TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Eindhoven University of Technology

MASTER

The recognition of the commands to the interactive program package SATER

Escher, H.A.

Award date:
1980

Link to publication

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 27. Apr. 2025

4107

Group Measurement and Control
Department of Electrical Engineerig
Eindhoven University of Technology
Eindhoven, The Netherlands

THE RECOGNITION OF THE COMMANDS
TO THE INTERACTIVE PROGRAM
PACKAGE SATER

by H.A.Escher

Submitted in partial fulfillment of the requirements for the degree of Ir. (M.Sc.) at the Eindhoven University of Technology.
The work was carried out in the Measurement and Control Group under the directorship of Prof.ir. F.J. Kylstra.
Advisors: Ir. A.J.W. van den Boom,
Ir. J.J. van Nunen.

336040

Summary.

This report deals with two different aspects of a program package for control theory. The first part deals with Sater, an already existing program package for interactive use. An imperfection of this package was the incorrect recognition of some user commands, namely of names.
A new interpreter has been designed and implemented, which uses a tree-like structure to store the permitted names. It does not recognize the words first, and the word number sequence next; it recognizes the names in one step.
The second aspect of a program package is the language which is used to write the programs. A preliminary study is made of a special programming language for control theory. The aims of such a language and some of the desired characteristics are presented in the last part of this report.

CONTENTS

Chapter		Page
1.	Introduction	5
2.	Sater, an interactive program package	6
2.1.	History, aims and specifications	6
2.2.	Structure of Sater	8
2.2.1.	The supervisor	9
2.2.2.	The application programs	10
2.2.3.	System Service Routines	10
2.3.	Relations between the application programs	11
2.4.	The use of tables to describe alterable data	12
2.5.	Problems in the recognition of names	17
2.6.	Storage structure of Sater	19
3.	Specification, design and realization of the interpreter	25
3.1.	Specifications of the interpreter	25
3.2.	Storage and recognition of the commands	30
3.2.1.	Separate recognition method	31
3.2.2.	Combined recognition method	33
3.3.	Comparison of the separate and combined recognition	35
3.3.1.	Separate recognition method	36
3.3.2.	Combined recognition method	40
3.3.3.	The selection of the storage structure for names	41
3.4.	Description of the data structure and program	43
3.4.1.	The data structure	43
3.4.2.	The interpreter programs	44
3.4.3.	Influences of the new interpreter on other Sater programs and data structures	48

Chapter		Page
4.	The table generation program TABGEN	49
4.1.	The structure of the tables	49
4.2.	Specification of the contents of the tables	50
4.3.	Temporary storage of data	54
4.4.	The algorithm to build the tree	56
4.5.	The TABGEN programs	61
5.	A special language for control theory (Dutch)	67
5.1.	Motivation and aim	67
5.2.	The design of a program	69
5.3.	Additional aspects of the language	70
6.	Conclusions and recommendations	74
Survey of the symbolic names.		77
Literature.		79

APPENDICES

A	Memory map of Sater	81
B	Survey of the names in Sater	83
C	The memory requirements for the combined and separate recognition method	85
D	Syntax of the input file SATNAMES.DAT	92
E	Error-messages during the generation of the tables	94
F	Survey of the files where the subprograms of TABGEN are stored	98
G	Changes, introduced in the Sater programs	99

1. Introduction.

Sater is an interactive program package for control theory. It consists of a general-purpose framework in which control theory programs are inserted. The programs are for instance parameter extimation routines, a simulation program, and programs to visualize the results, e.g. by means of a pole/zero or Bode diagram.

The package has been designed and implemented in the group ER of the Department of Electrical Engineering of the Eindhoven University of Technology. It has been copied by two other groups, of which one outside our university. Several changes have been introduced by them in order to adapt the program package to their computer system.
During the years the package has been used now, a few imperfections and limitations have been discovered.
One of the imperfections was that the supervisor might make mistakes during the recognition of some user commands.
A limitation that is felt when using Sater, is the lack of the possibility to combine a number of operations into one superroutine, which executes these operations sequentially. Another limitation is that at the same time only one copy of a certain dataset may be present in the system. When a new copy is created, the old one is deleted automatically. The limitations may be lifted by the use of a special programming language for control theory.

The subjects discussed in this report, originate from the abovementioned problematic.

2. Sater, an interactive program package.

2.1. History, aims and specifications.

Sater is a program package for interactive use. It originated from the need to combine the results of the work that is performed in the group ER in several fields of control theory and that results in computer programs. Before Sater was used, almost all the computations for e.g. parameter estimation were performed on the central computer of the university. Every programmer had only his own program in mind, with the consequence that results of one program were difficult to use as inputdata for another program. So, for every new or slightly different problem a new program was written. After being used to solve the specific problem, it disappeared somewhere on a bookshelf, and was not used for similar problems. Furthermore, it was very difficult for someone without experience in programming to become acquainted with the different techniques of the control theory. He had to spent a substantial part of his time available to write programs before being able to denote himself to control.
For these reasons it became clear that the best solution is an interactive program package. This has many advantages:

- it makes the step to use a computer easier;
- it offers students the opportunity to apply the theory in practice;
- it offers researchers the opportunity to examine new ideas on their implications in a fast and easy way;
- many programs as well as recent developments become available to many people.

The first version of the program was written for a PDP8-1 minicomputer with 8k byte of core memory. The programming language was Rog-algol, a subset of Algol. After a while this computer proved to be too small and too slow to satisfy the demands, because new and bigger programs were added to the package.

A second version of the package, called Sater, was designed for a PDP11/20 minicomputer by v.d. Boom and Lemmens (Litt. 1, 2 and 3). This computer has 16k of main memory and runs under the RT11 Operating System, which is a single-user O.S. The programming language is Fortran, because this is the only suitable high-level language that is supported by the manufacturer of the computer. The package has been copied by two institutes: the subfaculty of Psychology of the University of Tilburg (Litt. 4), and the group System and Control Engineering of the Department of Technical Physics of our own university (Litt. 5). They have adapted the package for use under the RSX-11 multi-user operating system. The copy, adapted by Bollen (Litt. 5) will also be used on our new computer, a PDP11/60 that runs under the RSX-11M operating system.

The design of the second version was based on the following aims:

1. Everyone should be able to use the system, irrespective his knowledge of programming languages.
2. The system should constitute a library of computer-programs; these programs should be able to exchange data.
3. The system should give the opportunity to many programmers to make a contribution to the package by means of an application program, without burdening them with problems concerning the interaction, graphical display of results or data transport between programs.

The specifications, resulting from these aims, are:

ad 1 - A manual should be superfluous; this means that the action of the user must be self-evident, or must be directed by the system at the moment it is needed. When the user doesn't know what to do, he should have the opportunity to ask for further explanations.
The number of actions performed by the user should be as little as possible. When it is clear what the next step will be, it should be pleasant if the system makes this step

without asking. If a question has almost always the same answer, the system should have this as a default-answer.
The user should be able to communicate with the system in a 'natural language'. The system should be able to ask what the user needs; so it should not have to ask to run a program, but it should also be able to ask for specific data.
When an abbreviation is sufficient to recognize a name, the system has to accept this, as well as when there exist more names for an operation or specific data.
The system should be insensible for errors of users; it means e.g. when a question has only a limited number of valid answers, any other answer must be rejected and a new answer must be requested; this also applies when the boundaries of numerical values are exceeded.

ad 2 - The application programs should be easily accessible; it should be possible to update the library easily; the programs should be reliable, and exchange of data between programs should be simple.

ad 3 - The reliability of the system may not be effected by errors in application programs.
The programmers of application programs should be freed of the burden of performing data transport between the system and the peripheral devices. For this purpose he should have a set of service routines for I/O, data exchange and communications with the user.
The package should have an error message system, so that the programmer can perceive and correct his errors.

2.2. Structure of Sater.

To satisfy the aims and specifications from the previous paragraph, the package has been constructed as a framework in which programs can be inserted. The system can be divided into three parts: a supervisor, a set of application programs, and a set of service routines.

The supervisor and service routines constitute the framework; the hierarchical structure is rendered in fig. 1. The design of the framework is such that it can be used for many application fields; only the application programs determine in what field it will be used.

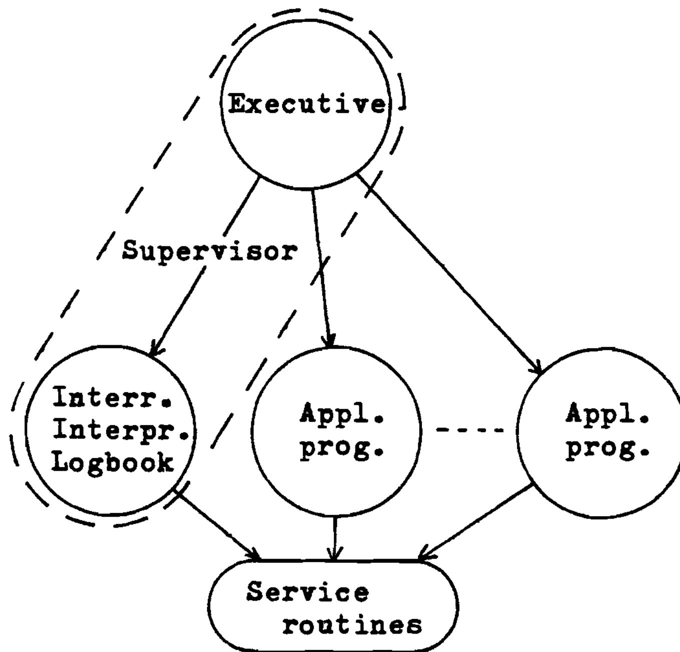


fig. 1: hierarchical structure of Sater.

2.2.1. The supervisor.

The program package contains a number of application programs which may have mutual relations, because they may use eachother's data. These relations are described by means of tables. The task of the supervisor is, by using these tables, to start the correct application program at the correct moment according to the wishes of the user.

The user can express his wishes during the conversation with the supervisor via the interrogator/interpreter. When an application program has been executed, new data have been created; this fact is listed in the logbook. The supervisor uses this logbook to verify whether or not an application program may be started, depending on the availability and validity of the inputdata.

The software for updating the logbook and for conversation with the user is a subprogram of the executive. The interpreter is also used by other programs.

2.2.2. The application programs.

The application programs determine the field of use of the package. In our case, this is the control theory. These programs are mainly numerical programs to process the data, e.g. parameter estimation and simulation. However, some programs serve for sampling of continuous signals or for file I/O; for this last application also the interpreter is used.

2.2.3. System Service Routines.

To ensure the reliability of the package and to relief the programmer of a heavy burden with respect to I/O and interactive conversation, a set of service routines has been written for performing these tasks. Every programmer must use these routines, and may not write his own routines to access devices or common data, in order to protect these devices and data against erroneous use.

The service routines can be subdivided into three groups: the I/O-system, the question and answer subsystem and the graphical subsystem. The I/O-system is in charge of the transport of data from and to the background memory and the interactive terminal. The question and answer subsystem asks questions, accepts answers and examines them on their admissibility. The graphical subsystem finally is in charge of the graphical presentation of data on the display and of the input of coordinates from the display.

2.3. Relations between the application programs.

For the solution of a range of problems from control theory, a set of programs is needed. As is mentioned already in 2.2. mutual relations may exist between the application programs; a numerical computer program, such as simulation, calculation of poles and zeroes, etc. can be regarded as an operator that transforms a set of numerical data (input data) into a different set of numerical data (output, result). In order to obtain a specific result, given some set of data, it may be necessary to run subsequently several programs, each one operating upon the result of a previous program, and after using additional data as well.
A number of rules has been laid down for the relations between the datasets, and the programs operating on these sets, the so-called operations:

- Each operation generates one, and not more than one, dataset. Most operations generate, each time they are performed, the same type of dataset; for a few others, the copy operations; this is not known in advance.

- An operation needs, depending on its type, none, one or several datasets as its input. The number and type of datasets is fixed in advance, except for the copy operations, where the type is not fixed. All the necessary datasets must be present before an operation may be performed.

- A particular dataset may be necessary as input for no, one or several different operations, depending on the type of dataset. This does not mean that the operations which may use this dataset, also must be performed.

- A dataset may be generated by one or more operations. This means that for obtaining this dataset anyone of the operations must be performed.

To visualize these relations between operations and datasets a graph may be drawn for the control theory; see fig. 2.
This graph constitutes the blue-print for the contents of our program package for control theory.

2.4. The use of tables to describe alterable data.

A program package like Sater is not a rigid and unalterable construction. It will be subject to continuous change, as new numeric programs become available and old programs become obsolete. So, programs should be easily inserted or deleted and the stored package description easily be modified. The same holds for the many pieces of text that are used throughout the interaction with the user in the form of warnings, questions, error messages, etc. They should be easily accessible for prompt display, but they should also be easily changed by programmers. All this is realized by using tables that are stored by the supervisor software before accepting user commands and executing calculation programs.
They are created by external programs, which are not part of the program package itself. The tables can be divided into a number of groups: a vocabulary, a set of dataset tables, a set of operation tables, a logbook, and a text pointer table. The use of tables allows the flexibility mentioned above, as tables can be updated very easily, and it doesn't introduce any changes in programs that have proven 'correct'. For instance, extension of the package with additional numeric programs requires the simple extension of the tables describing the relations between the operations and datasets, and the extension of the tables that are used to recognize and display names. The program package itself must be generated again by the Task Builder with inclusion of these new programs, but without affecting the existing programs.

The vocabulary contains a series of numbered words. Apart from sets of synonyms - which have the same number - every word has a different number. The vocabulary tables are constructed in such a way that retrieval of a word number if the word is supplied is relatively

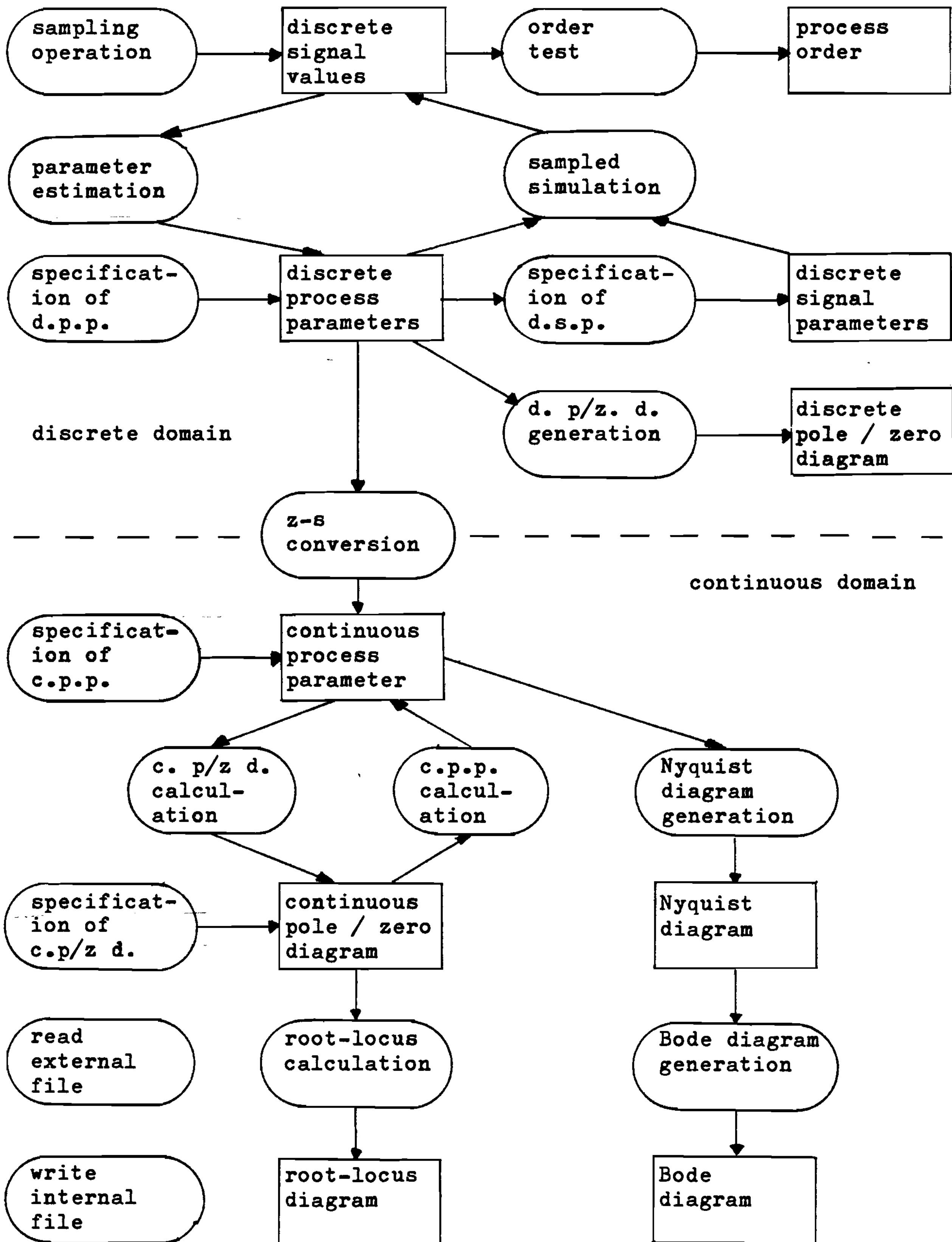


fig. 2: relations between datasets and operations.

easy, as is retrieval of the word as a string of characters if its number is given. Combinations of words of the vocabulary are used in the dataset and operation names.
Datasets are numbered too, and by the number mechanism it is possible to assign different names to the same dataset, precisely like the words of the vocabulary. Apart from their number, datasets may be identified by the numbers of the words that constitute the dataset name. These numbers reside in the dataset table, together with the numbers of the operations that yield the dataset in question as a result. In much the same way as in the vocabulary the entries of the dataset table may be accessed by name (via the vocabulary) or by number.
The operation table has only one entrance and this is by way of the operation number. The table contains the word numbers of the operation name and the numbers of the datasets that are needed to perform the operation.
All these tables are static. They do not change during an interactive session.
The logbook however, is updated every time a numeric program is executed. So it contains a record of the way the descriptive graph is traversed from the beginning of a session. The tables are used to decide what program should be executed in response to a text string, typed by the user.

The use of the tables will be clarified from the conversation between the interrogator and the user, cf. fig. 3.
The user types a name after a request of the interrogator to type a dataset name or operation number. The name consists of a number of abbreviated keywords, DISCR. SIGN. VAL., which stands for 'discrete signal values'. The interpreter looks up the word numbers of each of these words in the vocabulary. It finds the numbers 7, 5, 6. This sequence of word numbers is used by the interpreter to find the name in the dataset table; the result is dataset number 4. Next the interrogator will display the Sater name of this dataset, using the word numbers obtained from the dataset table and using

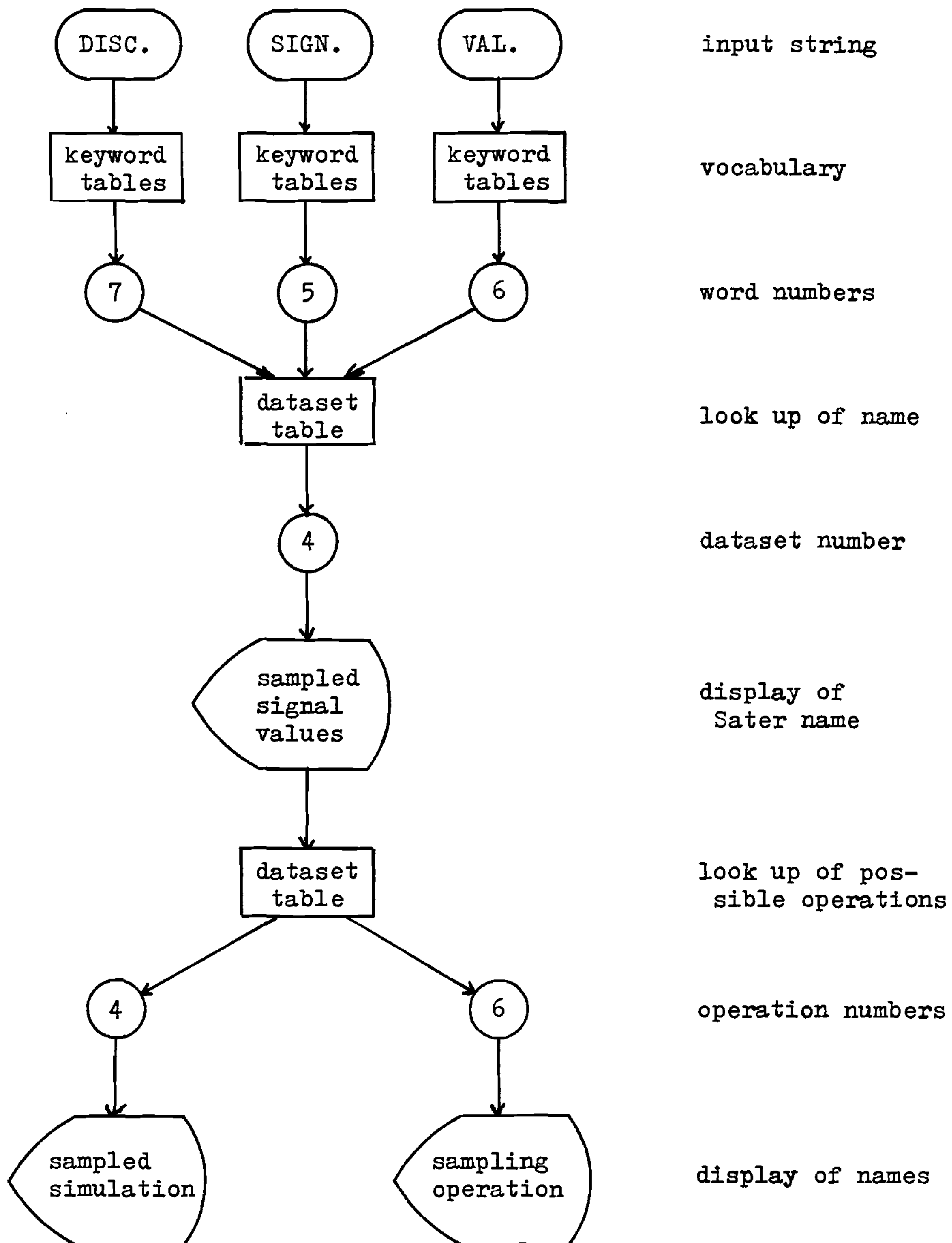


fig. 3a: the use of tables

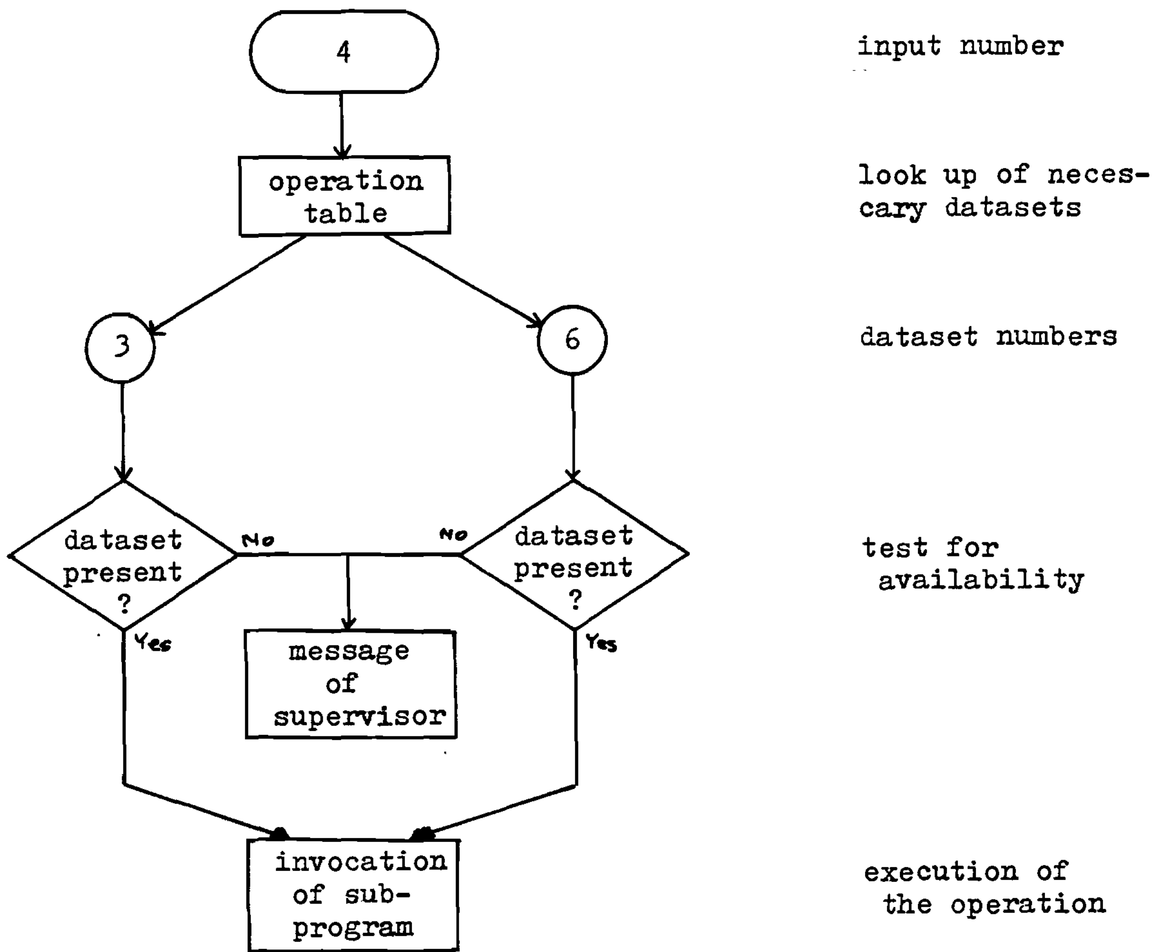


fig. 3b: the use of tables

the vocabulary; it displays: 'sampled signal values'.
The dataset table also contains the numbers of the operations that may create this dataset; the numbers are 4 and 6. The numbers and names of the operations are displayed by the interrogator that uses the operation table and vocabulary; they are '4: sampled simulation' and '6: sampling operation', out of which the user must make a choice by typing the operation number.

The user types '4', and the interrogator looks in the operation table which datasets are necessary as input for this operation. These are datasets 3 and 6. The logbook is examined whether these datasets are already present. When both are present, the operation 'sampled simulation' will be performed. After completion of the operation the presence of dataset 4 will be recorded in the logbook.
When an input dataset is not yet present, this one must be created first. The interrogator will give the list of operations that may create this dataset, using the dataset number.

Sater uses also messages during its conversation with the user. Every message and question that can be displayed by the system, is identified by a number. A subroutine, which may be called by any part of the system will display a message on the terminal if provided with the message number. Therefore it has to consult the stored message pointer table which indicates the position of the message text in a text file on disk.
By using a different file for the same messages in another language a switch can be made from one language to another, e.g. from Dutch to English.
In our system disk files, used to store system tables, messages and results are structured according to certain rules. This allows files that are created by one program to be opened by any other program by using standard subroutines. Questions to the user are also handled by common subroutines. Programmers only need to incorporate the right subroutine calls with the right arguments in their program to have questions displayed, default values provided

or answers from the keyboard read and checked. The same holds for graphic or numeric output of data. Moreover, some numeric functions and subroutines are also shared by different programs. In this way, programmers of numeric software are troubled as little as possible with requirements which are not characteristic for the purpose of their program, but which are nevertheless vital for the functioning of the interactive system.

2.5. Problems in the recognition of names.

The description given above of the recognition of names is the ideal case; in practice however, some problems can arise. To indicate the origin of this, a more detailed description will be given of the way the words and names are stored and accessed in the original system (Litt. 2).
Only dataset names could be recognized. Using the vocabulary, the words of the name were recognized first; and the resulting word number sequence was recognized next in the name table.
The vocabulary consists of three tables, KWP, KW and NKW, cf. fig. 4.
KW is an array of characters in which the words are stored, separated from eachother by a ' Ø ' character.

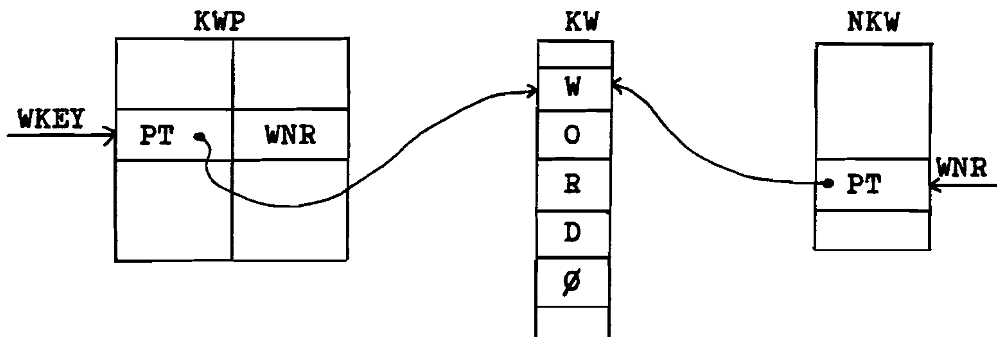


fig. 4: vocabulary tables.

Given the word number WNR, the start address of this word INKW can be found in NKW and is used for display.
In order to compare a word in the input line with the words in KW, KWP is used as entrance to KW. The key WKEY is derived from the

word in the input line according to a functional relation, i.e. the key is calculated according a specific function that uses certain characteristics of the word; these are here the first and second letter:

$$WKEY = (dimKWP/32) * (((1^{st} \text{ letter EXOR } 2^{nd} \text{ letter}) AND31) + 1)$$

in which dimKWP is the dimension of the table KWP.
This way of addressing is called 'hash-addressing'.

It is possible that the same key is calculated for different words; the pointers of these conflicting words are stored in the same table on the next free place. Together with the pointer to the word in KW, also the word number WNR is stored in KWP.
When a word is looked up in the vocabulary, first the key WKEY is calculated and the pointer PT is obtained. When PT is Ø, the word is not present in the vocabulary. Otherwise, the input word is compared with the word where PT points at. If they are similar, WNR is the word number. In case the words are different, the pointer of the next place in KWP is used to get the next word of the vocabulary for another comparison, because several words may have the same key. This search procedure is continued until the word is found or until PT is Ø, i.e. the word is not present.

The word number sequences are processed in almost the same way as the words. The number sequences are stored in an array DS; access to the sequences is obtained via the table DSP, using the functional relation:

$$DSKEY = ((WNR1 \text{ AND } 31) + 1) \times (dimDSP/32),$$

in which dimDSP is the dimension of the table DSP, and WNR1 is the word number of the first word of the name; cf. fig. 5.

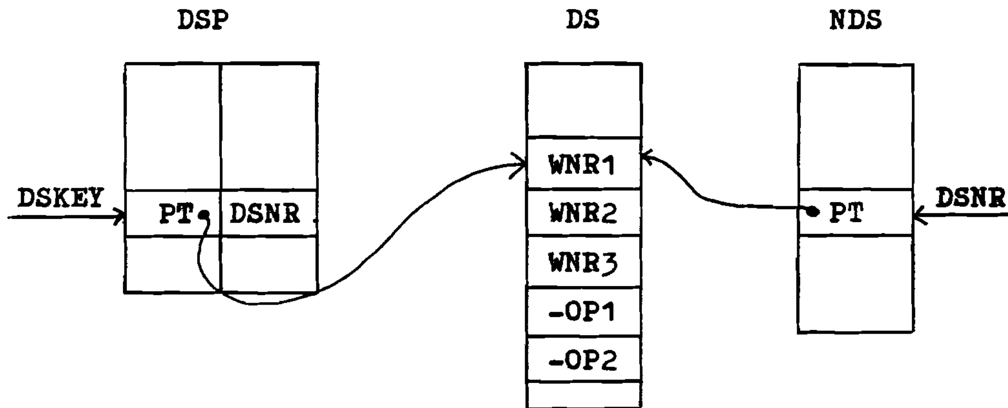


fig. 5: dataset name tables.

The numbers of the operations that may create the dataset are also stored in the array DS. The recognition procedure of a word number sequence is similar to that of words.

When a name has been typed on the keyboard, each of the words is looked up separately. When all words are recognized,the word number sequence is processed to find a name in it. When an abbreviation of a word is used, it is possible that it fits several words; however the recognition mechanism takes the first word which fits the abbreviation.
It may happen that this is not the word intended by the user. If so, it is very likely that the name cannot be recognized. So, in spite of the fact that a correct name is typed in an abbreviated way, the interpreter cannot find the name. On the other hand it sometimes happens that the interpreter will not detect that more names fit the input name, when words are abbreviated or when a name is not typed completely.

2.6. Storage structure of Sater.

Before discussing the storage structure of Sater itself, we'll first give a survey of the addressing mechanism of a PDP11 computer under the RSX-11M operating system, cf. the Task Builder Reference Manual (Litt. 6).

The primary addressing mechanism of a PDP11 processor is the 16-bit computer word. The maximum address space that the processor can reference at any time is a function of the length of this word. The highest number that can be represented in 16 bits is 65535. Because the PDP11 is a byte-addressable machine, the 16-bit word length allows it to address up to 65535 bytes (= 32 k words) of address space at any one time. The amount of address space that a machine can reference at any one time is called virtual address space. The physical address space may be larger than 32k when a 'memory management unit' is used.
An address space of 32k words may be too limited to contain a whole program; in order to execute programs bigger than 32k, an 'overlay' must be used. Several program segments which are logically independent - i.e. the components of one segment cannot reference the components of the other segment - will share now the same virtual addresses. A special mechanism handles the addressing of the correct segments at the correct moment, if necessary after loading of the segments.
The assignment of the virtual addresses is performed by the Task Builder, which generates an executable program from the object modules. The Task Builder supports two types of overlay structures: a disk-resident overlay structure, and a memory-resident overlay structure. The assignment of the virtual addresses is identical for both structures. The differences occur at run-time:

- disk-resident overlay structure: the computer keeps looking at the same memory area, but the contents of the memory changes. The overlay segments reside normally on disk, but share the same physical memory. Whenever a subroutine, residing in an overlay segment, is called, and this segment is not yet present in memory, this segment will be loaded, thus overlaying physically the previously present program segment. The relation between the virtual address space and the physical memory is given in fig. 6a and 6b. A program may be divided into three segments: MAIN, A and B, of which A and B are logically independent. A and B share in this structure the same virtual and physical address space.

- <u>memory-resident overlay</u>: the contents of the memory remains the same, but the computer shifts its scope from one memory area to another. The overlay segments are loaded into memory the first time they are called, and they will reside there and will not be destroyed. Other segments with the same virtual memory addresses are loaded in a physically different memory area. Once all segments in the structure have been called, 'loading' of overlay segments reduces to the remapping of the virtual address space to the physical locations in memory where the overlay segments permanently reside. This type of structure can only be applied when the hardware has a 'memory management unit', which supports the loading of a program everywhere in a memory that is bigger than the virtual address space. In fig. 6a and 6c the relation between the virtual address space and the physical memory has been given for the same program as stated above. Now A and B share the same virtual address space but not the same physical memory.

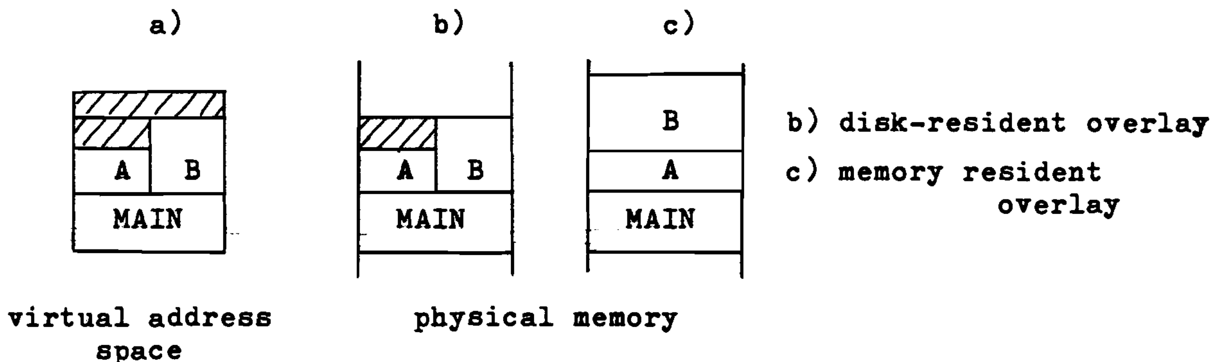


fig. 6.

The disk-resident overlay structure saves physical memory space, but will use more time for program execution because of repeatedly loading of segments. The memory-resident overlay structure will be faster because all segments are loaded only once, but it uses more memory; the physical memory occupied by one program can exceed 32k words. For both overlay structures however, the virtual address space practically available to the program, will be less than 32k,

because also the information on the overlay structure and the routines to handle it, must be stored in this area.
The arrangement of overlay segments within the virtual address space of a task can be represented schematically as a tree-like structure. It consists of a single root segment which is always in memory, and of the overlay segments, which are each represented by a branch, cf. fig. 7. Parallel branches denote segments that overlay eachother and therefore have the same virtual addresses. Routines that may be called by modules on all paths of the tree must be placed in the root. When several of these routines are logically independent, they may be overlayed, but this cannot be achieved in the tree, because they may be called by all other modules. The Task Builder offers the opportunity to generate multiple-tree structures, containing one main-tree, as described above, and one or more co-trees. These co-trees also have a tree-like structure, a root segment residing in memory, and two or more overlay segments. One difference between the main-tree and the co-tree is, that the root segment of the main-tree is loaded by the executive when the task is made active, while the segments within each co-tree are loaded by calls to the overlay run-time routines. Another difference is that the root segment of the co-tree may be a dummy segment, i.e. it may be empty, while the root of the main-tree must contain the main program.

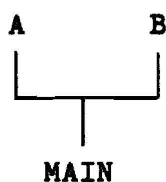
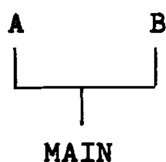
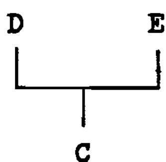


fig. 7.

In fig. 8 the structure is given for a program in which MAIN and the overlay segments A and B call the logically independent segments D and E. In the co-tree description the dummy segment C has been added to form the root; because C is empty it doesn't appear in the virtual storage map.





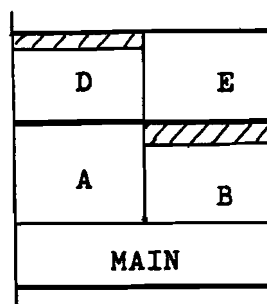
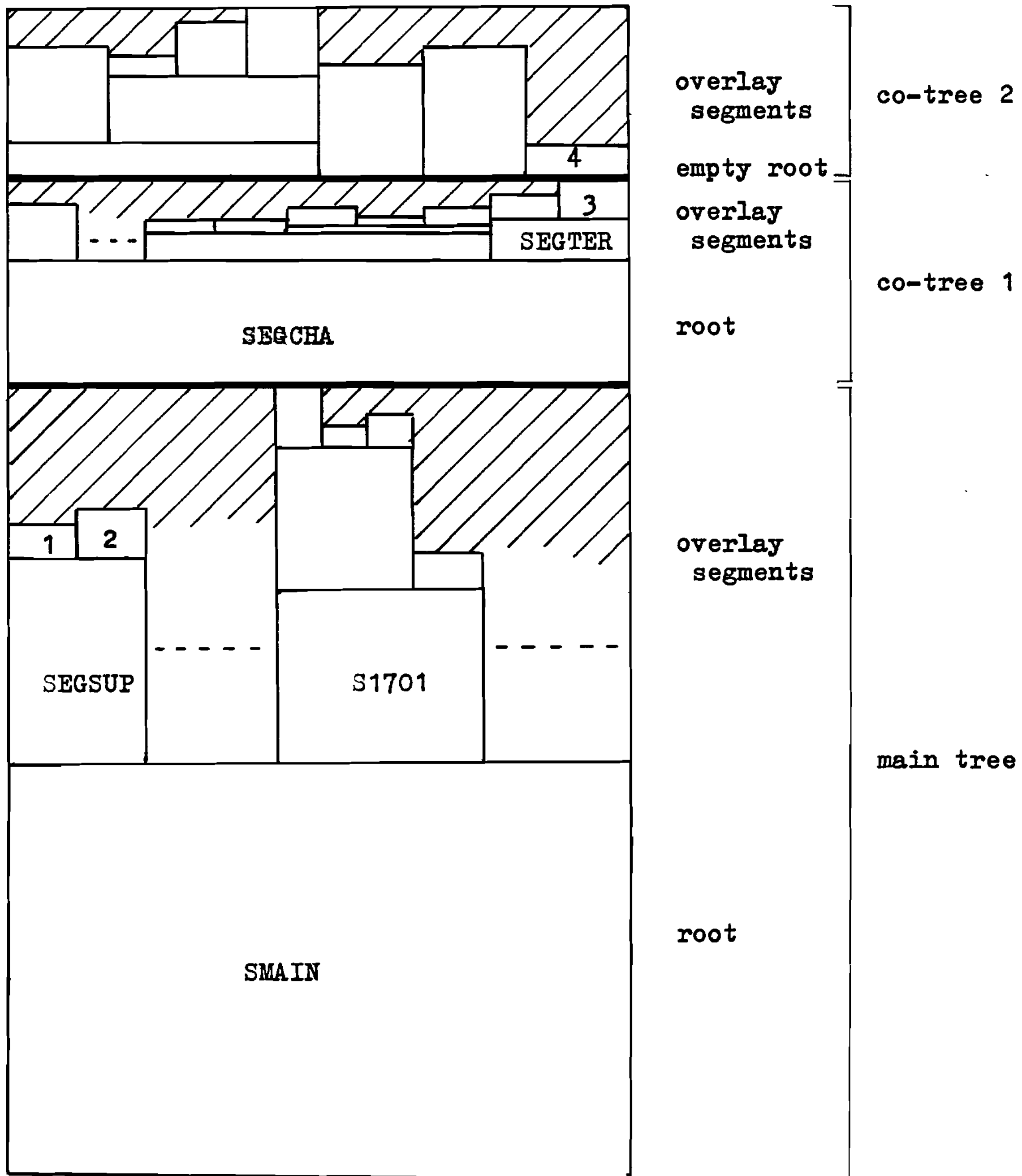


fig. 8.

The structure of the trees is specified to the Task Builder by means of a special language: O D L, the Overlay Description Language.

Also for the Sater program package overlay of the program segments must be used, because the total size is about 125k. P. Bollen has designed a structure to make the package fit in 30k virtual memory (Litt. 4). The structure comprises a main-tree and two co-trees; in the main-tree all application programs are stored, together with a part of the supervisor. The first co-tree contains all the system service routines, while the second co-tree contains routines that are used by several application programs, e.g. statistical routines for the parameter estimation programs.
The virtual memory map of this overlay structure is given in fig. 9, in which only part of the segments is drawn. The total overlay structure is given in Appendix A; it contains also the new interpreter. The data structure is stored in SEGSUP; the routines are stored in SEGINK and SEGNEW.



1: SEGR14 3: SEGINK
2: SEGR15 4: SEGNEW

fig. 9: partial memory map of Sater

3. Specification, design and realization of the interpreter.

3.1. Specifications of the interpreter.

The task of the interpreter is to transform the command which is given by the user on his terminal and which appears to Sater as an ASCII-string, into a code which may be used by the supervisor and two copy programs. The external relations of the interpreter are determined by the communication with the user and with Sater.

User commands.

On request of the interrogator, the user may give a command which is of one of the following four types:

- the name of the dataset to be created;
- the name or number of the operation to be performed;
- an empty line;
- a special command, indicated by ' ? ', '#' or 'ESC'.

The commands serve to specify the course of the session, explicitly by entering a name, number or special symbol, and implicitly by typing an empty line, which is interpreted in a specific way. Operations and datasets are called by name, and operations may also be called by number. Names of operations are for example:
Z-S conversion, specification of continuous pole/zero diagram, root-locus calculation; names of datasets are: Bode diagram, discrete signal parameters, and discrete signal values. A survey of all permitted names is given in appendix B. For the unexperienced user of Sater, these names are very useful because they correspond with his knowledge of the control theory. For the experienced user however, they are a horror because of their length. A short way to enter a command is offered by allowing the user to specify an operation by a number which is unique for every operation. The dataset to be created is determined by the specification of an operation, cf. par. 2.3.

As has been stated in paragraph 2.1 it should be possible to abbreviate names and words, when the abbreviation is sufficient to recognize the name. 'Bode' is sufficient to recognize 'Bode diagram'; as long as it doesn't conflict with other names, it is acceptable. 'Discrete signal' is not acceptable because it fits on two different names: 'discrete signal parameters' and 'discrete signal values'. Words may be abbreviated as usual by replacing the last letters by a dot, e.g. 'discr. sign. par.', or even 'd.s.p.' as long as this doesn't conflict with other names. The last abbreviation also fits on 'discrete system parameters' and consequently is ambiguous.
The admissibility of abbreviations depends on the names that must be recognized by the interpreter. Because the user probably will not know all the names, he may make some mistakes when using abbreviations; the interpreter must perceive these mistakes, so that the interrogator can warn the user and ask for a more specified name.

In order to determine its next step in the session, the interrogator can be satisfied by an empty command line instead of with an explicit command, like a name or a number. The interpretation of this command depends on the place in the session; it may mean that the first answer of a sequence of possibilities must be chosen, or that the interrogator may continue.

The fourth type of command the user may give, consists of three special commands:

'?' - indicates that the user does not understand the question or that he is uncertain about the course of the session. When this command has been given, a message will be displayed that gives further explanations, depending on the place in the session.

'ESC' - results in an immediate return to the beginning of the interrogator.

'#' - results in the termination of the interactive session after a request to confirm this intention.

The other two programs, which may use the interpreter, are copy programs. One program reads an external file and copies it to an internal dataset; the other program copies an internal dataset of Sater to an external file.
On request of the copy programs, the user may give a command, which is one of the following types:

- the name of the dataset to be transferred
- the number of the dataset to be transferred
- an empty line
- a special command, indicated by '?', '#' and 'ESC'.

The name and number are self-evident; a name may be abbreviated in the same way as for the interrogator. The interpretation of an empty line depends on the question which was asked. The special commands '?' and '#' have the same meaning as for the interrogator. The 'ESC' command will result in the termination of the copy program, and in the return to the supervisor.

With the above mentioned data,the following specifications have been laid down for the commands given by the user:

1. A command consists of a name, a number, a special symbol or an empty line.
2. An inputline may contain only one command, and is terminated by a 'CR'.
3. Special symbols are '?', '#' and 'ESC'.
4. A number is minimum one.
5. A name consists of one or more words.
6. A name may be abbreviated till at least one word by omitting the last words.
7. A word consists of one or more letters.
8. A word may be abbreviated till at least one letter by replacing the last letters by a dot.
9. Words are separated from eachother by the separators: space ( ), tab (   ), hyphen (-), slash (/), apostrophe (') and dot (.) which is used to abbreviate words.
   A dot may be followed by a hyphen, slash, or apostrophe.
   Dummy spaces or tabs may be typed before or after separators, except before a dot.

Errors in the inputline should be corrected by the 'DELETE' or 'CTRL/U' key. The interpreter will give an appropriate message when an erroneous inputline has been entered, e.g. when it contains two commands or an illegal character.

Sater relations.

The interface of the interpreter with Sater concerns

1. the memory locations where the data structure and the programs that will use it, may be stored; and
2. the way the interpreter presents its results.

The interpreter routine INKB is called in the same way as in the old situation. The presentation of the results is slightly different,

because an operation may be specified by a name too. The Fortran function INKB (IVAL) reads the keyboard and delivers the result. It executes the special command '#' to terminate the session, and gives an error message in case of an erroneous input string. On return INKB and IVAL may be:

INKB = 1: IVAL = Ø: the command line is empty.
- 2: the command line contains a dataset name; IVAL is the number of the dataset.
- 3: the command line contains a number; IVAL is this number.
- 4: the command line contains the special symbol 'ESC' or '?'; IVAL is the number of the symbol, 1 : ESC, 2 : ?.
- 5: the command line contains an operation name; IVAL is the number of the operation.

Actually there are no restrictions to the size of the data structure and routines, because overlay can be used when a suitable subdivision of data and programs is made.
Looking at the storage structure for Sater as mentioned in par. 2.6, three areas are available for the storage of the data structure and programs; these areas are:

1. the remaining free area in the main-tree, ranging from the top of the section SEGR 15 up to the highest address of the main-tree; this is about 4400 memory words;
2. the memory getting available in co-tree 1 by deleting the old interpreter routines; this is about 640 words;
3. the whole co-tree 2 which is about 4k words.

From the top of the second co-tree to the highest virtual memory address another 2.5k words are available. However, it is unwise to use this area, because it restricts future extensions and changes of the package; only the memory areas which are free within the existing trees, will be used.

3.2. Storage and recognition of the commands.

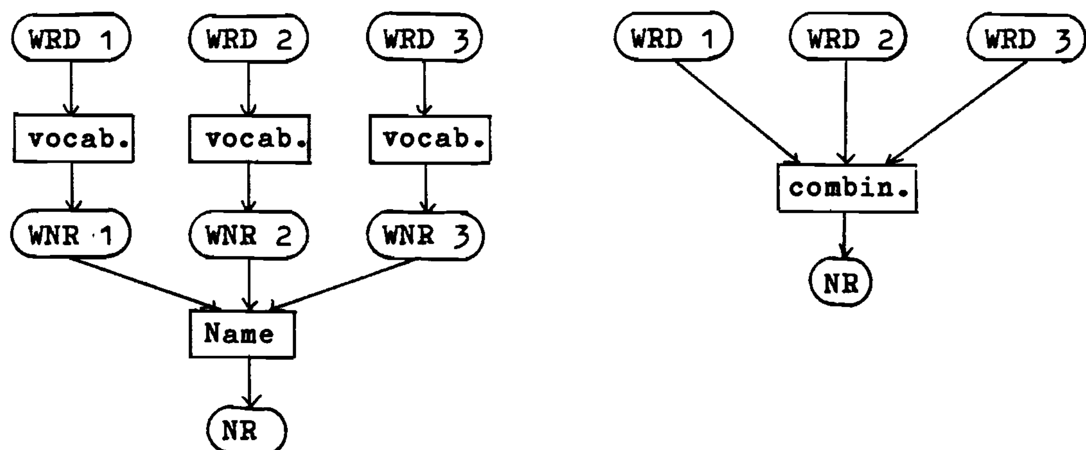
In order to recognize a given command, all permitted commands must have been stored in the memory of the interpreter, or a set of basic elements must be present together with the rules with which the given command can be processed. Both ways of storage of commands are used for the recognition of commands given to the supervisor. In order to recognize a special character, the character in the inputline is compared with the permitted special symbols which all are stored.

An integer number consists of one or more digits, ranging from 0 up to 9. When the character in the inputline is a digit, it will be processed together with the following characters which are digits; the result is a number that fits in one machine word.

The recognition and storage of names is of the second type: a set of basic elements and a number of processing rules. All permitted names are stored as a whole, but the words and name in the inputline may be abbreviated, so that the inputname is not similar to the stored name.
A name consists of one or more words; the recognition of the name can take place in two different ways:

1. separate recognition, in which first the individual words are looked up in a vocabulary and are provided with their wordnumber, and in which next the thus obtained wordnumber sequence is compared with the permitted sequences in order to obtain the number corresponding with the name;
2. combined recognition of words and name, in which words are not compared with the whole vocabulary, but only with those words which are possible on that place of the name after recognizing the preceding words.

These two methods are represented in fig. 10.



separate recognition method combined recognition method

fig. 10: recognition structure.

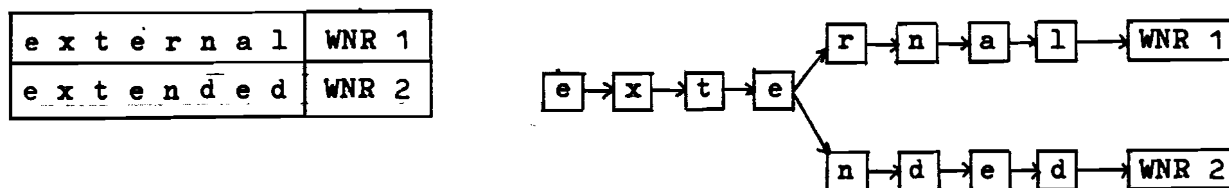
In order to make a comparison between the methods of recognition, first several storage structures for them will be discussed.

3.2.1. Separate recognition method

The separate recognition method converts the name in the inputline into a sequence of wordnumbers. By doing so, words and names get similar structures, because a word can be considered a sequence of letters. Therefore similar structures can be examined for the storage of both words and names and their corresponding numbers. The words and names are the keys to the wordnumber resp. name-number. The following structures can be used to store a sequence:

a - every sequence is individually stored as a whole;
b - the sequences are stored in a tree structure, in which every node of the tree contains one or more constituing elements of the sequence (i.e. letters or wordnumbers).

In fig. 11 an example of these structures is given for the storage of words, in which a node of the tree contains one letter. The words 'external' and 'extended' are the keys to the wordnumbers WNR 1 resp. WNR 2.



storage as a whole   storage in a tree

fig. 11: storage structures for words and wordnumbers.

A fast access to the stored wordnumber, when having an inputword, can be obtained by means of 'direct access' (see Lunbeck, Litt. 7). For this, it is necessary that the record of data to which access must be obtained, has a fixed length. When this is not the case, an index table must be used which points to the locations where the actual records are stored. Between the key K of the record and the address A in the table, a relation A = f (K) exists. Three types of relations can be distinguished:

- direct relation; e.g. A = K, A = K-64. Hereby it is desirable that the keys do have consecutive values, so that as few empty places as possible occur in the table. Consecutive key values are for example A ... Z, or 51 ... 100, when all intermediate letters or numbers occur.

When the keys are not consecutive one has to use:

- table relation: the index table is sorted according to the key values. This table must be searched, e.g. by binary search, to find the correct address;

- functional relation: the address is calculated from the key value, e.g. A = (K mod. 32) + 1. It is possible that several records get the same address; these records are called synonyms. For instance, the key values 17, 49, and 241 result in the same address when the relation A = (K mod. 32) + 1 is used. Several methods exist to handle this overflow (Litt. 7 and Litt. 8).

A key conversion is applied if the distribution of the keys does not satisfy, e.g. when there are too many synonyms when using the

functional relation. However, using key conversion with the functional relation method can solve the problems on one place, but may cause problems on another place.

The direct relation method is only useful to obtain access to a tree, in which the first letter or wordnumber is used for the key. For the wordnumber sequences it is also necessary that the numbers of the start words are consecutive, or can be made consecutive by key conversion.
The use of the first two letters of a word as the key to the table would result already in a table containing 676 addresses of which even half does not occur in practice.
Therefore, the table or functional relation must be used in all other cases than these particular storage in the tree.

The storage of words in a tree is useful when the vocabulary contains many words with identical characters in identical positions, because this will save memory space, in spite of the extra memory necessary for the pointers of the nodes.
Moreover, the recognition of a word will be faster, because that part of the word that has been recognized, will not be compared anymore with other words, if the comparison of the next letter fails.

The first version of the interpreter used the separate recognition of words and names, cf. par. 2.5. The words and wordnumber sequences were stored as a whole, and access was obtained by means of the functional relation.

3.2.2. Combined recognition method.

During the combined recognition both the word in the inputline and the next part of the name are recognized at the same time. This is achieved by comparing a word in the inputline with the words that are permitted on that position of the name, and by comparing the next word in the inputline only with those words that may follow

this recognized word. This will be clarified with an example.

EXAMPLE

It must be possible to recognize the names

- Nyquist diagram DSNR 1)
- discrete pole zero diagram (DSNR 2)
- discrete process parameters (DSNR 3)
- discrete signal values (DSNR 4)
- discrete signal parameters (DSNR 6)

Several words have synonyms: 'polar' for 'Nyquist', 'plot' for 'diagram', 'sampled' for 'discrete' and 'system' for 'process'. The possible word sequences for these names are represented in fig. 12, together with the corresponding namenumbers.

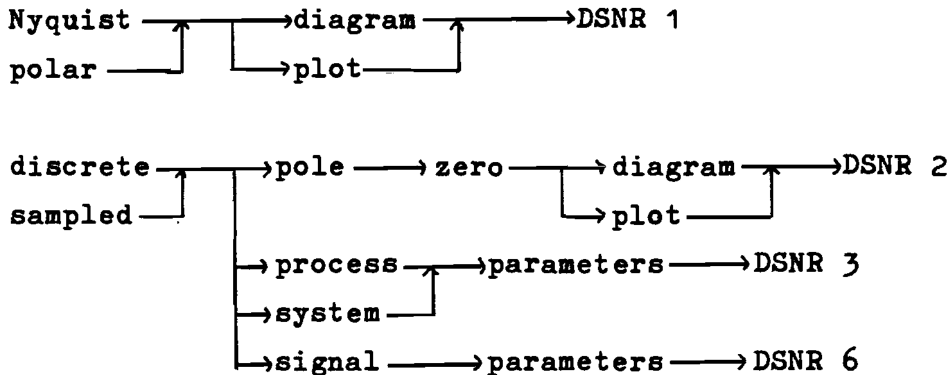


fig. 12: possible sequences for words.

The user has typed as his command: "sampled system parameters". First the word 'sampled' is looked up in the list of the permitted start words of the names. When this word has been found, the next word in the inputline,'system',is looked up only among the words that may follow 'sampled'. Finally 'parameters' is compared with the words that may follow 'system'. After recognizing this last word the namenumber DSNR 3 is obtained.

The relations described above can be placed in a tree-like structure, in which between two consecutive nodes also parallel branches are permitted. Because the words have a variable length and

because most words are used more than once, the nodes of the tree do not contain the words themselves, but a pointer to the word which is stored in a special word array.

3.3. Comparison of the separate and combined recognition.

The comparison of the recognition methods is based on the characteristics which are inherent to the structures. These characteristics are: speed of the algorithm, memory use of the data structure and the programs which use it, and convenience to generate and to use the data structure. Other characteristics for programs, like the suitability for maintenance and the comprehensibility of the program structure, are determined by the way the programs are implemented. The speed with which a name must be recognized, is related to the reaction time of the user. A processing time of 1 - 2 seconds is acceptable. The size of the data structure and the routines is not really limited, because overlay may be used when the available memory space proves to be too small. However, the use of overlay takes time, and this will decrease the speed.

The third characteristic of the structures is the convenience to generate and use the data structure. At the beginning of par. 3.2. already has been mentioned, that a set of basic elements and a number of processing rules must be given for the recognition of the names, because names and words may be abbreviated.
The set of basic elements consists of the complete names that may be typed, and of the numbers corresponding with these names. The processing rules specify what must be done in case a word or name has been abbreviated. An important aspect of the specification of the basic elements is the way the synonym words in the name are stored. A word A may be synonym to a word B and synonym to another word C, but the last two words, B and C, don't need to be synonyms of eachother. By the separate recognition method, the synonyms should be given the same word numbers if no conflicts arise. The number of possibilities to compose a name will increase considerably when all synonyms get different word numbers, if this is not strictly necessary. The necessity to give different word numbers

to synonyms must be examined prior to the generation of the data structure. By the combined recognition method, all words are stored which may occur on a certain place in the name; this also applies to the synonyms. In order to prevent excessive memory use, the synonyms are stored in parallel branches of the tree, which changes into a network structure.
The processing rules for the recognition must handle four variations of the input name: whole words in a whole name, whole words in an abbreviated name, abbreviated words in a whole name, and abbreviated words in an abbreviated name. Now will be examined how each of these combinations is processed during the separate recognition and during the combined recognition.

3.3.1. Separate recognition method.

When the words are stored in a tree, the recognition takes place node after node; if the letter in the next node is not similar to the next letter in the inputline, an alternative node will be taken to compare with, without needing to compare again the already recognized letters. In order to determine the word numbers of the words which fit an abbreviated word in the inputline, the subtree is searched which starts at the last recognized letter, cf. fig. 11.
When the words are stored as a whole, first the key to the stored word is calculated from the word in the inputline, and next the comparison of the words takes place. If the words are not similar, the next stored word is examined, which can be reached with the same key, and now the comparison of the word starts from the beginning. When the inputword has been abbreviated, all words that can be reached with the calculated key will be examined to fit the abbreviation, cf. fig. 11.
Mutatis mutandis the same considerations apply to the storage and recognition of the sequences of word numbers, in which a word number is stored in each node of the tree, resp. each word number sequence is stored as a whole.

With the above-mentioned considerations, at the same time two of the four variations of the inputname are discussed, viz. whole words in a whole name, and whole words in an abbreviated name. The result of the recognition of the words is the sequence of word numbers, which sequence might be abbreviated, and is looked up in the name-list to obtain the name number.

The word number sequence, resulting from the vocabulary when typing abbreviated words, does not need to be unique, because several words may fit an abbreviation, so that on some places in the sequence several word numbers are possible. It is not always necessary to examine all combinations that can be generated from these numbers to make up a valid name. If a certain combination does not exist because of a number in one of the first places, all combinations that can be generated with the word numbers of the following words don't need to be examined.
When using a tree for the storage of the number sequences, the set of word numbers resulting from the abbreviated inputword, is compared with the set of permitted next words. If two numbers are equal, on a stack is stored where the comparison of these two sets must continue after the fitting branch has been examined. A similar comparison is started with the sets of word numbers of the following words in this branch. The comparison of a branch is ended when the two sets of word numbers don't contain an equal number or when the inputname doesn't contain anymore word number sets. In the last case the tree is searched for name numbers, starting at the last recognized word number. When the comparison of a branch is ended, the last element of the stack is read, and the comparison continues with these sets. This is done until all possibilities are examined, i.e. until the stack is empty.

In fig. 12 an example is given of the words and names that may be stored, together with their respective numbers; the storage of the word number sequences both in a tree structure and in the storage structure as a whole. When the user types the name 'discrete signal parameters' in an extremely abbreviated form 'd.s.p.', the

resulting number sequence of the vocabulary is: (1, 2)', (3,2,4,5,6)', (7,8,6,9,10)'.
In the following part word numbers resulting from the vocabulary are indicated with an apostrophe, e.g. 4'
When comparing the number sequences, 1' is not a startnumber; so 2' is used.
The set (3,2,4,5,6)' is compared with the set (8,6,5,4); this fails on 8, but it fits on 6. Because the set (8,6,5,4) is not yet exhausted, on the stack is registered that the comparison must continue later with 5. Now the third set of the input, (7,8,6,9,10)', is compared with the set (10); this fits, and because there are no other numbers in this set, nothing is put on the stack. The input-string is empty further and the name number (2) is obtained in the tree.
Next the last element is taken from the stack and the comparison continues with the set (3,2,4,5,6)' and (.,.,5,4) in the same way. Doing so, a second namenumber, (5), is found. How these multiple results are handled will be discussed in par. 3.4. When the stack is empty, the comparison is finished and two namenumbers are found that fit on the user-command 'd.s.p.', viz. (2) and (5).

The way the word number sequences are stored as a whole, is depicted in fig. 12-d. When comparing the input set (1,2)', (3,2,4,5,6)', (7,8,6,9,10)' with the stored number sequences, first 1' is looked up to constitute a key number. It fails, and 2' is looked up, and fits. Next (3,2,4,5,6)' is compared with 8, and fails. The next number sequence that can be reached with the key 2' is searched, and comparison starts with the first word number. When no more sequences can be reached with the key 2' the comparison is completed and also now the two name numbers, (2) and (5), are found.

The processing of abbreviated words in an abbreviated name is almost identical to the above-described recognition of abbreviated words in a whole name. The input sequence will be exhausted earlier than the stored sequence, which means for the tree structure that the subtree must be searched for name numbers. For the storage as a whole, it is necessary to examine all word number sequences that can be reached with the key.

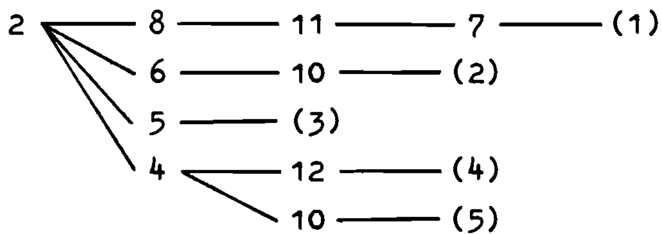
a : Words and word numbers:

diagram	:	1	simulation	:	5	parameter	:	9
discrete	:	2	system	:	6	parameters	:	10
samples	:	3	plot	:	7	zero	:	11
sampled	:	2	pole	:	8	values	:	12
signal	:	4	process	:	6			

b : Names and name numbers:

discrete pole zero plot	:	(1)
discrete process parameters	:	(2)
discrete simulation	:	(3)
discrete signal values	:	(4)
discrete signal parameters	:	(5)

c : Storage of number sequences in a tree:



d : Storage of number sequences as a whole:

2	8	11	7	(1)
2	6	10	(2)	
2	5	(3)		
2	4	12	(4)	
2	4	10	(5)	

fig. 12: example of storage of names.

3.3.2. Combined recognition method.

One of the four variations of the input name, whole words in a whole name, has been discussed already in par. 3.2.2.
The other three cases will be discussed here, using the names of fig. 12. The storage structure for the combined recognition method is given in fig. 13.

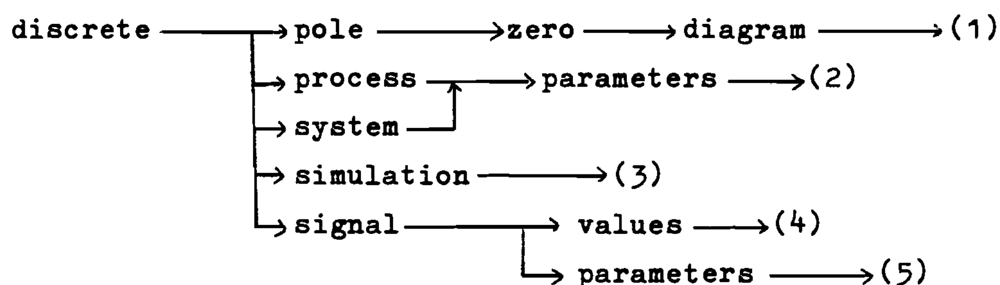


fig. 13: storage structure for combined recognition.

When the user has abbreviated the inputname, e.g. 'discrete signal', first 'discrete' is looked up, and then 'signal' is looked up in the list of words that may follow 'discrete'. Now the inputline is empty, so the subtree starting at 'signal' is searched for name numbers, and two different numbers will be found, (4) and (5).
When the user has typed the name 'd.s.p.', first the words starting with a 'd' are searched in the list of permitted startwords; this is only 'discrete'. Next 's' is compared with the list of words that may follow 'discrete'; it fits 'system', and on a stack is now registered that the comparison of 's' must continue later with the alternative word 'simulation'. Next 'p' is compared with the words that may follow 'system'; 'parameters' fits, and has no alternatives, so nothing is placed on the stack. The inputname and the stored name are exhausted and the name number (2) is found. Then the last element is taken from the stack, and 's' is compared with 'simulation'; it fits and on the stack is registered that the comparison must continue later with 'signal'. Because the inputline is not empty, while the stored name is ended, apparently this is the wrong name; so the last element is taken from the stack to continue

the comparison. Doing so, a second name number is obtained, viz. (5). Now the stack is empty, so that the recognizing of the inputline is completed.
The processing of the fourth variation of the inputname - abbreviated words in an abbreviated name - is a combination of the second and third type. When the inputline is further empty, the subtree, starting at the last recognized word, must be searched for name numbers.

With the above-mentioned, the description is completed of the ways the four variations of the inputname are processed during separate and combined recognition.

3.3.3. The selection of the storage structure for names.

The comparison of the two methods to recognize an inputname is based - as is stated at the beginning of par. 3.3. - on four characteristics: the memory use, the speed of the algorithm, the convenience to generate the data structure, and the convenience to use the data structure.
The memory use is divided into two parts: the memory used by the data structure, and the memory used by the routines which process the data structure. It is difficult to estimate the size of the routines; a detailed description of the individual routines is necessary to do so. The size of the data structures can be determined from the names that should be stored; these names are mentioned in Appendix B, where a survey is given of the present names. The memory requirements are calculated for a number of storage structures, cf. Appendix C. The results are:

- Separate recognition method

Words:	stored in a tree, one letter per node	± 1585*	words
	stored in a tree, two letters per node	± 1100*	words
	stored as a whole, overflow by 'hashing'	± 500	words
	stored as a whole, overflow in linked list	± 570	words

* Worst case situation, with no or as little similarities as possible.

Names:	stored in a tree, one number per node	± 710*
	stored as a whole, overflow by 'hashing'	± 390
	stored as a whole, overflow in linked list	± 450

- Combined recognition method: ± 1150 words.

The storage requirements for the separate recognition range from ± 900 up to ± 2300 memory words. The size of the structure for the combined recognition method differs only a little from the size of the smallest structure for the separate recognition method.
It is difficult to judge of the speed of the algorithms. Considering the number of actions that must be performed to recognize a name, it may be expected that both methods satisfy the requirements to respond within 1-2 seconds.
Concerning the convenience to generate the data structure, little can be said. The separate recognition method requires a preliminary search for conflicting synonyms. The combined recognition method stores all words on that place of the name where they occur, thus passing by this problem. The generation program will be mainly concerned with the test on erroneous input data and the display of suitable error-messages.
The combined recognition method is easier during use than the separate recognition method. The recognition of the words and the name takes place at the same time. The processing of the abbreviations is easier, because it is combined with the recognition of the words on their specific place in the name.
The two methods don't have really great advantages over eachother. Because of the ease to use the data structure, the combined recognition method is selected and will be used in the interpreter to recognize the names.

* Worst case situation, with no or as little similarities as possible.

3.4. Description of the data structure and programs.

3.4.1. The data structure.

The nodes of the trees, as given in fig. 11 and 13, are not practical because they contain elements of variable length, viz. words, and no, one or several pointers to the following words. The actual realization of the nodes is given in fig. 14. The nodes

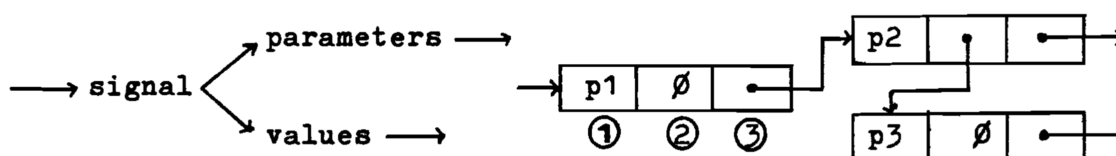


fig. 14: practical realization of the nodes.

are represented by a record which contains three pointers:

(1) a word-pointer, (2) an alternative-pointer and (3) a next-pointer.

The word-pointer points to the word that is stored in a word array, e.g. p1 points to signal, p2 points to parameters, and p3 points to values. The nodes of the words which may follow a certain word, are stored in a linked list of records, which can be traversed by using the alternative pointer. The alternative pointer of the last record of this list contains a special value, Ø.

The end of the name is indicated by a terminating record. This record consists also of three fields, just like the other records. The first field is Ø, and the next-pointer field contains the number of the name. The alternative pointer is used as normal.

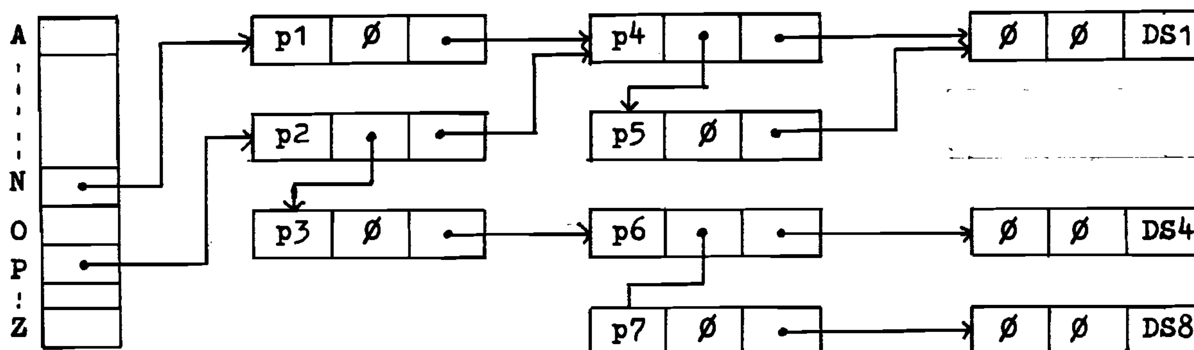
Many words are used as the first word of the names. For this reason, all start words of the names are not stored in one linked list, but a division is made in several lists, according to the first letter of the first word. Access to these lists is obtained via a special table, TRENTR.

The synonyms are stored in parallel branches. This is obtained by making the next-pointers of the synonyms point to the same record.

The structure of the tree / network is examplified in fig. 15. The following names are stored; in which a comma between words indicates that these words are synonyms:

- polar, Nyquist plot, diagram (DS1)
- process signals (DS4)
- process order (DS8)

TRENTR



p1: points to: Nyquist
p2: polar
p3: process

p4: points to: plot
p5: " " : diagram
p6: " " : signals
p7: " " : order

fig. 15: example of storage and overflow handling.

The access table, TRENTR (26), contains the pointers to the tree / network TREE (n,3) where the records are stored. The words are stored in the byte-array WRDARR (m) - this is not illustrated here - and are separated from eachother by a Ø-byte

3.4.2. The interpreter programs.

A strict separation is made between the interrogator and interpreter routines. The function INKB handles the interrogator actions, such as the execution of the special command to terminate the

session and the display of error-messages in case of an erroneous input. INKB calls also the interpreter DECODE.
The interpreter deals with four types of commands: names, numbers, empty line and special symbols, cf. par. 3.1. The special symbols are recognized by comparing the input character with the permitted special symbols. Numbers are processed in a routine which transforms the ASCII-character string of the digits into a number that fits a machine word.
The inputname may differ from the stored name as a result of the use of abbreviations. Several names may fit the inputname. When the inputname fits a stored name, either it has the same number of words as the stored name - the inputname is similar in length -, or it has less words - the inputname has been abbreviated -. In order to make a selection from the names found, a number of selection rules are laid down for the following two cases:

1. The inputname may be similar in length to one name, and may be as well an abbreviation of another name. The similar name will be the selected result. For instance, 'Bode diagram' (DS5) and 'Bode diagram generation' (OP5) are the stored names, and 'Bode diagr.' is the inputname: DS5 is selected.

2. Several names may fit the inputname, after applying the above-mentioned rule; the inputname is either similar in length to all these names, or is an abbreviation of all these names. One name is selected from these names if the following conditions both are satisfied:
   a - one of the names is a dataset name, and the other names are operation names;
   b - the dataset may be created by these operations.
   When the above conditions are satisfied, an operation name is selected, in case this is the only operation name in the list, and if this operation is the only one that may create the dataset; the dataset name is selected in the other cases.

The functions of the interrogator/interpreter routines and the way they pass their results, are discussed below.

FUNCTION INKB (IVAL)

This interrogator routine reads the inputline from the keyboard, and calls the interpreter DECODE. Depending on the result of DECODE, INKB passes this to its calling program, or gives an appropriate message, or calls the termination routine TERM.
The value of INKB and IVAL will be on return of this function:

INKB = 1, the inputline is empty
2, a dataset name has been typed, IVAL = number of dataset
3, a number has been typed; IVAL = number
4, a special symbol has been typed;
IVAL = number of symbol; 'ESC': 1, '?': 2
5, an operation number has been typed;
IVAL = number of operation.

The messages may be :

2 : type dataset name, or operation name or number
4 : name unknown, try again
352 : name too short, type a more specified name
353 : incorrect command, try again.

INTEGER FUNCTION DECODE (IRES)

This function is the interpreter routine; it examines the command line for a valid result. On return DECODE and IRES will be:

DECODE = 1, an empty command line has been typed
2, a special symbol has been typed; IRES = number of symbol
3, a number has been typed; IRES = value of number
4, a name has been typed; IRES = name number
5, an unknown name has been typed
6, several names fit the input name
7, illegal answer

This routine distinguishes the four types of command, and processes them separately. The words of the input and stored name are compared with eachother by the function CMPWRD. When the inputline is empty, all fitting numbers are searched by a subroutine GETNRS, which distinguishes between the names which are similar in length and which are abbreviated. After all possible names in the tree have been compared with the inputname, the function NAMENR examines the list of fitting names for a single result.

INTEGER FUNCTION CMPWRD (IPNT)

This function compares a word in the inputline, starting at location IP in that line, with the words of the linked list of alternative words, starting at record IPNT in the tree. On return, CMPWRD, IPNT and IP will be:

CMPWRD = 1, the word is not found in the linked list
2, the word in the inputline and a word of the list are totally similar; IP points to the last letter of the word in the inputline; IPNT points to the record of the word in the tree.
3, the word in the inputline is similar to a word of the list until the abbreviation point; IP points to this point in the inputline; IPNT points to the record of the tree, where the similarity occurs.

SUBROUTINE GETNRS (IPNT)

This routine searches for name numbers in the sub-tree, starting at record IPNT. The numbers are placed in a list NRS, applying the selection rule 1. When a name has been found which is similar in length to the inputname, only similar names will be placed in the list NRS.

FUNCTION NAMENR (IVAL)

This routine examines the list NRS of the name numbers which are found before by GETNRS. In case several numbers are present in the list, the routine applies selection rule 2. On return, NAMENR and IVAL will be:

NAMENR = 1, the list contains one number, or it contains several numbers, out of which one is selected; IVAL = name number.
2, the list is empty, no name has been found.
3, several names are possible, a unique answer cannot be selected.

3.4.3. Influences of the new interpreter on other Sater programs and data structures.

It is possible to insert the new interpreter in Sater with no other changes than the omission of the old interpreter routines and data structures. However, a small change has been introduced:
The tables which are used for the display of the names of the operations and datasets, contained word numbers; an index-table was used to find from these numbers the actual address of the words in the word array. This method has been abandoned, the tables now contain the actual addresses of the words. The display routine PRTEXT has been changed accordingly.
Another change has been introduced in the package, which is not related to the new interpreter; the number of operations on datasets which may be accommodated in the package has been increased.
A survey of all changes is given in Appendix G.

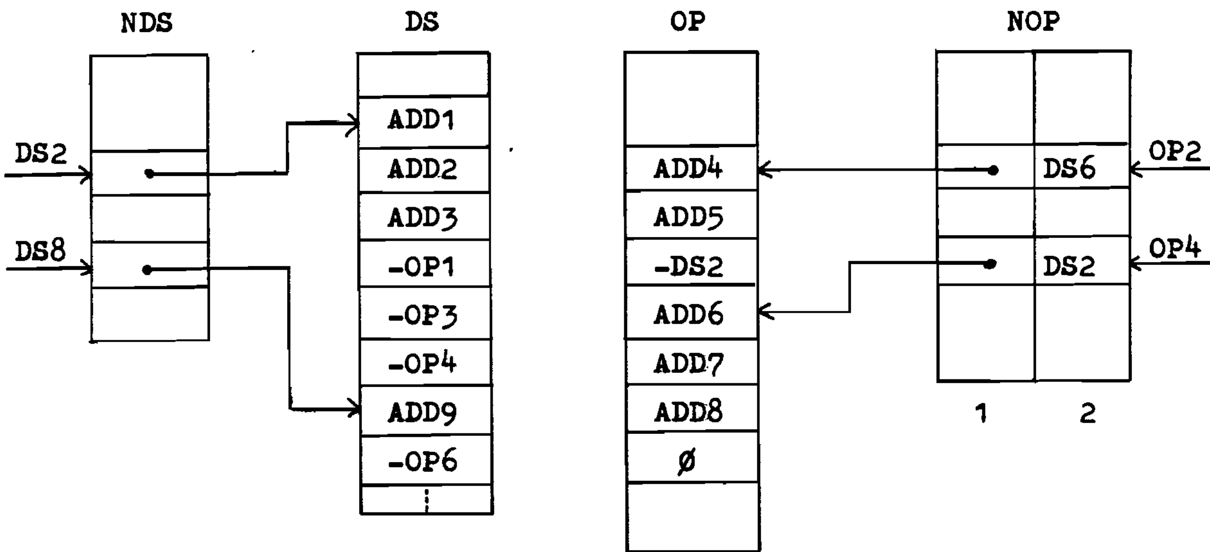
4. The table generation program TABGEN.

4.1. The structure of the tables.

The program TABGEN generates the file STABLSN.DAT, which is used by Sater. The file contains a number of tables and two scalars; they are subdivided into two sets:

- the tables which are used by the interpreter. These tables are TRENTR, TREE and WRDARR
- the tables which are used by the supervisor to display the names of the operations and datasets, and which describe the relations between the operations and datasets
- two scalars which specify the largest operation number, MOPNR, and largest dataset number, MDSNR; they are used by the supervisor and two copy programs to examine whether a number is not too high.

The tables for the interpreter are discussed in par. 3.4. The tables for the supervisor are NOP, OP, NDS and DS, cf. fig. 16. NOP and OP store the data concerning the operations; NDS and DS contain the data concerning the datasets. For each operation there is an entry in NOP, which contains in NOP (i,2) the number of the dataset, which is created by the operation, and in NOP (i,1) a pointer to the table OP. This table contains the name of the operation, which may be displayed, and the numbers of the datasets which serve as an input for the operation. The name consists of the addresses of its words in the word array WRDARR. The addresses are positive numbers, while the dataset numbers are negative. When an operation does not need an input dataset, a zero is placed in the table.
Each dataset has an entry to the table NDS. NDS contains the pointer to DS where the name of the dataset is stored and the numbers of the operations which may create it. The name is stored by means of the addresses of its words in WRDARR. The addresses are positive numbers, while the operation numbers are negative.



DSi : dataset number
OPi : operation number
ADDi : address in word array WRDARR

fig. 16: relation and name description tables.

4.2. Specification of the contents of the tables.

TABGEN generates the above-mentioned tables from the specifications as given by the system manager of Sater. These data must have been stored in the file SATNAMES.DAT prior to the execution of TABGEN; the formal of this file is discussed below. This file may be updated by using the text editor of the computer. Four types of data must be specified in the input file to be able to generate the tables:

- the largest operation number and the largest dataset number
- the relations between the operations and datasets
- the names which must be recognized, with their corresponding numbers
- the names of the operations and datasets, which are displayed.

The input file consists of four parts: one part for each type of data to be declared. Each part is terminated by a terminal line, which only contains a ' ' character. The text which is placed

after the fourth valid terminating line, is not processed. Additional information may be placed here, e.g. the date of specification. The syntax for the input is given in Appendix D. The semantics are discussed below, together with some restrictions and exceptions which only can be described in BNF by an explicit enumeration of all possibilities.
In the input operation n is represented by On, and dataset m is represented by Dm. The four parts of the input file are, in the order they must be specified:

1. Specification of the largest operation and dataset number.
   These data are specified by typing:
   D = 12
   O = 19
   This means that the highest dataset number is 12 and the highest operation number is 19. They must both be specified.
   An error-message will be given when they are not specified, or when they exceed the maximum as defined in TABGEN and Sater; the system maximum will be assumed for them. The maximum number of datasets is 20, and the maximum number of operations is 35.
   The data may be specified in an arbitrary order.

2. Specification of the relations between the operations and datasets.
   For each dataset must be specified which operations may create it, e.g.

   D2 : O2
   D3 : O3,O7,O10,O11,O12,O17

   Dataset 2 may be created by only one operation, 2; dataset 3 may be created by the operations 3, 7, 10, 11,12 and 17.
   For each operation must be specified which datasets serve as its input, and which dataset is created. When the operation does not need a dataset for its input, nothing is typed; e.g.
   O4 : D3, D6; D4
   O7 : ; D3
   O12: D4; D3

Operation 4 needs datasets 3 and 6 as its input and creates dataset 4. Operation 7 does not need an input dataset and creates dataset 3, while operation 12 needs dataset 4 as its input and produces dataset 3.
The specification of each dataset or operation consists of one line. Spaces and tabs may be inserted between the numbers and the separators: colon, semicolon and comma, e.g.: '07 :   D3'.

The specifications may be given in an arbitrary order.

3. Specification of the names, which must be recognized, and their corresponding numbers.
   At least one name must be specified for each operation and dataset; several different names may be specified for an operation or dataset. The sequence of the specifications is arbitrary.
   A word may have one or several synonyms; the synonyms are specified by inserting a comma between them.
   A name specification may contain at most 12 words without synonyms, and at most 20 words including synonyms.
   The words consist of letters only; they are separated from eachother by a space, tab or comma; a comma separates synonyms.Additional spaces or tabs may be inserted, but don't have a meaning.
   A name which contains no synonyms, consists of at most 68 characters, including spaces between the words.
   The specification of a name may be continued on a second line by typing a ' > ' ('greater then') character, followed by a 'CR', on the place where a space or tab may be typed.
   Some examples are:

```
O2 : sampled, discrete coefficient to sampled discrete pole,>
     p   zero,z diagram, plot
O11: tally, estimator, estimation
O11: prior knowledge fitting estimator, estimation
```

   The name of operation 2 contains several synonyms, and is too long for one line. Therefore ' > ' has been typed, and the specification continues on the next line.
   Several different names have been specified for operation 11.

4. Specification of the names, which must be displayed, and their corresponding numbers.
   For each operation and each dataset, a name must be specified by which Sater will mention the operation or dataset to the user. Only one name is permitted per operation or dataset. Synonyms are not allowed. A name may contain at most 12 words. The words may contain, in addition to letters, also the apostrophe, hypher, slash and dot (: which indicates an abbreviation); the permitted sequence is given by the syntax (see Appendix D). For instance: p.-z. is allowed and composes one word. The sequence of the specification is arbitrary.
   Some examples are:

   D 9: root-locus diagram
   012: parameter estimation i.v. method

   The user of Sater will type these names too; however, the characters like hyphen, slash or dot are treated by the interpreter in a different way than the words. These names must have been specified in part 3 in the following way:

   D 9: root-locus diagram
   012: parameter estimation instrumental variable method

The system manager of Sater might make two types of mistakes during the specifications of the numbers, relations and names:

1. the input data are syntactically incorrect,
2. the input data are not consistent, e.g. an operation number exceeds the maximum.

The generation program examines the input data for a correct syntax and for the following inconsistencies:

- an operation or dataset number exceeds its maximum value
- an operation appears on the list of operations which may create a certain dataset; however, the same dataset

is not mentioned as the output of that operation, or the operation is missing entirely.

- a dataset is specified to serve as an input for an operation, but no operations have been specified to create this dataset;
- a dataset is specified to be the result of a certain operation, but this operation does not appear on the list of operations which may create the dataset;
- an operation or dataset number has been specified during the relation description, but no name has been assigned to it during the specification for recognition or display, and vice versa.

The test on inconsistencies is performed after all specifications for a certain stage have been made, except for the tests for the maximum of the operation or dataset number. The occurrence of a syntax error in a line causes the data of the line not to be processed, c.q. not to be stored in the tables. This may result in inconsistency errors in a later stage.
In addition to the messages which are the result of erroneous input data, TABGEN may give a 'fatal error' message. This occurs when a table is full and new data must be added to this table; the exceeding of the limits is signalled. A fatal error occurs too, when a read error is made by the computer, or when less then four terminating lines have been typed.
TABGEN will give an appropriate error message and will stop immediately.
A survey of all error messages is given in Appendix E, together with an explanation of the cause, if necessary.

4.3. Temporary storage of data

TABGEN generates two temporary data structures during the generation of the tables for the output file. The first structure is the temporary storage of the relations between the operations and datasets; the second is an access structure to the word array WRDARR.
The relations between the operations and datasets, and the names by which the operations and datasets are displayed, are stored in the same tables DS and OP, but they are specified at different moments.

Therefore, a temporary storage of the relation description is necessary. The operations which may create a certain dataset, are stored in TNDS, cf. fig. 17; the number sequences are separated from eachother by a zero. The datasets which serve as an input for an operation are stored in TOP and the sequences of the different operations are separated from eachother by a zero. A zero is stored in the table TOP when the operation does not need an input dataset. The number of the dataset which is created by an operation, is stored immediately on its place in NOP.

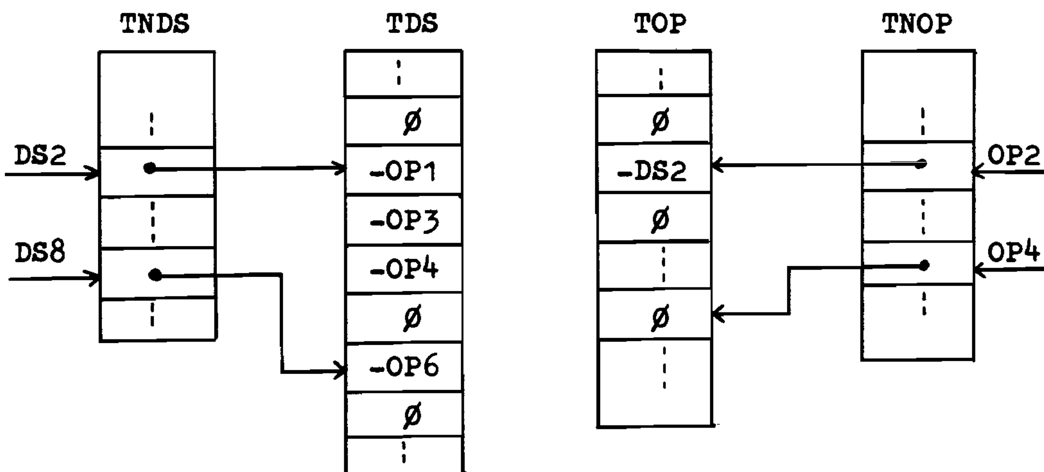


fig. 17: temporary storage of the relation tables.

During the processing of the names for recognition and display, it is very useful to have direct access to the word array WRDARR, when must be examined if a word is already present in the array.
For this purpose a data structure WRDPNT is used, in which the addresses of the words, which start with the same letter, are stored in a linked list, cf. fig. 18. Access to the linked lists is obtained via the table WPENTR, which contains not only an entry for the letters A ... Z, but also for the apostrophe. The last element of a linked list contains a zero instead of the next-pointer.

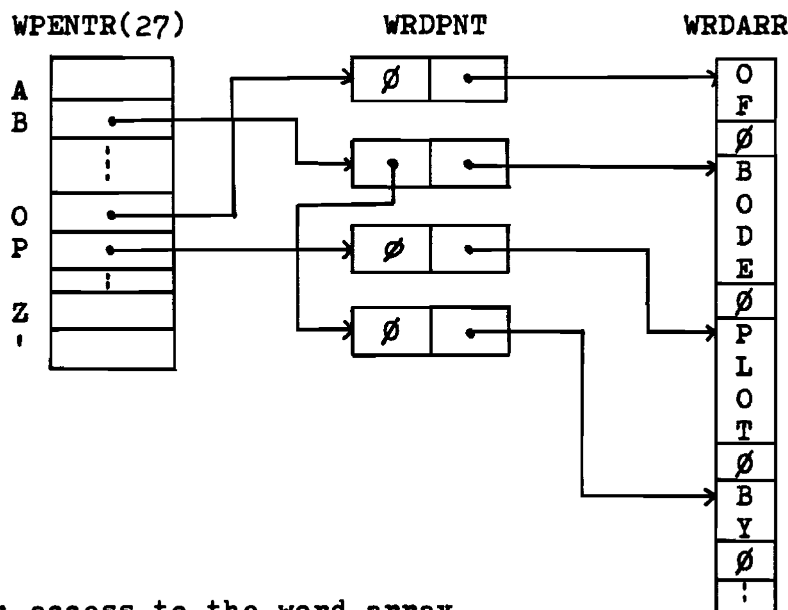


fig. 18: access to the word array.

4.4. The algorithm to build the tree

The output string of words has been converted by other routines into a list NAMARR (21,2), which contains the word addresses in WRDARR and the status of the words in the input string. The bound of one dimension is 21, because the specification of a name may consist of at most 20 words; in the array element after the last word, a special status word is put to indicate the end of the name. NAMARR is used to build the tree. The status of the word may be:

STATUS = 1: the word is the first word of a set of synonyms; the set may consist of only one word.

2: the word is one of the following words in the set of synonyms.

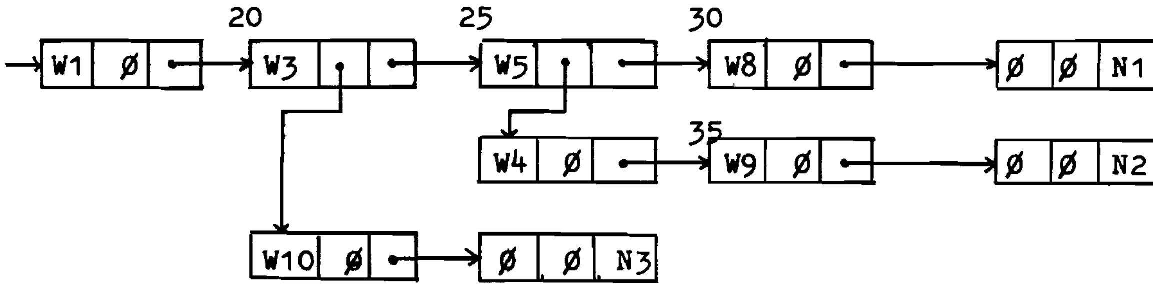
3: the input line is exhausted; this is placed after the last word of the name.

During the generation of the tree, only the addresses of the words are compared, added, or processed otherwise. However, in the following description we will often speak of 'words', instead of 'word addresses'. The explanation of the algorithm is easier to understand in this way.

The tree structure is implemented as an array of the elements. As new elements are defined, they will be stored sequentially in the array.

The words in the list NAMARR are processed set after set. The set of synonyms is compared with the words which are stored on that place in the name, and are stored in a linked list. When an input-word is not found in the linked list, it is added to the tree. This tree element must be provided with a pointer to the next element of the name; this is performed when all words of the set are processed, to which the newly attached element belongs. In order to know which elements must be provided with a next-pointer, the addresses of the new tree elements are stored in the list ADDLST (20). The length of this list is 20 elements, because at most 20 words may be mentioned in one name specification. When no word of a set was recognized, the following words of a name will not be present too, so they must also be added to the tree. Before the first word of the next set is added to the tree, the address of the next free element of the tree is assigned to the next-pointers of the elements, which are registered in ADDLST, because the next word will be stored there; the list ADDLST is cleared.
When a word is recognized in the linked list of stored words, the next-pointer of this recognized word is stored on the stack RECLST (20,2).
Some synonyms of a set may have been specified before to be synonym; these words will have the same next-pointer. This pointer will be stored on the stack only once. It may happen that words that are synonyms in the context of one name, were used separately in the context of other names. They have different next-pointers, and will be processed separately.
When all words of a set have been processed, and at least one word has been recognized, the following is done. The next-pointers of all elements which are registered in ADDLST, get the number of the element, where the first recognized word of the set points at, and the list ADDLST is cleared. All next-pointers in RECLST, which have been added in the last step, are provided with the location number in NAMARR where the next set starts, except for the last one.
The last next-pointer is taken from the stack and the comparison continues with the next set in NAMARR and the linked list, which starts at the specified next-pointer.

When the input of the name is completed, the corresponding name number is attached, if it is not yet present. Next is examined whether the stack RECLST is empty; if this is true, the processing of this name is finished. When RECLST is not empty, the comparison continues with the set and the linked list, which addresses are stored in the last element of the stack.
An example is given in fig. 19, where the initial tree (a) is given, and the sequence of word numbers (b) which compose the name, which has number N4. The contents of ADDLST and RECLST during the several stages of the addition are given in (c). The tree addresses are mentioned in the figure as normal numbers, e.g. 20, but are mentioned below as Ti, with i is a number, e.g. T20.
The first word W1 is recognized, and its next-pointer, T20, is placed on the stack RECLST, (1). The first set is exhausted, and the stack contains only one new next-pointer. In order to notice this, the algorithm does not only have a momentary stack-pointer for RECLST, but also a pointer which registers the value of the stack-pointer after processing the previous set. T20 is taken from the stack (2) and is used for the next comparison. The word W2 is not found in the linked list, and is added to the tree on location T100, which is the next free place in the tree at the bottom of the linked list. The address T100 is placed in ADDLST, (3). The synonym W3 is recognized, and its next-pointer, T25, is stored in RECLST,(4). The second set is now exhausted; the next-pointer of tree-element T100 is made T25, and ADDLST is cleared. RECLST contains only one new next-pointer; this one is taken from the stack (5), and is used for the next comparison. W4 is recognized, and its next-pointer T35 is placed on the stack (6). Also the synonym W5 is recognized, but it has a different next-pointer, so it is placed on the stack too, which contains two elements, T35 and T30, (7).
The set of synonyms is empty, so the next location number in NAMARR is added to the first element of the stack. The comparison continues with the last element, T30, (8). W6 and W7 are both added to the tree in location T101 and T102: this is registered in ADDLST, (9) and (10). This set is empty now; the whole name has been processed, so the name number is added to the tree on location T103, and the



a: initial tree structure

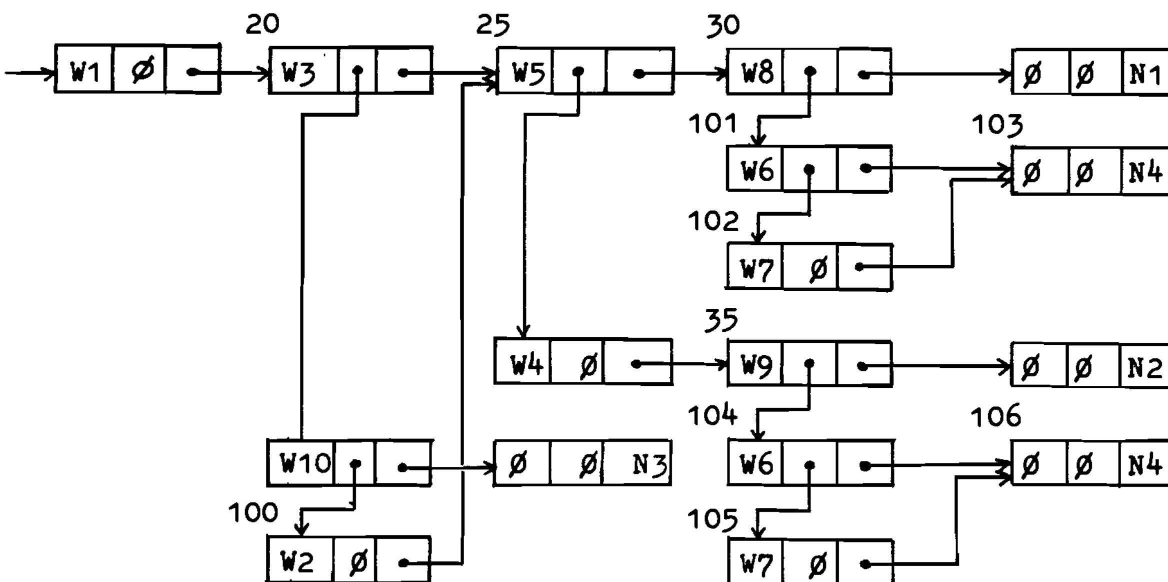
NAMARR

1	W1	1
2	W2	1
3	W3	2
4	W4	1
5	W5	2
6	W6	1
7	W7	2
8	-	3
.	-	-
.		
21	-	-

b: input list

phase	ADDLST	RECLST
1	-	T20
2	-	-
3	T100	-
4	T100	T25
5	-	-
6	-	T35
7	-	T35
		T30
8	-	T35, 6
9	T101	T35, 6
10	T101	T35, 6
	T102	
11	-	-
12	T104	-
13	T105	-
14	-	-

c: course of the build-up



d: final tree

fig. 19: insertion in the tree.

next-pointers of T101 and T102 are made T103. Next ADDLST is cleared, and the last element is taken from the stack (11). The comparison continues using the linked list, which starts at T35, and the set which starts at location 6 of NAMARR. After they have been processed, the stack RECLST is empty: the processing of this name is completed. The final structure of the tree is given in fig. 19-d.

4.5. The TABGEN programs

The program TABGEN consists of a main program, which declares the dimensions of the arrays and calls a number of subprograms. Four of these subprograms process each a part of the input file, corresponding with the type of input data - the maximum operation and dataset number, the relation description, the names for recognition, the names for display -. The fith subprogram performs the terminating actions, such as the creation of the output file, if no errors occurred. These subprograms call other subprograms; most subprograms process data from or store data in some tables, of which the dimension may be changed in the future, when they prove to be too small. When these adjustments must be made, the dimensions of the arrays need to be changed only in the main program.
The dimensions of the arrays are declared implicitly in the subprograms by the use of adjustable arrays (cf. Litt. 9, par. 2.6.6.). The adjustable arrays permit a subprogram to process arrays with different dimension bounds by specifying the bounds as well as the array name as subprogram arguments. In the description of the main program will be discussed how the changes must be introduced.
The description of the hierarchical structure of the subprogram calls is given in fig. 20. A subprogram is called by one or several other subprograms in the same block or in a higher block. A subprogram is called by only one program, when its name is shifted to the right over two places, compared with an above name of a subprogram. This last program is the only calling program, e.g. only SYNTST calls NXTTST.
The design of the four subprograms, which process the input data, is similar:

- an input line is read, and displayed on the user terminal
- the line is tested for its syntactical correctness
- the line is processed according to the function of its part of the specification; tests for inconsistency of the input are performed if this is possible for a single line
- when the terminating line is encountered, the inconsistency tests are performed which apply to this whole section of the specifications.

<table>
<tr><td colspan="5">MAIN</td></tr>
<tr><td rowspan="3">MAXNRS</td><td rowspan="2">RELAT</td><td>RECOGN
PRSENT
NRSRCH
BLDTRE
PSHREC
ADDWRD
ADDNR
WRDCMP</td><td>DISPL
NAMETR
NMBRTR</td><td rowspan="4">FINALE
NAMES</td></tr>
<tr><td colspan="2">TRANSL
SPEC
COMP
SYNTST
NXTTST
NRTST
LETTER
NEXTW</td></tr>
<tr><td colspan="3">STRTST
IODTST
IODBEP
NEXTCH</td></tr>
<tr><td colspan="4">LINE
ERROR
MESSAGE
SKSPAC</td></tr>
</table>

fig. 20: hierarchical structure of subprograms.

The discussion of the programs of TABGEN will not be as extensive as the discussion of the interpreter programs. Only the main program is discussed and the five subprograms which are called by this program. A survey of the files where all programs and subprograms are stored, is given in Appendix F. The program listings contain data on the function of the subprograms and on the way they present their results. These listings may be obtained from the system manager of the Sater package.

Main program TABGEN

This program opens and closes the input file SATNAMES.DAT, calls the subprograms MAXNRS, RELAT, RECOGN and DISPL, which process the input-file, and calls the subprogram FINALE, which gives some generation statistics and creates the output file when no errors have been detected.
TABGEN contains the declarations of the dimensions of the arrays: NDS, TNDS, NRSOP, NOP, TNOP, NRSOP, DS, TDS, OP, TOP, WRDARR, TREE and WRDPNT. The size of these arrays throughout all subprograms may be changed just by changing their sizes and the values of a number of variables in the main program. These variables get their values by a DATA-statement and are used, together with the array names, as subroutine arguments. The value of a variable must be equal to the dimension bound of its corresponding array. The corresponding variables and arrays are:

LIMDS : NDS, TNDS, NRSDS
LIMOP : NOP, TNOP, NRSOP
MTREE : TREE
WAMAX : WRDARR
WPMAX : WRDPNT

MDS : DS
MTDS : TDS
MOP : OP
MTOP : TOP

The change of these array sizes in TABGEN must be attended in general with the change of the array size in Sater. The change of MTDS, MTOP and WPMAX does not influence Sater, all others do.

SUBROUTINE MAXNRS

This program reads the maxima which are allowed for the operation and dataset number. The test for syntactical correctness of the input line, and the processing of the line are combined. When an error is detected, an appropriate message is given, and the system maximum is assumed to be the user specified maximum.

SUBROUTINE RELAT

This program processes the relation description. It reads an input line, tests for syntax errors, and stores the data in NOP and in the temporary tables TDS and TOP, after it has examined whether the operation and dataset numbers do not exceed the permitted maximum, and whether the operation or dataset has not been specified before. When the terminating line has been encountered, the program tests for the following inconsistencies:

- an operation is mentioned in the list of operations which may create a dataset: however, no input and output datasets have been specified for this operation, or the specified output dataset is a different one (error 28);
- a dataset is mentioned as input for an operation; but it has not been specified which operation may create this dataset (error 29);
- a dataset is mentioned to be the ouput dataset of an operation; this operation, however, is not mentioned in the list of operations which may create this dataset (error 30).

SUBROUTINE RECOGN.

This program processes the specification of the names for recognition. It reads an input line by means of LINE and performs the syntaxtest by means of SYNTST. This subprogram may encounter a valid continue character ' >' on the line. RECOGN will read the second line of the name and performs the syntaxtest. These two lines compose the specification of one name, and are processed together. The subprogram NRTST examines whether the name has been declared during the relation description; when this is not true, NRTST gives an error message, and RECOGN reads the next line. PRSENT examines whether the input name is already present in the recognition tree; when the name is present, but it belongs to a different operation or dataset, an error message is displayed by it. PRSENT puts the word array addresses of the recognized words in the array NAMARR.

The subprogram TRANSL finds the other words of the input name in the word array and puts the word addresses in NAMARR; when a word is not yet present, TRANSL adds it. BLDTRE adds the resulting number sequence to the recognition tree, according to the algorithm of par. 4.4.
RECOGN registers that the operation or dataset has got a name and reads the next line.
When a terminating line is encountered, RECOGN examines whether all operations and datasets, which were specified during the relation description, have got a name for recognition.

SUBROUTINE DISPL.

This program processes the specifications of the names for display. It reads the input line by LINE and performs the syntax test by means of SYNTST. NRTST tests if the dataset or operation has been specified before during the declaration of the relations, and if a name has not been specified before for this operation or dataset. The words of the name are found in the word array by TRANSL, and are added to the word array, when they are not yet present. The addresses of the words of the input name are stored in NAMARR. The resulting sequence of word addresses is stored in the array OP or DS by NAMETR, depending on the fact whether it is an operation name or a dataset name. NMBRTR adds the corresponding sequence of dataset or operation numbers, as was specified during the relation description to OP or DS. Finally, DISPL examines whether all operations and datasets, which were specified during the relation description, have got a name for display.

SUBROUTINE FINALE

This program performs the final actions of the generating program. It creates the output file STABLSN.DAT, when no errors were detected (IFC = Ø). A survey is given of the number of locations which were used in each of the tables: the generation statistics.

When no errors were detected, a survey may be given of all names which are stored in the recognition tree. The subprogram NAMES lists this survey on the console terminal or on the line printer.

5. Een speciale taal voor de regeltechniek.

5.1. Motivatie en doel.

Het ontwerp van het Sater-pakket, dat in de vorige paragrafen ter sprake kwam, was gebaseerd op het idee om een bibliotheek van programma's samen te stellen, om daarmee problemen uit de regeltechniek op te lossen. Ieder programma verricht er een specifieke operatie, zoals simulatie of parameter schatting. Een beperking van Sater is, dat het geen hiërarchische structuur van de operaties kent, waardoor meerdere operaties samengevoegd kunnen worden tot een superoperatie die met één aanroep meerdere operaties uitvoert, b.v. die eerst de proces parameters schat en vervolgens het polen-nulpunten plaatje weergeeft. Sater kan ook geen operaties gelijktijdig uitvoeren, zoals de ingangs- en uitgangssignalen van een proces inlezen en tegelijkertijd ook de proces parameters schatten. Een derde beperking is dat het pakket in Fortran geschreven is; het is noodzakelijk alle algorithmen geheel te herschrijven wanneer het pakket geïnstalleerd zou moeten worden op een computer, die alleen Algol of Pascal kan verwerken.

Een andere benadering van het oplossen van regeltechnische problemen is het gebruik van een speciale taal. Deze taal moet regeltechnische begrippen en bewerkingen bevatten, in plaats van mathematische, zoals b.v. bij Fortran en Algol. Het primaire doel van een programmeer-taal is om het de programmeur mogelijk te maken om zijn gedachten te formuleren in termen van abstractie, die geschikt zijn voor zijn probleem, in plaats van in termen van de faciliteiten die hem geboden worden door de hardware (Wirth, litt. 10).

Door het gebruik van een speciale taal wordt de oplossing van het probleem in twee stukken verdeeld, vgl. fig. 21:

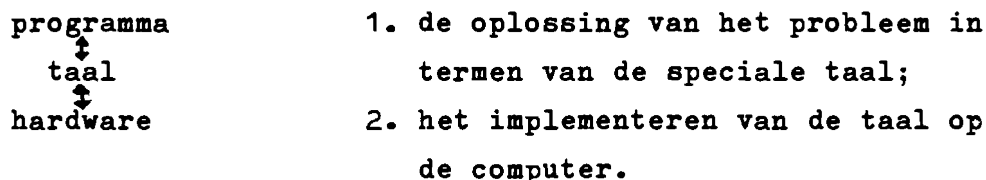


fig. 21.

Dit brengt als extra voordeel met zich mee, dat de overdraagzaamheid van programma's wordt vergroot. Wanneer een oplossing voor een bepaald probleem gevonden is op een computer, dan kan deze oplossing ook op een andere computer gebruikt worden. Alleen de implementatie van de taal moet aangepast worden; het programma blijft ongewijzigd.
De taal hoeft niet een geheel nieuwe taal te zijn, die een eigen compiler nodig heeft. De taal kan ook een toevoegsel zijn voor een andere procedure taal, b.v. Fortran of Pascal. Dit voorkomt de noodzaak een nieuwe compiler te ontwerpen en te realiseren, omdat van bestaande, geteste programmatuur uitgegaan wordt.
De taal moet zodanige elementen en operaties bevatten, dat deze geschikt is voor het oplossen van de veelsoortige problemen uit de regeltechniek. Hij moet o.a. geschikt zijn voor:

- simulatie
- procesbesturing
- parameter en toestandsschatting
- optimalisatie van regelaars
- de bepaling van de overdrachtsfunctie van samengestelde systemen, b.v. teruggekoppelde systemen
- kwaliteitsbeschouwingen, wat betreft b.v. stabiliteit en regelbaarheid.

Een algemeen streven bij het ontwerpen van een taal is het beperken van het aantal elementen van de taal-typen en operaties tot een minimum. Dit geldt ook voor een speciale taal voor de regeltechniek, ondanks het feit dat vele toepassingsgebieden beschreven moeten worden.
De volgende methodiek werd gevolgd om te bepalen welke aspecten een rol spelen in de speciale regeltaal. Een bepaalde operatie in een deelgebied van de regeltechniek wordt als uitgangspunt genomen; de operatie wordt m.b.v. meer elementaire bewerkingen beschreven in de vorm van een programma. Nagegaan wordt of deze bewerkingen nog verder opgedeeld kunnen worden, of zij beperkingen opleggen aan de programmeur, of bepaalde gegevens nog ontbreken, en of de bewerkingen computer-onafhankelijk zijn.

5.2. Opzet van een programma.

De onderzoeksmethode wordt hieronder gedemonstreerd aan de hand van een voorbeeld. De verkregen resultaten hebben algemene geldigheid, en zijn deels afgeleid bij eerdere programma's.
Het doel van het programma is een simulatie uit te voeren van een integrator, die op zijn ingang de som van een sinus en een stapfunctie krijgt toegevoerd. De ingangssignalen en het uitgangssignaal moeten op de terminal worden weergegeven.

```
'var' S1,S2,S3,S4 : signaal,
      OV1 : overdrachtsfunctie;
S1 := tijdreeks (sinus, A=2, W=3),
S2 := Laplace (1/s),
OV1 := Laplace (1/s);
S3 := S1 + 3 x S2,
S4 := output (OV1, S3),
'display' (S1,S2,S4).
```

Er is uitgegaan van slechts twee typen: signalen en overdrachtsfuncties; over de inwendige representatie hiervan wordt niets vastgelegd; in paragraaf 5.3 komt dit verder ter sprake.
De 'var' statement voorziet de variabelen van een type; hierdoor kan de compiler nagaan of de operaties op de gegevens geoorloofd zijn. Deze type toekenning zal ook voor in- en uitvoer variabelen moeten gelden opdat ook hun geldigheid onderzocht kan worden.
Het programma kent verschillende scheidingstekens voor de statements, welke tekens een verschillende betekenis hebben voor de uitvoeringsvolgorde van de statements. De komma duidt erop dat de bewerkingen gelijktijdig uitgevoerd mogen worden; de punt-komma geeft aan dat de voorgaande bewerkingen alle uitgevoerd moeten zijn, voordat de volgende bewerking uitgevoerd mag worden. Door dit aldus aan te geven wordt parallele processing mogelijk gemaakt; wanneer de computer dit niet toestaat, beschouwt de compiler de komma als een punt-komma.
Voor de specificatie van de constante signalen S1 en S2 moet een begintijd en een eindtijd opgegeven worden. S1 en S2 zijn 'constanten' omdat van tevoren bepaald wordt welke hun waarden zijn.

Ofschoon niets is vastgelegd over de inwendige representatie van signalen en overdrachtsfuncties, wordt bij de opgave van de constante waarde wel een representatievorm gespecificeerd. De opgegeven waarde wordt met formatting getransformeerd naar de inwendige representatie.
De beschrijvingswijze moet ook geschikt zijn voor de verwerking van niet-lineaire overdrachtsfuncties.
Een speciale functie 'output' wordt gebruikt voor de bepaling van het uitgangssignaal van een overdrachtsfunctie. Deze functie heeft de overdrachtsfunctie en het ingangssignaal als ingangsvariabelen.
Deze notatie wordt toegepast in plaats van S4 := OV1 (S3), omdat door deze laatste notatie OV1 zowel een variabele als een operatie kan zijn; dit maakt het testen door de compiler moeilijker.
De specificatie van een overdrachtsfunctie hoeft niet expliciet te gebeuren, b.v. door het geven van het toestandsmodel; het moet mogelijk zijn de relatie tussen een aantal signalen op te geven, waardoor deze overdrachtsfunctie bepaald is, vgl. fig. 22, alwaar in Laplace-notatie geldt Y = H.E, X = B.Y en E = U-X. Voor lineaire systemen kan een vervangende overdrachtsfunctie van u naar y berekend worden. In niet-lineaire systemen gaat dit niet, en zal in het model zelf gerekend moeten worden om y te bepalen uit u.

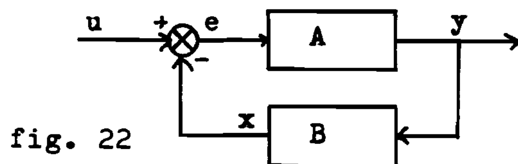


fig. 22

In het programma-voorbeeld is toch enige computer-afhankelijkheid geslopen door het begrip 'display'. Hierdoor kan het resultaat niet op b.v. een plotter gezet worden. Voor de variabelen S1, S2 en S4 is geen tijdsduur gespecificeerd, en ook zijn geen schalingsfactoren gegeven; die parameters zijn wel nodig voor de weergave.

5.3. Andere aspecten van de taal.

Zoals in de vorige paragraaf werd aangegeven, worden als nieuwe typen geintroduceerd: signalen en overdrachtssystemen, zonder dat hierbij melding gemaakt wordt van de representatievorm, b.v. discreet of continu. Ook wordt geen onderscheid gemaakt of de

systeembeschrijving geschiedt m.b.v. het toestandsmodel of b.v. de differentie-vergelijking. Dit onderscheid is met opzet niet gemaakt omdat de representatie-vormen in elkaar over te voeren zijn. De representatie-vormen die gebruikt worden, zijn een gevolg van de stand der techniek. Bij het oorspronkelijk berekenen met de hand, waren het polen-nulpunten beeld en het Bode- en Nyquist-diagram geschikte representatie-vormen. Door de opkomst van de computer vindt een verschuiving plaats naar het discrete domein, met gebruik van het toestandsmodel. Een volgende stap kan zijn het gebruik van de inpuls responsie;door het beschikbaar komen van array-processoren kunnen de convolutie reeksen sneller verwerkt worden. Door het gebruik van de abstracte begrippen signaal en overdrachtssysteem, worden aan de inwendige representatie geen regels opgelegd, en wordt de implementatie van de begrippen vrijgelaten. Voordat een keuze gemaakt kan worden voor een bepaalde inwendige representatievorm, zal eerst onderzocht moeten worden welke representatie zich het beste leent voor de verschillende toepassingsgebieden.

Een ander aspect van de taal is reeds deels ter sprake gekomen in de beschouwing van de programma opzet. Dit aspect is de tijd; het komt op twee manieren naar voren:

1. Op macro-niveau: de tijdsschaal van de bewerkingen is verschillend, zij zijn te onderscheiden in:
   - eenmalige bewerkingen, b.v. de bepaling van de overdrachtsfunctie van een systeem, wanneer de overdrachtsfuncties van de deelsystemen gegeven zijn;
   - voortgaande bewerkingen, zoals simulatie of procesregeling; deze mogen ook op discrete tijdstippen plaatsvinden.
2. Op micro-niveau: bij de beschrijving van de uit te voeren operaties wordt een onderscheid gemaakt tussen bewerkingen die strict sequentieel uitgevoerd moeten worden, en bewerkingen die parallel uitgevoerd mogen worden.

De onderverdeling op micro-niveau vindt plaats i.v.m. het beschikbaar komen van computersystemen die parallele verwerking mogelijk

maken.
De aanwezigheid van de tijd maakt het nodig het programma onder te verdelen in blokken, omdat bepaalde programmadelen eenmalig zijn, en andere voortdurend. De inhoud van de blokken hangt af van de structuur die het programma krijgt. Hiervoor is aansluiting mogelijk bij de Continuous System Simulation Language (CSSL, Litt. 11), waarvan CSMP (Litt. 12) de bekendste uitvoering is. Bij deze simulatie-talen wordt de specificatie van het probleem onderverdeeld in vijf blokken:

1. de specificatie van het systeem;
2. de specificatie van het gedrag van de onafhankelijke variabele, en de opgave van het integratie algorithme an de stapgrootte bij discrete systemen;
3. het besturen van de simulatie zelf, b.v. wat betreft het beeindigen van de simulatie;
4. de specificatie voor het vastleggen van gegevens voor, tijdens en na de simulatie;
5. de specificatie voor het aanpassen van parameters t.b.v. een volgende run.

Op grond van de bovenstaande specificaties wordt een programma gerealiseerd dat bestaat uit drie gebieden:
- het begin gebied: dit omvat de bewerkingen die voor aanvang van de eigenlijke simulatie verricht moeten worden, zoals het bepalen van de overdrachtsfunctie en het initialiseren van de parameters;
- het dynamisch gebied: hier vindt de eigenlijke simulatie plaats; na iedere slag wordt gekeken of de run afgelopen is, en of gegevens naar buiten gevoerd moeten worden;
- het eind gebied: hier worden afsluitende bewerkingen uitgevoerd, zoals het genereren van statistische gegevens; na deze fase wordt overgegaan naar het begin gebied, alwaar nagegaan wordt of een volgende run gedraaid moet worden.

Een geheel andere indeling van het programma wordt verkregen wanneer aansluiting gezocht wordt bij real-time programmeer-talen, zoals Concurrent Pascal (Litt. 13). Hierin kunnen bewerkingen, die parallel mogen worden uitgevoerd, beschouwd worden als parallele

processen. Wanneer deze bewerkingen gegevens met elkaar moeten uitwisselen, kan dit geschieden via een buffer; de synchronisatie van de processen geschiedt hierbij door het gebruik van een 'monitor'. De gedwongen sequentiële bewerkingen worden ondergebracht in één proces. Synchronisatie met de buitenwereld kan verkregen worden door de interrupt door het operating system te laten vertalen naar een subroutine-aanroep in een monitor.

Het verder onderzoek naar een speciale taal voor de regeltechniek zal zich moeten richten op drie gebieden:

1. het vastleggen van de elementen van de taal: de typen, operaties en structureringsregels;
2. het bepalen van een representatievorm voor de overdrachtsfuncties, die geschikt is voor alle toepassingsgebieden in de regeltechniek;
3. het bepalen van een implementatievorm van de taal, die zowel gedwongen sequentiële als mogelijk parallele uitvoering van programma's ondersteunt.

6. Conclusions and recommendations.

A new interpreter has been designed and implemented in Sater. It uses a single step method to recognize the names; this method permits an easy detection of ambiguities in the input name.
The design of the program, which generates the necessary table, is such that the sizes of the table may be adjusted in an easy way. However, the change of the table sizes in the generation program will almost always result in changes of the table sizes in Sater.
The interpreter connects names with operation and dataset numbers. This may be extended by connecting names with message numbers too. This allows the supervisor to display a message when the user has typed a name which for instance is ambiguous because it must be preceded by 'continuous' or 'discrete'. The supervisor may give a message like:: the name must be preceded by 'continuous' or 'discrete'.
The number of application programs, which may be inserted in the package, has been increased from 20 up to 35. The maximum number of datasets, which is permitted, has not been changed and remains 20. The figures are not fixed but may be changed in the future by adjusting all corresponding tables.

The application programs, which have been inserted in the package, do not all satisfy the rules which have been laid down for the relations between the operations and datasets. One departure originates from the former limitation that the package could only contain 20 application programs. The addition of new operations would exceed this number. Therefore, two application programs contain two operations each. Both operations create the same dataset, but one operation uses manual input, while the other calculates the dataset from another dataset. Because of the manual

input, no dataset has been specified in the relation tables to be the input dataset of this application program. The supervisor does not test for the availability of an input dataset even when the calculation operation will be performed. The extension of the maximum number of application programs offers the opportunity to divide these application programs into two different operations. Each operation will satisfy the rules.
A second departure of the relation rules results from a limitation which is imposed by the structure of the supervisor. Only one copy of a dataset may be present at a time. When a new copy is created, the old one is deleted automatically, even when it was created by a different operation. In some cases, this is very inconvenient. Therefore, the availability of two different copies has been introduced for the dataset 'discrete process parameters'. One copy may be created from the manual input; the other one may be created by a parameter estimation program. The application program asks the user of Sater to specify which of the two datasets must be used for the calculations. Therefore, the supervisor cannot test for the availability of a dataset before calling the application program.
This problem is solved by a drastic change of the supervisor structure. The concept of the availability of at most one copy of a dataset must be abandoned. The concept of variables of a certain type and operations on these variables must be introduced. The types are the present datasets. The names of the variables are are assigned by the user; the type of a variable may be derived from the operation which assigned a value to it, because an operation always creates a dataset of the same type. The supervisor keeps a record of the names and types of the datasets which have been created. The user specifies to the supervisor which operation must be performed and which are the names of the input and output datasets. The supervisor examines whether the necessary datasets

are present; if so, it calls the application program and passes the names of the input and output datasets to it.

The above-mentioned remarks all concern the existing program package Sater. A preliminary study has been performed for a new design of a program package. This resulted in the concept of a special language for control theory. Several aspects have been discussed. The research on this subject should be continued in three fields:

1. the defenition of all elements of the language: the types, the operations and the structuring rules,
2. the determination of one internal representation of the transfer function, which can be applied in all fields of control theory,
3. the implementation of the language elements, which allow sequential and parallel processing.

Survey of the symbolic names

For the names is also mentioned at which page they are defined or redefined.

name	page	meaning
ADDLST	57	list for treegeneration
CMPWRD	47	interpreter routine: comares two words
DECODE	46	interpreter routine:decodes input line
DS	18,49	supervisor table: contains the dataset names and operation numbers
DSi	51	represents dataset i
DSP	17	table of the old interpreter
GETNRS	47	interpreter routine: searches the tree for name numbers
INKB	28,46	main interpreter routine
KW	17	table of the old interpreter
KWP	17	see KW
MDSNR	49	largest dataset number, which is permitted
MOPNR	49	largest operation number, which is permitted
NAMARR	56	array containing the word addresses; used for table generation
NAMENR	48	interpreter routine: tries to select one number from the list which is created by GETNRS
NDS	18,49	supervisor table; points to DS
NKW	17	table of the old interpreter

name	page	meaning
NOP	49	supervisor table; points to OP, and contains the numbers of the output datasets for the operations
Oi	51	represents operation i
OP	49	contains the names of the operations, and the numbers of the datasets which are needed for the input
RECLST	57	stack for the generation program
SATNAMES.DAT	50	input file of the table generation program TABGEN
STABLSN.DAT	49	output file of TABGEN
TABGEN	49	program which generates the tables for the supervisor
TNDS	55	temporary storage of the numbers of the operations which may generate a dataset
TNOP	55	temporary storage of the numbers of the datasets which serve as an input for an operation
TREE	44	storage structure for the recognition of the names
TRENTR	44	access table to TREE
WPENTR	55	access table to WRDARR during tree-generation
WRDARR	44	array where the words are stored

LITERATURE

1 A.J.W. van den Boom en W.J.M. Lemmens:
Sater, an interactive program package for education and research in parameter estimation-, control- and signal analysis techniques.
IFAC Symposium on Trends in Automatic Control Education, Barcelona 1977.

2 W.J.M. Lemmens:
Een overzicht van de besturingsprogrammatuur van het interactief programmapakket Sater.
Intern rapport, vakgroep ER, mei 1979.

3 A.J.W. van den Boom:
The interactive program package SATER for estimation and control in education and research.
Journal A, Vol. 20, no. 2, April 1979.

4 Braakman, van Bussel:
Intern rapport, subfaculteit Psychologie, K.H. Tilburg, jan. 1979.

5 R. Bollen:
Het interactieve programmapakket Sater met betrekking tot multi-variabele systemen en de identificatie daarvan volgens Guidorzi.
Afstudeerverslag vakgroep Systeem- en Regeltechniek, afd. Natuurkunde T.H.E., oktober 1980, NR-642 (1980-10-13) Bollen.

6 Digital Equipment Corporation:
RSX-11M Task Builder Reference Manual.

7 Prof.dr. R.J. Lunbeck en drs. F. Remmen:
Bestandsorganisatie.
Academic Service, Den Haag 1977.

8 N. Wirth:
Algorithms + Data Structures = Programs.
Prentice-Hall, Inc., Englewood Cliffs, N.J., 1976.

9 Digital Equipment Corporation:
PDP-11, Fortran, Language Reference Manual.

10 N. Wirth:
Programming Languages: what to demand and how to assess them.
Berichte des Institutes für Informatik; Eidgenössische Technische Hochschule Zürich, no. 17, March 1976.

11 The SCi Continuous System Simulation Language (CSSL)
Simulation, Vol. 9, number 6, dec. 1967, pp 281-303.

12 Frank H. Speckhart, Walter L. Green:
A guide to using CSMP.
Prentice-Hall, Englewood Cliffs, N.J., 1976.

13 P. Brinch Hansen:
The architecture of concurrent programs.
Prentice-Hall, 1977.

APPENDIX A.

Memory map of Sater.

A survey is given of the memory area's where the program segments will be loaded when a routine is called, which resides in this segment.
This survey already contains the new interpreter. The data structure is stored in segment SEGSUP; the programs are stored in SEGINK and SEGNEW.

```
SMAIN.TSK;1 OVERLAY DESCRIPTION:

BASE    TOP        LENGTH
----    ---        ------
000000  061747  061750  25576.      SMAIN
061750  104703  022734  09692.         SEGSUP
104704  112433  005530  02904.            SEGR14
104704  113757  007054  03628.            SEGR15
061750  062543  000574  00380.         S0101
062544  067133  004370  02296.            SM0101
062544  071057  006314  03276.            SM0102
071060  075117  004040  02080.               SM0103
071060  075757  004700  02496.               SM0104
061750  073773  012024  05140.         SEGSTA
061750  070003  006034  03100.         S0301
070004  074653  004650  02472.            SM0301
070004  073143  003140  01632.            SM0302
061750  065243  003274  01724.         S0401
065244  070067  002624  01428.            SEGHGR
065244  074757  007514  03916.            SEGHSI
061750  062203  000234  00156.         S0501
062204  066263  004060  02096.            SM0501
062204  071013  006610  03464.            SM0502
061750  074007  012040  05152.         SEGR78
061750  067627  005660  02992.         S1001
067630  074333  004504  02372.            SM1001
067630  075117  005270  02744.            SM1002
061750  067467  005520  02896.         S1101
067470  072067  002400  01280.            SM1101
067470  073677  004210  02184.            S1105
073700  102327  006430  03352.               SM1102
073700  101627  005730  03032.               SM1103
061750  072303  010334  04316.         S1201
072304  100317  006014  03084.            SM1201
072304  100677  006374  03324.            SM1202
```

```
061750  063757  002010  01032.              S1601
063760  076467  012510  05448.                  SM1601
063760  072767  007010  03592.                  SM1602
061750  100407  016440  07456.              S1701
100410  101733  001324  00724.                  SM1701
100410  120507  020100  08256.                  SM1702
120510  121167  000460  00304.                      SM1703
120510  121167  000460  00304.                      SM1704
120510  124463  003754  02028.                      SM1705
120510  122247  001540  00864.                      SM1706
120510  123473  002764  01524.                      SM1712
100410  100567  000160  00112.                  SM1707
100410  101407  001000  00512.                  SM1708
100410  101503  001074  00572.                  SM1709
101504  103477  001774  01020.                      SM1710
101504  106273  004570  02424.                      SM1711
101504  102213  000510  00328.                      SM1713
101504  102667  001164  00628.                      SM1714
061750  063067  001120  00592.              S1801
063070  073343  010254  04268.                  SM1801
063070  077453  014364  06388.                  SM1802
061750  071233  007264  03764.              SEGR19
------  ------  ------  ------  ------  ------  ------  ------
124464  140403  013720  06096.          SEGCHA
140404  144733  004330  02264.              SEGRES
140404  140667  000264  00180.              SEGBRK
140404  141203  000600  00384.              SEGCHI
140404  141567  001164  00628.              SEGSTR
140404  140767  000364  00244.              SSS70
140404  143007  002404  01284.              SEGDRW
143010  143617  000610  00392.                  SSS60
143010  143573  000564  00372.                  SSS66
143010  143077  000070  00056.                  SSS62
143100  143737  000640  00416.                      SSS63
143100  143713  000614  00396.                      SSS64
143100  145157  002060  01072.                      SSS65
140404  144673  004270  02232.              SEGTER
144674  147127  002234  01180.                  SEGINK
144674  150423  003530  01880.                  SEGFPN
------  ------  ------  ------  ------  ------  ------  ------
150424  150423  000000  00000.          DUMCOT
150424  153347  002724  01492.              PARACC
153350  166427  013060  05680.                  SEGP01
153350  161553  006204  03204.                  SEGP00
161554  163463  001710  00968.                      SEGP02
161554  167447  005674  03004.                      SEGP03
161554  170073  006320  03280.                      SEGP04
150424  162173  011550  04968.              SCPE01
150424  163457  013034  05660.              SCPE02
150424  153437  003014  01548.              SEGNEW
```

APPENDIX B.

Survey of the names in Sater.

The list below presents a survey of the names which must be recognized in Sater.

Synonym words are separated from eachother by a comma.

A '>' character means that the name specification continues on the next line.

Dii denotes dataset ii; Ojj denotes operation jj.

```
D1: NYQUIST DIAGRAM,PLOT,PLAATJE
D1: POLAR,POLAIR DIAGRAM,PLOT,PLAATJE
D2: SAMPLED,DISCRETE POLE,P ZERO,Z DIAGRAM,PLOT
D2: Z PLANE PLOT,DIAGRAM
D3: SAMPLED,DISCRETE PROCESS,SYSTEM PARAMETERS
D4: DISCRETE SIGNAL VALUES,SAMPLES
D4: INPUT,I OUTPUT,O VALUES,SAMPLES
D4: PROCESS,SYSTEM SIGNALS
D5: BODE DIAGRAM,PLOT
D6: DISCRETE SIGNAL PARAMETERS
D7: CONTINUOUS SYSTEM,PROCESS PARAMETERS
D8: PROCESS,SYSTEM ORDER
D9: ROOT LOCUS DIAGRAM,PLOT
D10: INTERNAL FILE BY MEANS OF THE READ OPERATION
D11: EXTERNAL FILE BY MEANS OF THE WRITE OPERATION
D12: CONTINUOUS POLE,P ZERO,Z PLOT,DIAGRAM

O1: NYQUIST,POLAR PLOT,DIAGRAM GENERATION
O1: NYQUIST GENERATION
O1: PROCESS,SYSTEM PARAMETERS TO NYQUIST DIAGRAM,PLOT
O2: SAMPLED,DISCRETE POLE,P ZERO,Z DIAGRAM,PLOT GENERATION
O2: SAMPLED,DISCRETE COEFFICIENT TO SAMPLED,DISCRETE POLE,P >
ZERO,Z DIAGRAM,PLOT
O3: PARAMETER ESTIMATION EXTENDED MATRIX METHOD
O3: EXTENDED MATRIX METHOD ESTIMATION,ESTIMATOR
O3: MATRIX ESTIMATOR,ESTIMATION
O3: LEAST SQUARES ESTIMATION,ESTIMATOR
O3: EMM,EMMETHOD,LS,LSQUARES ESTIMATOR,ESTIMATION
O4: DISCRETE,SAMPLED SIMULATION
O5: BODE DIAGRAM,PLOT GENERATION
O5: BODE GENERATION
O6: SAMPLING OPERATION
O7: SPECIFICATION DISCRETE,SAMPLED PROCESS,SYSTEM PARAMETERS
O7: DISCRETE,SAMPLED PROCESS,SYSTEM PARAMETERS SPECIFICATION
```

```
08: SPECIFICATION DISCRETE,SAMPLED SIGNAL PARAMETERS
08: DISCRETE,SAMPLED SIGNAL PARAMETERS SPECIFICATION
09: Z S CONVERSION
010: PARAMETER ESTIMATION GENERALISED LEAST SQUARES METHOD
010: GENERALISED LEAST SQUARES ESTIMATOR,ESTIMATION
010: GLS,GLSQUARES ESTIMATOR,ESTIMATION
011: PARAMETER ESTIMATION TALLY METHOD
011: TALLY ESTIMATOR,ESTIMATION
011: PRIOR KNOWLEDGE FITTING ESTIMATOR,ESTIMATION
011: PKF,PKFITTING ESTIMATOR,ESTIMATION
012: PARAMETER ESTIMATION INSTRUMENTAL VARIABLE METHOD
012: INSTRUMENTAL VARIABLE ESTIMATOR,ESTIMATION
012: IV,IVARIABLE ESTIMATOR,ESTIMATION
013: ORDER TEST
013: TEST OF ORDER
014: READ EXTERNAL FILE FROM PERIPHERAL DEVICE
015: WRITE INTERNAL FILE TO PERIPHERAL DEVICE
016: ROOT LOCUS CALCULATION
017: PARAMETER ESTIMATION MAXIMUM LIKELIHOOD METHOD
017: MAXIMUM LIKELIHOOD METHOD ESTIMATOR,ESTIMATION
017: MAXIMUM LIKELIHOOD ESTIMATOR,ESTIMATION
017: MLM,MLMETHOD ESTIMATOR,ESTIMATION
018: SPECIFICATION CONTINUOUS POLE,P ZERO,Z DIAGRAM,PLOT
018: CONTINUOUS POLE,P ZERO,Z DIAGRAM,PLOT SPECIFICATION
019: SPECIFICATION CONTINUOUS PROCESS,SYSTEM PARAMETERS
019: CONTINUOUS PROCESS,SYSTEM PARAMETERS SPECIFICATION
#
```

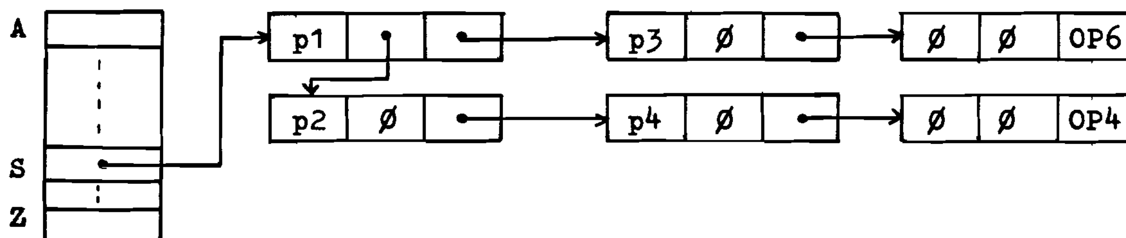
APPENDIC C.

The memory requirements for the combined and separate recognition method.

The memory requirements of a number of storage structures are calculated. The list of names of Appendix B is used as starting-point for the calculations; Appendix B gives a survey of the present names in Sater. This list contains 52 names with an average length of 4 words/name and it contains 80 words with an average length of 6½ letter/word.

The formula's, which are deducted for the storage of the names, assume that the average word length is 6½ letter, and that the average name length is 4 words. The total number of words which is stored, is denoted by n; the total number of names which is stored, is denoted by m.
Only one structure is examined and calculated for the combined recognition method, because the number of variations is limited.

I. Combined recognition method



p1 points to: sampling
p2 points to: sampled

p3 points to: operation
p4 points to: simulation

The words are stored in a byte-array, which contains one letter per byte; the words are separated from eachother by a Ø-byte. The memory requirements are (in memory words of 2 bytes):

$$26 + k \times 3 + (6\tfrac{1}{2} + 1) \times n/2$$

in which: k = number of tree elements. The last term is the byte-array.

About 275 tree elements are necessary to store the names of Appendix B. The actual requirements are:

$$26 + 275 \times 3 + (6\tfrac{1}{2} + 1) \times 80 / 2 = 1151.$$

II. <u>Separate recognition method</u>

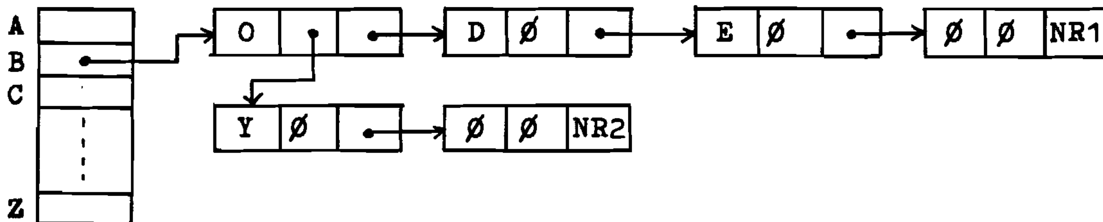
Six different structures are examined for the storage of words, and three structures for the storage of the word number sequences.

A. Storage structures of words.

When applying a tree structure, the average word length in a tree may decrease one, because this letter is stored in the access table; it may increase one because of the alignment of a word. The number of nodes per word increases one by the storage of the word number.

When applying the separate storage structure, the average word length is increased one, because of the Ø-byte between the words, which are stored in a byte-array with one letter per byte.

1. Tree, with one letter per node; access is obtained by the direction relation of the first letter.

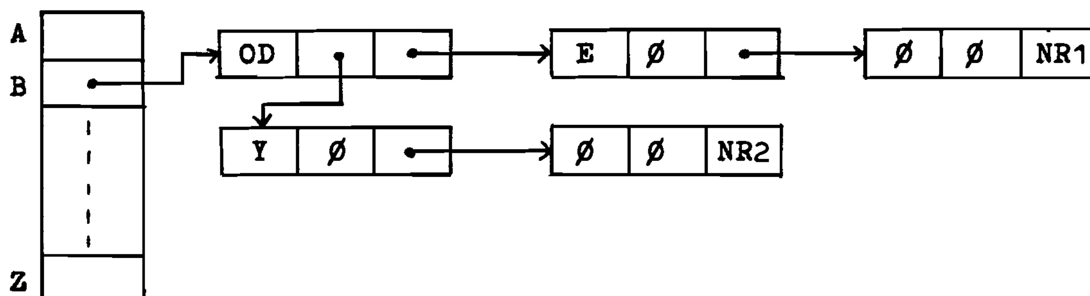


The words 'Bode' and 'by' are stored, with their respective word numbers NR1 and NR2. The memory requirements are:

$$26 + nx \quad (6\tfrac{1}{2}-1)+1 \quad x\ 3 = 26 + 19\tfrac{1}{2}\ x\ n$$

This is the 'worst case', when no letters are in common except for the first one. In practice, some letters will have been declared by other words, so the actual requirements are less.

2. Tree, with two letters per node; access is obtained by the functional relation method, using the first letter.

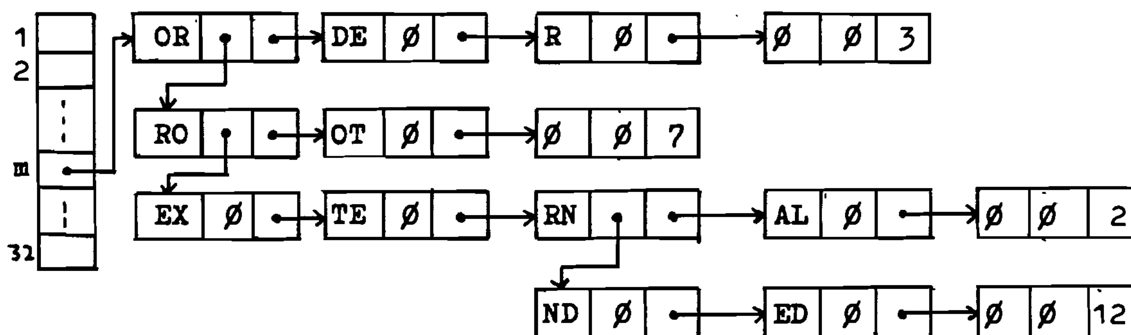


Also 'Bode' and 'by' are stored here.

The memory requirements are:

$26 + n \times (6\frac{1}{2}-1+1)/2 + 1 \times 3 = 26 + 12\frac{3}{4} \times n$ (worst case)

3. Tree, with two letters per node; access is obtained by the functional relation method, by which the words are projected on 32 places.

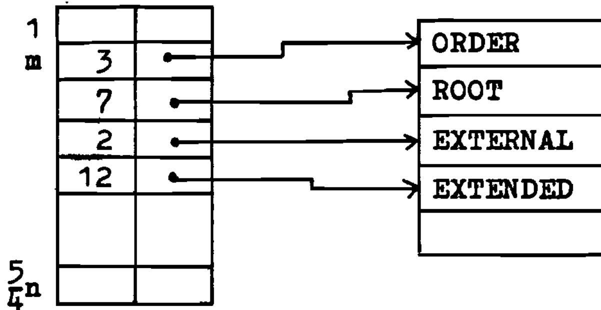


The words: 'order', 'root', 'external' and 'extended' are stored; they have identical keys.
The memory requirements are:

$32 + n \times (6\frac{1}{2}+1)/2 + 1 \times 3 = 32 + 14\frac{1}{4} \times n$ (worst case)

In practice, part of the duplets will be declared by more words.

4. The words are stored as a whole separately in a word array; access is obtained by a functional relation; overflow is handled in the access table.

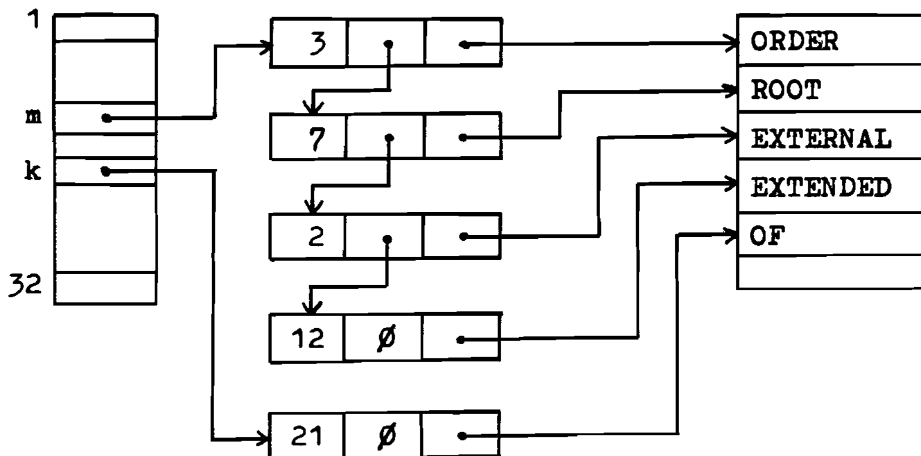


The first row contains the word numbers.
The memory requirements are:

$$\frac{5}{4} \times n \times 2 + n \times (6\tfrac{1}{2} + 1)/2 = 6\tfrac{1}{4} \times n$$

In the word array the words are stored with one letter per byte, and are separated from eachother by a zero-byte.

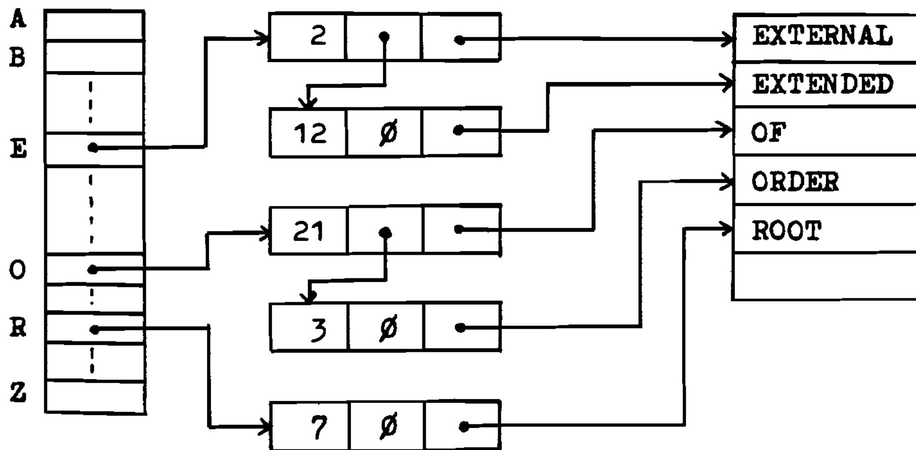
5. The words are stored completely; access is obtained by a functional relation; because of overflow handling, an intermediate step is used.



The memory requirements are:

$$32 + 3 \times n + n \times (6\tfrac{1}{2} + 1)/2 = 32 + 6\tfrac{3}{4} \times n$$

6. The words are stored completely; access is obtained by the functional relation method, using the first letter of the word, because of overflow handling an intermediate step is used.



The memory requirements are:

$$26 + 3 \times n + n \times (6\tfrac{1}{2} + 1/2) = 26 + 6\tfrac{3}{4}\ n$$

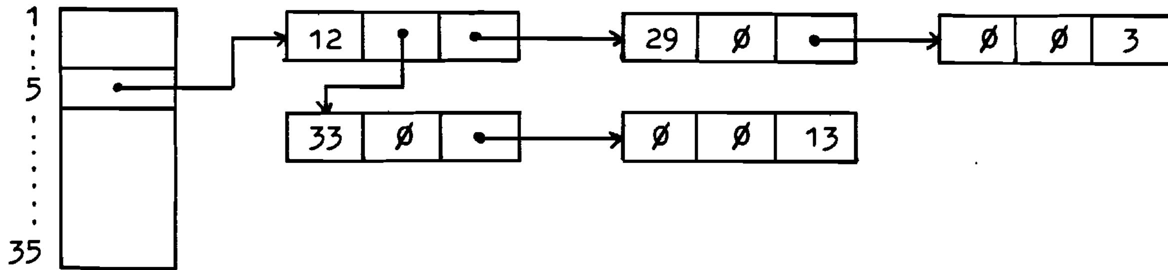
For these six structures the actual memory requirements are for a list of 80 words:

1. 26 + 19½ x 80 = 1586 (worst case)
2. 26 + 12¾ x 80 = 1046 (worst case)
3. 32 + 14¼ x 80 = 1172 (worst case)
4. 6¼ x 80 = 500
5. 32 + 6¾ x 80 = 572
6. 26 + 6¾ x 80 = 566

For 1. - 2. and 3. adjustments must be made because some letters and duplets are used in more words. For the list of Appendix B these are: 195 letters, resp. 24 duplets, resp. 120 duplets, so that the actual memory requirements for 1. - 2. and 3. will be: 1390, resp. 1022, resp. 1050; and for 2. and 3. possibly up to 120 less, depending on the adjustment.

B. Storage structures of the word number sequences.

1. The number sequences are stored in a tree. An optimal choice may be made for the word numbers of the first words by assigning to them consecutive numbers.

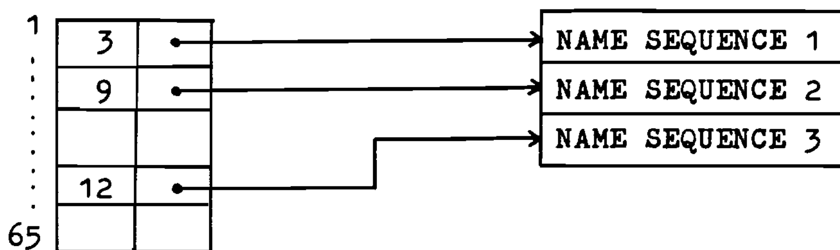


The memory requirements are:

$35 + (4 \times m - 35) \times 3 + m \times 3 = 15m - 70$

This is a 'worst case' situation when only the start words are equal.

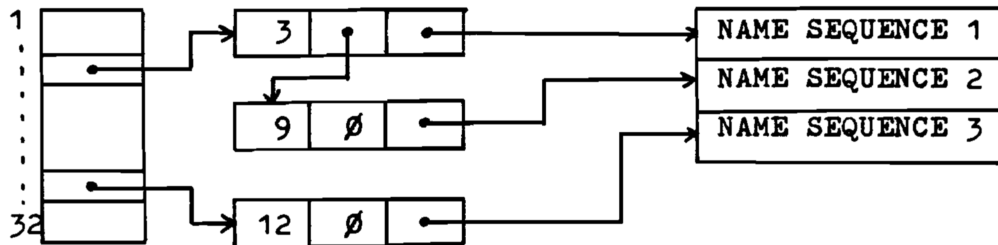
2. The sequences are stored as a whole, separated from eachother by a zero; the overflow is handled in the access table.



The first column contains the name number.
The memory requirements are:

$\frac{5}{4} \times 52 \times 2 \; + \; 4 \times m + m = 5m + 130$

3. The sequences are stored as a whole, separated from eachother by a zero; the overflow is handled in a special list.



The memory requirements are:

$32 + 3 \times m + 4 \times m + m = 8 \times m + 32$

The actual memory requirements for the storage of 52 names are in these structures:

1. $15 \times 52 - 70 = 710$ (worst case)
2. $5 \times 52 + 130 = 390$
3. $8 \times 52 + 32 = 448$

Appendix D

Syntax of the input file SATNAMES.DAT

The syntax of the input file SATNAMES.DAT for the generation program TABGEN is presented below. It consists of four parts, which are specific for each type of data, and a general part, which applies to all parts.

```
<file>      ::= <maxspec> <relspec> <recspec> <dispspec> <else>
<else>      ::= all sorts of characters
```

Specification of the maxima

```
<maxspec>  ::= <maxlines> <termline>
<maxlines> ::= {<dumline>} <maxDS> {<dumline>} <maxOP>
                {<dumline>} |
                {<dumline>} <maxOP> {<dumline>}
                <maxDS> {<dumline>}
<maxDS>    ::= {<separ>} D {<separ>} =
                {<separ>} <number> {<separ>} <CR>
<maxOP>    ::= {<separ>} O {<separ>} =
                {<separ>} <number> {<separ>} <CR>
```

Description of the relations

```
<relspec>  ::= <rellines> <termline>
<rellines> ::= {<dumline>} <relatlin> |
                <rellines> {<dumline>} <relatlin>
<relatlin> ::= <DSline> <OPline>
<DSline>   ::= <DSnr> : <OPNRS> <CR>
<OPline>   ::= <OPnr> : <DSin> ; <DSnr> <CR>
<OPNRS>    ::= <OPnr> | <OPNRS> , <OPnr>
<DSin>     ::= {<separ>} | <DSNRS>
<DSNRS>    ::= <DSnr> | <DSNRS> , <DSnr>
```

Specification of the names for recognition

```
<recspec>  ::= <reclines> <termline>
<reclines> ::= {<dumline>} <recoglin> |
                <reclines> {<dumline>} <recoglin>
<recoglin> ::= <ODnr> : <recname>  <CR>
<recname>  ::= <wordset> | <recname> <separ> <wordset>
<wordset>  ::= {<separ>} <recword> {<separ>} |
                <wordset> , {<separ>} <recword> {<separ>}
<recword>} ::= <letter> | <recword> <letter>
```

Specification of the names for display

```
<disspec>  ::= <dislines> <termline>
<dislines> ::= {<dumline>} <displine> |
                  <dislines> {<dumline>} <displine>
<displine> ::= <ODnr> : <dispname> {<separ>} <CR>
<dispname> ::= {<separ>} <dispword> |
                  <dispname> <separ> {<separ>} <dispword>
<dispword> ::= ' | <letter> | <combin> | <word> <letter> |
                  <word> <spec> | <word> <combin>
<combin>   ::= <letter> .
<spec>     ::= - | / | '
```

Generally

```
<ODnr>      ::= <DSnr> | <OPnr>
<DSnr>      ::= {<separ>} <Dii> {<separ>}
<OPnr>      ::= {<separ>} <Oii> {<separ>}
<Dii>       ::= D <number>
<Oii>       ::= O <number>
<number>    ::= <digit9> | <digit9> <digit>
<digit>     ::= 0 | 1 | ... | 9
<digit9>    ::= 1 | 2 | ... | 9
<letter>    ::= A | B | ... | Z
<termline>  ::= {<dumline>} <term>
<term>      ::= {<separ>} # {<separ>} <CR>
<dumline>   ::= <separ> {<separ>} <CR>
<separ>     ::= space | tab
<CR>        ::= carriage return
```

Additional rules are:

1. The input line may consist of maximum 72 characters.
2. During the specification of the names for recognition,it is permitted to substitute one <separ> by the continue character '>' followed by {<separ>} <CR>. The specification consists now of two lines. Rule 1 applies to each of both lines.
3. The names may consist of at most 12 words, without synonyms.
4. The names may consist of at most 20 words, including synonyms.

APPENDIX E.

Error-messages during the generation of the tables.

The program TABGEN gives an error-message when an error occurs during the generation of the tables. Normally, these messages consist only of an error-number, but some additional information is given in several cases. The kind of error may be such, that the program cannot be continued; the error is 'fatal', and the program is terminated prematurely.
The error numbers and their corresponding meaning are specified below; they are classified according to the stage of the specification.

Generally

1 the 'end-of-file' character was encountered during a read. This means that not all four parts of the specifications have been terminated with a valid terminating line: 'CR'.

2 error-during-read. This may be caused by a system error of the computer, or - what is more likely - the input file SATNAMES.DAT does not exist.

3 the limit of a table is exceeded. Additional information is given about the table, which caused the error: >>>>> ttt; MAX = nnnn <<<<<
ttt is the name of the table, and may be DS, TDS, OP, TOP, WA (= WRDARR), TRE (= TREE) and WP (= WRDPNT); nnnn is the limit of this table.

* The limits are defined by the sizes of the arrays. This error can be corrected by enlarging the array, in TABGEN and Sater.

4 an inputline consists of more than 72 characters; this line is skipped, because it cannot be processed correctly.

Specification of the maximum operation and dataset number

11 the specified maximum exceeds the system limit.
The additional message is:
>>>>> mmm; MAX = nn <<<<<
mmm may be DSN, which refers to the maximum dataset number, or OPN which refers to the maximum operation number; nn is its maximum.

12 no maximum has been specified for the operation and/or dataset numbers.

13 syntax error.

Specification of the relations

21 a dataset number is not permitted on this place.
22 an operation number is not permitted on this place.

23 syntactically incorrect number.

24 more data expected on this line.

25 illegal character.

26 an operation or dataset number exceeds its maximum

27 the relations for this dataset or operation have already been specified.

28 Additional information is: DATASET nn.
An operation is mentioned in the list of operations, which may create dataset nn; no input and output datasets have been specified for this operation, or the specified output dataset is not nn.

29 Additional information is: OPERATION mm.
A dataset is mentioned as input for operation mm; it is not specified which operation may create this dataset.

30 Additional information is: OPERATION mm.
A dataset is mentioned to be the output dataset of operation mm; operation mm is not mentioned in the list of operations which may create this dataset.

<u>Specification of the names for recognition and display</u>

31 illegal dataset or operation number.

32 illegal character.

33 more data are expected.

34 a comma is followed by a CR

35 a comma is followed by a comma.

36 the continue character '>' is not permitted on the second line of the specification.

37 the line is not empty after the continue character '>'.

38 a continue character '>' is not permitted during the display declaration.

39 synonyms are not permitted during the display declaration.

40 a name, without synonyms, may not consist of more than 12 words.

41 the name specification may not contain more than 20 words, including synonyms.

42 a name may not consist of more than 68 characters, including a space between the words.

43 Additional information is: DATASET: nn , or OPERATION: mm.
The specified name for this dataset or operation is identical to the name of dataset nn or operation mm.

44 the name of this operation or dataset has already been specified for display.

45 this operation or dataset has not been specified during the relation description.

46 Additional information is: OPERATION mm.
Operation mm is specified during the relation description, but a name has not been specified for recognition or display.

47 Additional information is: DATASET nn.
Dataset nn is specified during the relation description, but a name has not been specified for recognition or display.

APPENDIX F.

Survey of the files where the subprograms of TABGEN are stored.

ADDNR	: TABG17	NEXTW	: TABG15
ADDWRD	: TABG17	NMBRTR	: TABG14
BLDTRE	: TABG16	NRSRCH	: TABG13
COMP	: TABG10	NRTST	: TABG12
DISPL	: TABG4	NXTTST	: TABG8
ERROR	: TABG6	PRSENT	: TABG11
FINALE	: TABG5	PSHREC	: TABG17
IODBEP	: TABG7	RECOGN	: TABG3
IODTST	: TABG7	RELAT	: TABG2
LETTER	: TABG9	SUSPAC	: TABG7
LINE	: TABG6	SPEC	: TABG9
MAXNRS	: TABG1	STRTST	: TABG7
MESSAG	: TABG6	SYNTST	: TABG8
NAMES	: TABG18	* TABGEN	: TABGEN
NAMETR	: TABG14	TRANSL	: TABG10
NEXTCH	: TABG7	WRDCMP	: TABG17

* Main program

APPENDIX G

Changes, introduced in the Sater programs.

SSS01	SMAIN	The number of subprogram calls has been extended from 20 to 35
SSS29	SUPV	Common area's TBL and EKW have been changed: /TBL/ DS,NDS,LNDS,NOP,OP,LNOP /EKW/ KW . Two commons are added: /TREE/ TRENTR,TREE /TABDIM/ MAXDS,MAXOP . Added : DATA MAXDS,MAXOP ; this specifies the dimension bounds of NOP and NDS. READ statement has been changed accordingly.
SSS30	SLCTOP	/TBL/ and /TABDIM/ : see SSS29. Label list of INKB has been extended to 5 labels. Parameter list of PRTEXT contains the dimension of the table. Display of the operation name has been added at label 125 and 165.
SSS31	PRTEXT	/EKW/ : see SSS29 . Parameter list has been extended with IDIM, the dimension of the table ICODE. KEYP=NKW(ICODE(K)) is now KEYP=ICODE(K) . DTEXT(' ',1) is now CHOUT(32)
SSS32	UPLOG	/TBL/ : see SSS29 . Dimension bounds have been adjusted.
SSS33	INKB	deleted
SSS34	LKUPDS	deleted
SSS35	LKUPKW	deleted
SSS42	INKB	new
SSS43	DECODE	new
SSS44	CMPWRD	new

SSS45	GETNRS	new
SSS46	NAMENR	new
S1401	SASR14	/TBL/ , /EKW/ and /TABDIM/ : see SSS29 . OP has been declared to be integer. Dimension bounds have been adjusted. Label list of INKB has been extended to 5.
S1501	SASR15	see S1401.