALTS
001 - BROMIDES
001 - METALS
001 - LIQUID FLOW
001 - SHEETS
001 - STRIPPING
001 - RUBIDIUM
001 - DENSITY
001 - FISSION
001 - STEAM

CONCLUSIONS

The SLC-II System is self-consistent in applications in which the final result is used by man (e.g. machine translation). In the case of its usage in information retrieval (document and fact-retrieval), the SLC-II is combined with a data base management and retrieval package which is being implemented at CETIS. The completion of this package is scheduled for early 1974 in the batch version and for 1975 with the conversational interactive extension for the entire system.

BIBLIOGRAPHY

/1/ BROWN, A.F.R.; ;The SLC System for Machine Translation", (1965), EUR 2428 e

/2/ PERSCHKE, S.; "The Computer Programs of the SLC System for Machine Translation", (1965), EUR 2583 e

/3/ PERSCHKE, S.; "SLC-II Eine Software für linguistische Datenverarbeitung", Fachtagung "Information Retrieval Systeme", u. "Management Information Systeme", Gesellschaft für Informatik, Stuttgart 9-11/12/1970

/4/ PERSCHKE, S.; "SLC-II One more Software to Resolve Linguistic Problems", International Meeting on Computational Linguistics", 4-7/9/1971, Debrecen

/5/ PERSCHKE, S.; "SLC-II, A Programming System for Natural Language Text Processing. A Comparison with Previous Special Purpose Programming Languages.", International Computing Symposium, (1972). ACM, Venezia, 12-14/4/1972

/6/ PERSCHKE, S.; "A Generalized Information Retrieval System and the Associated Software", Atti del Seminario "Sistemi di riperimento e selezione automatica dell'informazione." Accademia dei Lincei, Roma. In press.

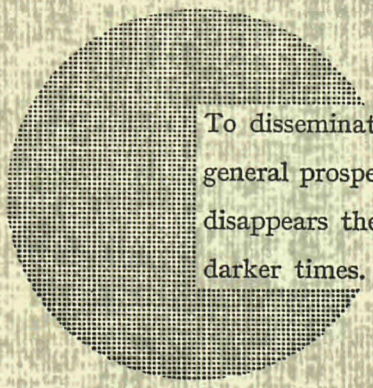
/7/ Joint Research Centre Annual Report, 1971 EUR 4842 e

/8/ Joint Research Centre Annual Report, 1972, EUR 5060 e

NOTICE TO THE READER

All scientific and technical reports published by the Commission of the European Communities are announced in the monthly periodical **"euro-abstracts"**. For subscription (1 year : B.Fr. 1 025,—) or free specimen copies please write to :

Office for Official Publications
of the European Communities
Boîte postale 1003
Luxembourg
(Grand-Duchy of Luxembourg)



To disseminate knowledge is to disseminate prosperity — I mean general prosperity and not individual riches — and with prosperity disappears the greater part of the evil which is our heritage from darker times.

Alfred Nobel

SALES OFFICES

The Office for Official Publications sells all documents published by the Commission of the European Communities at the addresses listed below, at the price given on cover. When ordering, specify clearly the exact reference and the title of the document.

UNITED KINGDOM

H.M. Stationery Office
P.O. Box 569
London S.E. 1 — Tel. 01-928 69 77, ext. 365

BELGIUM

Moniteur belge — Belgisch Staatsblad
Rue de Louvain 40-42 — Leuvenseweg 40-42
1000 Bruxelles — 1000 Brussel — Tel. 12 00 26
CCP 50-80 — Postgiro 50-80

Agency :
Librairie européenne — Europese Boekhandel
Rue de la Loi 244 — Wetstraat 244
1040 Bruxelles — 1040 Brussel

DENMARK

J.H. Schultz — Boghandel
Møntergade 19
DK 1116 København K — Tel. 14 11 95

FRANCE

Service de vente en France des publications des Communautés européennes — Journal officiel
26, rue Desaix — 75 732 Paris - Cédex 15e
Tel. (1) 306 51 00 — CCP Paris 23-96

GERMANY (FR)

Verlag Bundesanzeiger
5 Köln 1 — Postfach 108 006
Tel. (0221) 21 03 48
Telex: Anzeiger Bonn 08 882 595
Postscheckkonto 834 00 Köln

GRAND DUCHY OF LUXEMBOURG

Office for Official Publications of the European Communities
Boîte postale 1003 — Luxembourg
Tel. 4 79 41 — CCP 191-90
Compte courant bancaire: BIL 8-109/6003/200

IRELAND

Stationery Office — The Controller
Beggar's Bush
Dublin 4 — Tel. 6 54 01

ITALY

Libreria dello Stato
Piazza G. Verdi 10
00198 Roma — Tel. (6) 85 08
CCP 1/2640

NETHERLANDS

Staatsdrukkerij- en uitgeverijbedrijf
Christoffel Plantijnstraat
's-Gravenhage — Tel. (070) 81 45 11
Postgiro 42 53 00

UNITED STATES OF AMERICA

European Community Information Service
2100 M Street, N.W.
Suite 707
Washington, D.C. 20 037 — Tel. 296 51 31

SWITZERLAND

Librairie Payot
6, rue Grenus
1211 Genève — Tel. 31 89 50
CCP 12-236 Genève

SWEDEN

Librairie C.E. Fritze
2, Fredsgatan
Stockholm 16
Post Giro 193, Bank Giro 73/4015

SPAIN

Libreria Mundi-Prensa
Castello 37
Madrid 1 — Tel. 275 51 31

OTHER COUNTRIES

Office for Official Publications of the European Communities
Boîte postale 1003 — Luxembourg
Tel. 4 79 41 — CCP 191-90
Compte courant bancaire: BIL 8-109/6003/200

OFFICE FOR OFFICIAL PUBLICATIONS OF THE EUROPEAN COMMUNITIES
Boîte postale 1003 — Luxembourg 6457

CDNA05116ENC