Abt. V Co – Lochstreifen,
Bezug 1.2.76.

1 Lochstreifen 16,95 mm 101 Zeilen

1 Lochstreifen 107 Zeilen, Code "Hollerith"

1 Lochstreifen 142 Zeilen, Code Fundstellen

1 Lochstreifen Testprogramm gelb:

101, 108, 120, 136, ~~142~~, 144
180, 184, 190, 205, 226, 233

3. Korr.

Konrad Zuse

Der Plankalkül

Contents

published by :

"Die Gesellschaft für Mathematik und Datenverarbeitung, Bonn"

1972

Achtung!
Die Seitenzahlen wurden um 12ahl verschoben, da eine Seite übersehen wurde (36a!)

Prefactory Note

The "Gesellschaft für Mathematik und Datenverarbeitung (GMD) mbH Bonn " has been in close contact with Professor. Dr. Ing. Konrad Zuse over a long period. In addition to a number of contacts a symposium " Viewpoints for the development of algorithmic languages " was held at which Professor Zuse read a paper on his " Plankalkül " .

I am particularly anxious that the publication of the " Plankalkül " by the " Gesellschaft für Mathematik und Datenverarbeitung " should take place at this time and, hopefully, promote further interest in these fields of study.

F. Krückeberg

Preface to the English Version

The English version was published in 1975 by the " Gesellschaft für Mathematik und Datenverarbeitung mbH, Bonn ". My special thanks go to Dipl.-Ing. Gerhard Overhoff for the assistance of the translation. In the years 1940 – 1945 he has already been interested in my research, stimulating and supporting it. Much to my regret Mr. Overhoff died January 27, 1975, after having completed this translation.

The English version contains some modifications in comparison to the German version. Some sections in chapter 3 are omitted. Furthermore, the Plankalkül has been revised in the last two years. This led to some modifications in the commentary. In the original Plankalkül – in order to preserve its historial form – only a few typographical errors were corrected. For this reason, the English version too deliberately contains some incorrect programs.

For the version of the Plankalkül I want to thank also Mr. Dipl.-Inf. Hohmann, the "Deutsche Forschungs–Gemeinschaft " , " Gesellschaft für Mathematik und Datenverarbeitung mbH Bonn " and the " Siemens Gesellschaft " .

K. Zuse

Preface

The Plankalkül was developed and written by me in 1945 in the small village of Hinterstein in the Alps. The difficult situation prevailing after the war meant that I was unable to continue the hardware research that I had begun in 1936. This did mean however that I was able to concentrate on theoretical studies and as a result of these the concept of an algorithmic language was born. I called it " Plan - kalkül " , but unfortunately it could not be published at the time.

Over the past 30 years the study of algorithmic languages became a science in its own right, and one with considerable practical importance. Some of the developments that took place in this field followed the lines I had proposed, while others took a different course. For example, PL/1 and Algol 68, two modern programming languages, have something in common with Plankalkül (see " The Plankalkül of Konrad Zuse, a Forerunner of Today's Programming Languages" in " Elektronische Rechenanlagen 1972, Heft 2 ") .

I believe that Plankalkül is still of significance today, and the publication, which was not possible during the early post–war years, can now be achieved. As part of this publication a concise summary of the preparatory work I had intended to write for my doctoral dissertation " Statements of a theory of general calculating " is included. There is also a comment written by me in 1972.

The work, supported by " Siemens Gesellschaft, München ", is published by " Gesellschaft für Mathematik und Datenverarbeitung mbH, Bonn " . Special thanks are due to Professor Gumin and Professor Krückeberg for their assistance. It is my hope that this publication will stimulate further fruitful cooperation.

STATEMENTS OF A THEORY OF GENERAL CALCULATION

with special consideration of the Calculus of Propositions and its application to relay circuits.

by Konrad Zuse 1944

Concise Summary

Published 1972 as an addentum to the publication of the Plankalkuel (PK)

Preface

The paper on the Theory of General Calculation was written during world war II by the author who intended is as his doctor thesis. However, as a consequence of the unfortunate conditions prevailing in Germany during and after the war, he could not realize his plan and the paper was not published at that time.

There exist a great number of publications on the subject today and the application of mathematical logic to the design of computers and the theory of switching have developed into a special sience.

The author was unfortunately occupied with other tasks after the war, which did not allow him to continue with his theoretical work on this subject.

In some ways the paper be regarded as preparation for the Plankalkuel . The following is the text of the paper in concise form, and it should be helpful for an understanding of the Plankalkuel .

Normal typing is used for the original text of the manuscript, and Italics for later additions.

Introduction

The voluminous repetitive computations associated with statically undetermined systems encountered during studies of Civil Engineering induced me, to conceive an automation of the numerical computations. In pur - suit of this idea I built several experimental computer models.

My first aim was to construct computers merely for numerical computations. However, in the course of my design work I soon developed a concept for combinatorial computations. The difficulties in the design of the complicated control units of my computer models and the realisation that, in principle , all calculating ope - rations can be solved by means of relais cicuits, induced me to develop a "Conditional calculus".. Later , I discovered that my Conditional Calculus corresponded to the Calculus of Propositions. Supported by the latter's logical formalism , I elaborated statements to a theory of General Calculation with special conside - ration of the Calculus of Propositions and its applications to the design of relay circuits.

I found that the term " calculate " has different meanings depending on its use in the language. In science , engineering, and economy " calculate " is generally used in association with numerical operations. Colloquialy however , it is frequently applied to combinatorial processes.

A practical example for a non numerical calculating operation is the derivation of a mathematical expression , i.e. the application of a formuly by which the derivative of a given algebraic expression can be produced. Most of the mathematical transformations of expressions, equations, and systems of formulas can be considered

as calculations, if clearly defined algorithms exist for the production of results as functions of given data .

In the solution of engineering problems, the development of a formula is very often only schematic routine work and , as such , a calculating process in the combinatorial sense. It is the programming of the numerical computation for a system of a given structure. The inputs for this procedure are the data of the topological structure of the system and the output is the agorithm, i.e. the program for the computation of desired resulting values .

Similarly, payroll accounting not only comprises numerical computing, but also combinatorial calculating, since the course of the numerical computations is governed by a program, which is dependent on many conditions, like marital status, number of children, overtime hours, etc.

In statistics , also numerical computations form only a part of the calculating processes. The selection of Data according to certain criteria, theire valuation, classification, and sorting represent combinatorial operations.

So far, nobody seems to have felt the need to incorporate all of the combinatorial operations in a uniform formalism.

Even in railway engineering, where clearly defined specifications govern the operation of switching and signalling devices, where complex mechanical gears have been developed, which realize the specifications , no systematic formalism has been applied so far .

The paper is intended as a stimulus for concerned parties. In this , it is my aim to familiarize the practical engineer with mathematical logic.

For this purpose the formalism of logic must be adaptable to practival use by enegineers and consequently must be free of any philosophical reasoning.

Chapter 1.

Statements of a General Calculations Calculus

A) General Introduction

1) Definition of the term "Calculation"

In the following text we will regard all schematic operations, formulas, derivations, and algorithms as calculating processes if, according to a program, resulting data are computed from given input data. The numerical calculations belong to the lowest level of the calculation thus defined.

Consequently, to calculate means: " To compute new data according to a (algorithm) program".

2) Explanation of the Term "Data"

a) General Data

Data can be of very different nature (modes, types), they may be numbers, names, adresses, signals, ranks, coordinates, etc. All data have content. The content is the meaning of the data. We must distinguish between the constant and the variable part of the data. The difference between the two can best be explained using the example of a form or questionaire with preprinted constant information and blank spaces provided for the addition of variable information, e.g. name, birthday marital status, etc.

The preprinted information corresponds to the constant part of the data and the blank spaces to the variable part. As long as the blank spaces are not filled in they represent undetermined values of variables. In order to calculate with such variables one has to refer to them by means of symbols and to interrelate them by formulas. The varibility of the data is determined by the number of the values which their variable part can assume. Consequently, the sign of a number is a two–fold variable, of a decimal digit a ten–fold and of a letter a twenty–six fold variable.

b) Two–fold Variable Data

The most simple form of data are the two–fold variables. Leibniz applied this perception to numbers by developing a number system with only the two digits 0 and 1, which he called "Dyadik". This Binary System has also been applied in other fields, for example in the Calculus of Propositions with the values true and "false" instead of 0 and 1. Henceforth in this text, such two–fold variables will be called "Yes–No–Value". In principle, any data, no matter how complicated can be represented by Yes–No–Values.

Decimal numbers are composed of ten–fold variable digits. Each of these digits can be represented by 4 Yes–No–Values if coded in the binary system. For example:

$$73 = \overset{7}{0}LLL \qquad \overset{3}{0}0LL$$

To distinguish the decimal figure 1 from the binary "On" or "Yes", the corresponding binary values will be represented by a capital L, the "Off' or "No" by a Zero (0).

The corresponding binary number is :

$$73 = L00L00L$$

Letters can be coded by 5 Yes – No– Values, as has been done in teleprinting.

In general, a series of n Yes – No – Values has a variability of 2^n. Therefore, 5 Yes – No – Values can represent 32 characters : e.g. the 26 letters of the alphabet and 6 others. The spaces between characters also have to be treated like characters.

Representation of Yes – No – Values (Y–N–Values) :

Y–N–Values can be represented in determined or in undetermined form. For the determined form we need two–fold variable symbols, for instance (–,+) or (0,L). For the undetermined form we may use letters.

c) Structure of Data

In principle, any data can be represented by a sequence of Y–N–Values, but the sequence may become infinite. An irrational number, as an example, can only be represented exactly by an infinite sequence of digits. In practice, numbers are limited by a finite number of digits. In this case, the data are of a fixed structure.

In contrast to those are data of variable structure. The number of the employees of a factory, for example, is varying in size.

Variables of composed but fixed structure are for instance:

complex numbers, vectors, matrices, etc.

B) Formalisms Interrelating Data

1) The Term Algebraic Dimension

There is a set of formalisms available to interrelate data. In Logic the term Operation is used if the combination of two variables of the same type produces a result of the same type. Such interrelations and combinations represent the basis of Algebra as a whole. Addition , subtraction, multiplication and division are operations in this sence. Composed data, like vectors and matrices can also be connected by operations, but they do not always produce a result of the same type. Althrough the sum of two vectors is again a vector, their scalar product is a number.

The Calculus of Propositions utilizes a formalism in which the combination of two Y-N-Values again produces a Y-N-Value [+)] . Propositions may be true or false. While Mathematical Logic aims to arrive at true conclusions based on a given set of axioms, only the syntactic combination of Y-N-Values does matter here. Therefore, there would be no sense in using the terms " true " and " false " for our purposes.

Formalisms can be established for very different types of tasks, all of which may be called " algebraic " . The well - known logician Schröder spoke of the " Algebra of Logic " . Since we will soon have to handle formulas in which data a different type must be interrelated, we will introduce the term " algebraic dimension " , to define the structure of the data; for instance :

Yes-No-Values (Y-N-Values)
real number
complex number
plane vector
spacial vector
Pair of coordinates
matrix
determinant of n^{th} order

Formulas with variables of the same algebraic dimension only will be called " formulas of homogenous algebraic dimension " .

The expression

$$a^2 + b^2$$

is such a formula because all its values are numbers.

+) Hilbert-Ackermann, Grundzüge der theoretischen Logik, 2. Aufl. , Berlin 1937
Hilbert - Bernays, Grundlagen der Mathematik, 1. Band, Berlin 1934

The formula

$$(a + b)^2 = a^2 + 2ab + b^2$$

is also composed only of numbers, however the expression as a whole is not a number, but a statement of the dimension Y-N-Value which we also consider to be of homogenous algebraic dimension.

2) Formulas of Homogenous Algebraic Dimension

a) The Calculus of Proposition

The chapters 2) to 4) contain a description of the formal aspects of the Calculus of Proposition in the representation given by Hilbert - Ackermann. The following chapter 5) may deserve special attention :

5) Binary Numbers as Propositions

A binary number can be represented by a conjunctional combination of propositions Z_n , wherein Z_n means that the n^{th} power of 2 is a component of the number. In this way the number L00L can be represented by the following formula :

$$Z_3 \wedge \overline{Z_2} \wedge \overline{Z_1} \wedge Z_0$$

In this form we cannot only characterize single numbers, but also sets of numbers. If we confine the application of the expression to integral numbers with four binary digits, the expression $Z_3 \wedge \overline{Z_0}$ defines the set of the following numbers :

L000	(8)
L0L0	(10)
LL00	(12)
LLL0	(14)

Special attention is then given to "Normalized Forms" and the "Principle of Duality", because of their great importance in the design of circuitry.

The following chapter is of special importance for the "Plankalkuel".

6) Calculus of Classes, Predicates , and Functions

The possibility of reducing all data to a sequence of Y-N-Values implies the possibility of tracing any complex calculation back to the Calculus of Propositions. In this, the complex representation is coded into a sequence of Y-N-Values.

The employees of a factory, for instance, can be identified by a number with four decimal digits. This number can be transformed into a binary number. We get a subset by selecting a special class e.g. the female workers. The list of the numbers of these persons corresponds to the disjunctive normal form. A simplification of this form is handly possible if the numbering of the employees was continuous without regard to their sex.

By introducing another coding system, e.g. by assigning special digits for sex, family status, job, business , age, etc. we are able to calculate with this code by applying it to the calculus of propositions. This is generally practised in statistics, however, without recognition of the logical character of these operations.

As an example, the entire code for one person can be composed as follows :

1) Age	100 - fold variable
2) Sex	2 - fold variable
3) Family status	4 - fold variable
	(single, married, widowed, divorced)

Coding with binary digits we need seven digits for item 1) one digit for item 2) and two digits for item 3) . We can call them A_0 - A_9 :

A_6 A_5 A_4 A_3 A_2 A_1 A_0 = Age (binary number)

A_7

–	female
+	male

A_8 A_9

–	–	single
–	+	married
+	–	widowed
+	+	divorced

To select the persons with the following characterists

1) female
2) single, widowed, or divorced
3) age between 16 and 31 years

we can use the following formula to classify them :

$$\bar{A}_7 \wedge \overline{\bar{A}_8 \wedge A_9} \wedge \bar{A}_6 \wedge \bar{A}_5 \wedge A_4$$

or

$$\bar{A}_7 \wedge A_8 \vee \bar{A}_9 \wedge \bar{A}_6 \wedge \bar{A}_5 \wedge A_4$$

As an algebraic dimension we have developed a sequence of Y-N-Values for a class of persons. Composed data of this type we can represent by a single symbol, e.g. x . x_0 , x_1 , x_2 , ... then are the values which x can assume. The composition of the values of x produces a new class. The propositional formula is a predicate of the algebraic dimension Y-N-Value , representing a proposition with the meaning :

" x has the property A " .

In this way we can define a predicate by a " definition equation " :

$$Pd_1(x) =_{Df} \bar{A}_7(x) \wedge A_8(x) \vee \bar{A}_9(x) \wedge \bar{A}_6(x) \wedge \bar{A}_5(x) \wedge A_4(x)$$

We can characterize the class selected by this predicate as follows :

$$\hat{x}\ (Pd_1(x))$$

(those x, to which the predicate Pd_1 applies)
Now we introduce other predicate :

$$Pd_2(x) =_{Df} A_7(x) \wedge \bar{A}_8(x) \wedge \bar{A}_9(x)$$

Then $x\ (Pd_2(x))$ represents all male single persons.

The predicate :

$$Pd_3(x) =_{Df} (\bar{A}_6(x) \wedge A_5(x) \wedge A_4(x)) \vee A_6(x))$$

selects all persons older than, or exactly 48 years of age.

Now we are able to form the following predicate :

$$Pd_4(x) =_{Df} Pd_2(x) \wedge \overline{Pd}_3(x)$$

Then $\hat{x}\ (Pd_4(x))$ represents the class of all male single persons who are less than 48 years old.

It is the aim of this paper to deal with " computable functions " . These are functions which provide the possibility of deriving new data from given data. In the expression

$$[<(2,3) \wedge <(3,7)] \rightarrow <(2,7)$$

$<(2,3)$ represents the proposition : two is less than three, etc.

In mathematical logic the validity of these propositions is checked by their derivation from axioms. From a calculating aspect the expression is a formula with two numbers as variable and a proposition as result. The algebraic dimension of the variables must be known for the calculation of this result.

The algebraic dimension can be independent of the technical structure of the data. The rules of geometry for the elements " line " and " point " e.g. are valid regardless of the form in which these elements are represen - ted, (as vectors, equations, pairs of values , etc.). But for practical calculation their structure must be known.

If, for instance, in the function $<(a,b)$ the numbers a and b are represented by binary numbers of three digits, then the propositional formula is a follows :

$$(b_2 \wedge \bar{a}_2) \vee [(b_2 \sim a_2) \wedge (b_1 \wedge \bar{a}_1)] \vee [(b_2 \sim a_2) \wedge (b_1 \sim a_1) \wedge b_0 \wedge \bar{a}_0)]$$

Here $a_2, a_1, a_0, b_2, b_1, b_0$ represent the digits of the binary numbers a and b with the values $2^2, 2^1, 2^0$.

In the original text is following an introduction into the " All " and " Existance " operators. Their impor - tance for General Calculation will be dealt with in the " Plankalkuel ".

The following chapter may also deserve interest :

7) Formulas of Nonhomogenous Algebraic Dimension

A simple example are the already mentioned relations $<,>$. They connect numbers with each other, which however, do not produce another number but a Y-N-Value. In the expression

$$(y = 3x^2) \wedge (x > 2) \rightarrow (y > 12)$$

operations with numbers and propositions are combined in one formula.

In this context the symbol = represents an operation.

Another example is :

$$s = \frac{b}{2} t^2 \quad \text{eq} \quad t = \sqrt{\frac{2s}{b}} \qquad (\text{eq : equivalent})$$

The two formulas are equivalent and represent the same relation in different form.

Similarly, we can write :

$$s = \frac{b}{2} t^2 \wedge v = b \cdot t \rightarrow s = \frac{v \cdot t}{2}$$

Here is logically exactly formulated what is usually put in plain language.

If we want to state that only the positive value of a squareroot must be considered then we write:

$$y > o \wedge y = \sqrt{x}$$

A well - known operation of nonhomogenous algebraic dimension is the scalar product by which two vectors are connected and produce a scalar value.

$$y = (\mathfrak{A} \; \mathfrak{B})$$

8) Development of Calculations of Higher Order on the Basis of the Calculus of Propositions

It has been already shown how numbers can be represented by Y-N-Values. Now we will see how operations with binary numbers, e.g. addition, can be performed with the Calculus of Proposition.

Our operands are the binary numbers x and y with the digits

$$x_n \cdots x_i \cdots x_1 , x_o \qquad\qquad y_n \cdots y_i \cdots y_1 , y_o$$

We are asking for the sum z with the digits

$$z_{n+1}, z_n \cdots z_i \cdots z_1 , z_o$$

At first, we compute the sum of the digits of each single digital position without considering the carry - over . It can be 0, L , or L0 . The latter is a binary number with two digits. The last digit of this number we call c_i . Then, we estabish the following table for c_i :

x_i y_i	c_i
0 + 0 = 0	0
0 + L = 0L	L
L + 0 = 0L	L
L + L = LL	0

We can write : $c_i \text{ eq } (x_i \not\sim y_i)$

We now calculate the carry - over notation d_i , which indicates, if a carry - over has to be made from the position $i - 1$ to the position i . This is the case, if the digits x_{i-1} and y_{i-1} both equal One , or if c_{i-1} and d_{i-1} are both positive :

$$d_i \text{ eq } (x_{i-1} \wedge y_{i-1}) \vee (d_{i-1} \wedge c_{i-1})$$

z_i can be computed from c_i and d_i . If there is no carry - over to the position i , $(\overline{d_i})$, then z_i equals c_i , if there is a carry - over then z_i is disvalent to c_i .

$$z_i \text{ eq } (c_i \not\sim d_i)$$

For binary numbers with four digits the formulas for addition are the following :

$(x_0 \not\sim y_0) \text{ eq } c_0$	$x_0 \wedge y_0$ eq d_1	c_0	eq z_0
$(x_1 \not\sim y_1) \text{ eq } c_1$	$(x_1 \wedge y_1) \vee (d_1 \wedge c_1)$ eq d_2	$(c_1 \not\sim d_1)$	eq z_1
$(x_2 \not\sim y_2) \text{ eq } c_2$	$(x_2 \wedge y_2) \vee (d_2 \wedge c_2)$ eq d_3	$(c_2 \not\sim d_2)$	eq z_2
$(x_3 \not\sim y_3) \text{ eq } c_3$	$(x_3 \wedge y_3) \vee (d_3 \wedge c_3)$ eq d_4	$(c_3 \not\sim d_3)$	eq z_3
		d_4	eq z_4

Thus is demonstrated, that the addition of numbers can be performed with the Calculus of Propositions. It is known, that the more complex arithmetic operations can be reduced to addition and subtraction.

Consequently, arithmetic operations can be represented by operations of the Calculus of Propositions.

In the chapter " computation of Formulas " the difference between implicit and explicit expressions is discussed first. This is clear in normal algebra. The rules of algebra usually allow a simple implicit formula to be transformed in such a way that the wanted value stands isolated on one side of a equation.

This is not always possible with propositional formulas. It has also been attempted here to develop general rules for explicit representation in equations. But this development does not seem to be of practical importance . The " Plankalkuel " has not been based on it. Therefore, this section is omitted here.

The following chapter is of importance for the " Plankalkuel ".

9) Inflexible Programs

A method, especially suited for calculating machines, is the use of programs which consist of a sequence of operations which the machines have to perform in order to compute the results. For this purpose the inter -

mediate values have to be named, too . A simple method is that to number all variables including the input - variables systematically with V_1 , V_2 ... and then to list each operation separately :

For example :

$$x = \frac{a}{2} \pm \sqrt{\frac{a^2}{4}} - b$$

Input values

$a = V_1$

$b = V_2$

Program

$V_1 : 2 = V_3$

$V_3 \cdot V_3 = V_4$

$V_4 - V_2 = V_5$

$\sqrt{V_5} = V_6$

$V_3 \cdot (-1) = V_7$

$V_7 + V_6 = V_8 = x_1$

$V_7 - V_6 = V_9 = x_2$

2 and –1 are the constants in this formula.

It can easily be seen that calculations of any length can be performed in the above way, if they can be reduced to the elemtary operations.

The program for a determinant of the degree three looks like this :

$$\Delta = \begin{bmatrix} V_1 & V_2 & V_3 \\ V_4 & V_5 & V_6 \\ V_7 & V_8 & V_9 \end{bmatrix}$$

$V_1 \cdot V_5 = V_{10}$

$V_{10} \cdot V_9 = V_{11}$

$V_2 \cdot V_6 = V_{12}$

$V_{12} \cdot V_7 = V_{13}$

$V_3 \cdot V_4 = V_{14}$

$V_{14} \cdot V_8 = V_{15}$

$V_1 \cdot V_6 = V_{16}$

$V_{16} \cdot V_8 = V_{17}$

$V_2 \cdot V_4 = V_{18}$

$V_{18} \cdot V_9 = V_{19}$

$V_3 \cdot V_5 = V_{20}$

$V_{20} \cdot V_7 = V_{21}$

$V_{11} + V_{13} = V_{22}$

$V_{22} + V_{15} = V_{23}$

$V_{23} - V_{17} = V_{24}$

$V_{24} - V_{19} = V_{25}$

$V_{25} - V_{21} = V_{26} = \Delta$

Such programs can be developed for any type of formula of homogenous algebraic dimension.

In propositional formulas, the elemtary operations are

$$\wedge, \vee, \neg \; [\sim, \not\vdash, \rightarrow]$$

It is possible here to number the variables singly in sequence.

For example :

$(a \wedge b) \vee (c \wedge d)$ eq e

$a = V_1$

$b = V_2$

$c = V_3$

$d = V_4$

program : $V_1 \wedge V_2$ eq V_5

$V_3 \wedge V_4$ eq V_6

$V_5 \vee V_6$ eq V_7 eq e

In principle, the shape of the program is the same as that for the handling of numbers. Thus, we are able to apply the program for the determinant also to Y-N- Values. Our problem is then the following :
The given elements are a_1, b_1, c_1, and a_2, b_2, c_2. The relation $R(x, y)$ means : "x fits y". x we substitute by a_1, b_1, c_1, y we substitute by a_2, b_2, c_2. Now we develop the matrix for the relation $R(x, y)$, in that we coordinate the elements a_1, b_1, c_1 to the lines and a_2, b_2, c_2 to the columns. Then the symbols + or – attached to the nine elements of the matrix, which designate if the relation for the corresponding pair is valid or not. In undetermined form the symbols are substituted by the variables $V_1 - V_9$.

	a_2	b_2	c_2
a_1	V_1	V_2	V_3
b_1	V_4	V_5	V_6
c_1	V_7	V_8	V_9

If in the program, we now substitute the multiplication by the conjunction ($\wedge$) and the addition or subtraction by the disjunction ($\vee$) for the determinant, then we get as a result a Y-N-Value, which indicates, whether a distribution of the elements $a_1, b_1, c_1 ; a_2, b_2, c_2$ is possible for the given elements V_1 to V_9 of the matrix in which the elements of each pair fit together. The following problem demonstrates a practical application : At a party of three ladies and three gentlemen not all possible pairs fit together. We must look for a seating arrangement which will provide each lady with a gentleman to suit her.

The formula of the propositional determinant renders the list of possible pairs to be formed as follows :

$$(V_1 \wedge V_5 \wedge V_9) \vee (V_2 \wedge V_6 \wedge V_7) \vee \ldots \vee (V_3 \wedge V_5 \wedge V_7)$$

Different levels of programs also exist which are similar to the different levels of formulas of homogonous algebraic dimension . The lowest level is again occupied by the programs with Y-N-Values. Since it is possible to compile the higher level operations by means of propositional operations, it is in principle also possible to compile numerical programs by the same means.

It is also possible to compile programs of a higher level, for example those with complex numbers, by means of numerical operations.

Furthermore, we are able to design programs in which matrices represent the input variables and in which the operations with them stand for the corresponding combined operations of matrix calculating.

For the execution of such programs, the calculating unit should be designed to perform these operations. However, it will be advantageous to employ " subprograms " for repetive computations.

Programs for variables and operations of different algebraic dimension also certainly exist . They would require calculating units for operations of different levels and storageunits for data blocks of varying sizes.

All these programs are of a fixed structure, which means that the type, number, and sequence of the variables and operations are the same for all computations and that the content of the data only changes.

Flexible Programs

Contrary to inflexible programs, High Level Programs handle programs in which the input variables influence the sequence and type of operations. If an employee gets a monthly salary of less than RM 300,- he is not obliged to pay for health insurance. The computation of the reductions is a function of the salary. Another example is the computation of the area of two plane figures which may or may not partly overlap and where the overlap area is only to be considered once . Again this program varies, depending on the input variables.

In principle, it can be demonstrated, that such variable programs can be reduced to inflexible programs if large sections of them are allowed to run idle. So a program for a full matrix can be used for a reduced matrix in which some of the elements are equal to zero. This method is not economical however , because numerous multiplications by zero have to be performed unecessarily.

The programs for the " All " and " Existence " operators with series of conjunctions and disjunctions are another example. These operations must be executed for all elements with an inflexible program. This method is redundant, since the Existence Operator already yields „ positive " as soon as the first positive element occurs. Likewise, the All Operator already yields „ negative " , when the first negative element occurs. In this case the variability of the program is the length of the compution.

As far as is possible upto now a summery of the different types of programs is shown as follows.

I. Inflexible Programs

a) Programs of homogenous algebraic dimension

1) For Y-N-Values

2) For real numbers

3) For complex numbers

4) For vectors, matrices, etc.

b) Programs of nonhomogenous algebraic dimension

II. Flexible Programs

This survey is not generally valid. We are able, for instance, to put another level between 1) and 2) for programs in which only single additions occur and where the arithmetic operations for real numbers are reduced to additions.

The flexible programs represent the true field of higher level combinatorial computations. However, it is not yet possible to deal with them here at this stage.

These Flexible Programs will be a special subject of the Plankalkuel.

In Chapter 2 the application of the Calculus of Propositions to relay circuits is discussed.

At that time (around 1940) publications by Shannon and C.H. Piesch which already existed, dealt with problems of switching algebra. The Calculus of Propositions and the laws of switching are systematically compared in chapter 2, contrary to their treatmentin those papers. In this way - in my opinion - the problem became much clearer. The application of the Duality - Principle for Propositions to relay circuits proved to be very advantagenous.

Furthermore, different types of relays are discussed, e.g. the mechanical switching element, the electro - magnetic relay, and the tube - relay (based on the research of Dr. Schreyer). In order to become independent of certain relay techniques I introduced an "Abstract Switching Representation".

Nevertheless, these investigations are of no direct importance for the "Plankalkuel".

Prof. Konrad Zuse

COMMENT OF THE " PLANKALKUEL "

Introduction

The " Plankalkuel " was developed by me in the years before 1945. I wrote the following comment in 1972. The " Plankalkuel " – hereafter called PK – is reproduced with this comment in its original form. It was translated into English in 1974. In order to facilitate the understanding of the PK this comment occasionally refers to algorithmic languages in use today, like Fortran, Algol 68, PL /1.

The term " Kalkuel " (calculus) does not correspond to its meaning in mathematics in a strong sense. It is more what we call an algorithmic language today.

First, I want to comment on the development of the German computer upto 1945, the year in which the PK was established in its final form.

A) Development upto 1945

The German computer developed entirely independently of those developed almost simultaneously in the United States and other countries.

I. Development of hardware

1) 1934 : I was a student of civil - engineering at the Technische Universität Berlin - Charlottenburg and had already developed the basic concept of a program - controlled calculating machine with the following features :

a) Program - control by punched tape with one address code or three address code.

b) Application of the Binary System to arithmetic unit, storage unit and adress - coding.

c) Floating point representation (at that time I called it " Halblogarithmische Form " (semi - logarithmical form) .

d) Application of Bi - stable switching elements, e.g. relays.

II 2) 1936 : I began to construct test models, the first was model Z1, in a purely mechanical technology. I developed a " Mechanische Schaltgliedtechnik " (mechanical switching element system) . This model was completed about 1938; but it was operable only in some parts. (E.g. the arithmetic unit in binary system with floating - point and the storage unit).

3) 1938/39: Construction of the model Z2. Its arithmetic unit had electromagnetic relays (fixed - point binary representation with 16 bit word length). This model was also a test model. But it demonstrated the feasibility of my computer development.

4) 1939/41 : Construction of the model Z3. This was contructed using electromagnetic relay - tech - nique solely. (Binary system, floating point, 22 bit word length, 8 - channel code for punched tape control, 64 - word storage.)

The model Z3 was completed in 1941 and was the first completely functioning program - control - led calculating machine in the world. A number of mathematical problems were tested on it (linear equation - system,) quadratic equations, determinants, matirces, and special aerodynamic

problems, e.g. wing fluttering) .

The Z3 computer was destroyed in an airraid in 1944 .

5) 1942:Contruction of a special computer model S1 to measure the wings of guided missiles.

This computer was installed in the production line for the HS 293 (Henschel Flugzeugwerke) guided missiles and was intended to eliminate the wing - inaccuracies of this missile with the aid of a special calculation. (Electromagnetic relay - technique, binary system, fixed point, sequential hardware program with rotary switches.)

6) 1943/44 : Construction of the Computer Model S2. It was built for the same purpose as model S1, but the metering clocks were connected directly to the calculator and were read automatically through the program (Analog - to - digital conversion). This application represented the realization of the first process - control by a computer in the world.

7) 1942/45 : Construction of the Universal Program - controlled Calculator Z4. This computer re - presented an extension of the computer Z3 (electromagnetic relay - technique , binary system, floating point, punched tape control, mechanical storage unit, originally designed for a capacity of 1024 words). The computer was completed at the end of world war II for the computation of simple programs;

8) 1944 : Construction of a small test model for Propositional Calculus in relay technique.

9) 1937 : In cooperation with Dr. Schreyer the first steps towards the development of electronic computers were undertaken. First Dr. Schreyer constructed a test model fitted with electronic tubes (see thesis of Dr. Schreyer) .

10) 1940 - 45 : Dr. Schreyer contructed another test model using electronic relay technique at the Technische Universität Berlin - Charlottenburg with an arithmetic unit of 10 binary digits.

11) Around 1944 : Dr. Dirks constructed a test model with a magnetic storage unit (disc) and simple electronic calculating units. This development was performed independently of the work of Dr. Schreyer and myself. Dr. Dirks and myself had no knowledge of each other until 1952.

II. Development of Software

1) 1936/37 : Development of a "Bedingungs - Kombinatorik " (Conditional Calculus) . The con - sequent use of the relay technique, in other words, the application of construction elements with two distinct stages induced me to analyse theoretically the rules which such constructions and cir - cuits follow. This resulted in the " Bedingungskombinatorik ". Later on , I realized that my calculus was formally identical with the Calculus of Propositions.

In this way I created "Schaltalgebra" (switching algebra). Now I was able to design my computer models in an " Abstrakte Schaltgliedtechnik " (abstract switching element technique). The mecha - nical " circuits " of the model Z1 essentially corresponded to the electromechanical circuits of my later models Z2, Z3, Z4 .

The switching theory also facilitated the design of our electronic computers. Schreyer only had to develop electronic switching components for the propositional operations Conjunction, Disjunction ,

and Negation and to connect them in accordance with the theoretical circuits already developed.

2) 1943/44 : Ansätze einer Theorie des Allgemeinen Rechnens (Statements of a General Calculation Theory) . I elaborated the concept outlined in II. 1) with the intention of making it the subject of a doctor - thesis. An extract of that paper has been enclosed because it will facilitate the under - standing of the PK .

3) 1943/44 : Preparation of a general purpose algorithmic language (PK) in the form of some notes.

III. Design of Computers during World War II

My computer models, constructed during the war, were ordered for military applications. Because of the limitations of personnel and material they aimed at cleary definable goals , which I could achieve with the limited recources and limited deadlines then at my disposal . The possibilities of my swit - ching algebra were systematically applied. Consequently with regard to their logical design, I believe that my computer models may have been well ahead of the contemporary developments in the USA.

Nevertheless, I intentiously made no use of the facilities of conditional instructions, address trans - formations and others, since I could not run the risk of increasing the capability of my computers without adequate recources for their realization, e.g. the construction of storage units of the required high storage capacity.

Despite this, I already designed during World-War II several theoretical models which utilized logical instructions, program selection, address computation, and the facilities of my Conditional Calculus. I already realized then that computers must also be able to store programs as well as data. But to me this capability seemed to be such a self - evident requirement then, that I omittet to apply for a patent for its realization ~~(see PK- chapter 1, page ...)~~ X Bitte Seitenzahl angeben!

Some of the aforementioned capabilities were incorporated into the design of the Z4 model which was in construction during the war. They were realized practically, when the Z4 model was installed at the " Eidgenössische Technische Hochschule " (ETH) in Zürich in 1951 .

I was then particularly occupied with the problem of content addressable memories, and the organisa - tion of storage and retrieval(this term may not have been used in 1945) of data of varying struc - tures as outlined in the PK.

Already in the years between 1937 and 1945 I had in mind to construct a special - purpose calculator " Planfertigungsgerät " for the Construction of programs in addition to the usual computer for the execution of numerical programs. Actual development, later, went in a different direction . The ordi - nary computer was improved step by step to deal with logical problems. The problems of my " Plan - fertigungsgerät " are solved today by compilers or generators on ordinary computers.

Most of the theoretical studies which I undertook during the war remained on paper, and of these , most have been lost.

IV. Knowledge of Other Developments

As already mentioned, developments other than mine and that of Dr. Schreyer were performed independently. I only became acquainted with the fundamental work of Babbage in 1938, when I was applying for a patent in the USA. Moreover, I had no knowledge of the very important works of Turing (" On computable numbers ") , and the Switching Algebra of Shannon. Only by accident in 1944 did I learn of the Aiken computer, USA. However, the picture of the MARK I computer supplied by the German secret service could give us only a general impression.

B) The Situation in 1945

Most of my models which were constructed during the war in Berlin were destroyed in airraids. Only the Model Z4 could be saved in an adventurous truck journey from Berlin to the Alps. There it was hidden in the small village of Hinterstein, Allgäu.

At that time, however, it was not possible to continue with the construction of the machine. My small staff of 12 dispersed.

So I had sufficient time to continue with my theoretical investigations.

The Model Z4 could hardly be operated. Moreover, we did not even need an algorithmic language to program this model.

In those circumstances I developed the PK as a desk - work theory, regardless as to wether suitable equipment for its implementation would be available in the near future.

C) Comparison of the PK with Other Algorithmic Languages

With regard to the aforementioned facts, the great difference which exists between the concept of the PK and the development of programming languages which were started later can be understood.

Cobol, Fortran, and Algol, for instance, were introduced in the years between 1950 to 1960 to satisfy the demand for programming facilities for the computers which had already been constructed and put to use. Accordingly, these languages corresponded to the daily occuring problems. Numerical problems were of major importance at that time. Only slowly did a trend develop towards universal calculation in the sense of the PK.

Then, however, I was quite alone with my PK - oriented concept . Contrary to the lively interest which my hardware development met with, my work in software (the term did not then exist) was hardly noticed. I discussed the PK only occasionally even with my best friends and my closest teammates. And then we disagreed, even in the essential concept; Moreover, the influence of the American developments was exceedingly strong at that time and dominated the general thinking.

The indifference towards the PK was somewhat disappointing for me , when the official discussions about Algol started in 1955. Some of the participants had sufficient knowledge of the PK to cooperate and in my opinion it would only have been fair if they had openly announced and utilized the ideas anticipated in the PK.

There are algorithmic languages available today which are also efficient from a logical point of view such as PL/ 1 and Algol 68. They offer some facilities, already contained in the PK, and beyond that, some formulations which facilitated the practical operation of computers.

This shows the decisive difference between the PK and all of the programming languages developed later : the goal of the PK is, to formulate the computational and logical coherence of a program by explicit and univocal instructions, but the PK does not attempt the practical execution of programs. There is no " imple - mentation" of the PK.

In contrast to that, the modern algorithmic languages developed through the practical use of computers . This is manifested by the input - output and transfer statements (READ, PRINT, PUT. etc.) , further - more, by facilities for the runtime organisation.

The PK was originally developed in the form of a two - dimensional representation. This form would have to be transferred into a sequence of characters, as is common with the other programming languages for the execution of programs .

In view of the previously described situation in 1945, it is obvious why I was not concerned with programs for numerical calculations when I developed the PK. I did not expect any difficulties in this field then , and consequently I concentrated my efforts mainly on the logical problems beyond the common numerical calculations. These circumstances may later have been the reason for the opnion that the PK was not the right base for the development of algorithmic languages. Moreover , the PK was almost unknown. After the completion of the PK in its preliminary form, I became completely absorbed by the management of a computer plant of my own. Besides that, for a long time no computers were available which could be applied to higher - level logical problems such as those exemplified in the PK. As a consequence of all this the PK was not published and remained sleeping in a drawer of my desk.

Today computer techniology has arrived at a stage where the discussion of the problems treated in the PK have gained great practical importance.

D) Future Importance of the PK

This paper is being published in order to compensate for the timely publication of the PK omitted in the past. Of course, we have to look at it differently today than we would have then.

In this paper I will not yet discuss in detail the question as to what extent considerations of the PK can be applied to future developments of algorithmic languages. But I nevertheless want to state that I believe that basic reflections are necessary. At present, tendencies in the field of the theory of formal languages and the problems connected with them diverge considerably. On the one side there are the ambitions of the people directly involved in the practical use of computers. Their interest is directed towards flowcharts of problems, easy implementation , and the development of compilers and operating systems.

On the other side a definitely theoretical tendency exists to solve the problems of the formal languages by means of modern mathematics. Although some useful theoretical knowledge has certainly be achieved, it cannot as yet be stated that these endeavours have produced results of importance to the computer software, if related to the investment.

I can only suggest that these problems should be considered now. In another paper I hope to be able to offer a concept which will be based on a solid theoretical foundation and, nevertheless , be applicable to practical problems.

THE PLANKALKUEL

Comments of Chapter 1

General Programs

II. 1) General Notations for Classification

The PK has adapted the method of Babbage to number the occuring variables systematically. In simple programs the numbers correspond to the addresses in the storage unit of the computer. This method can be extended for more complicate programs. In an algorithmic language , for instance the following notations must be distinguishable through different codes :

Variables , intermediate values, results etc. (nowdays also called objects, parameters ...)

Structures types and modes of data

Components of structured data, selectors

Programs , subprograms, procedures, functions, operators , etc.

Marks of labels for program request .

In the PK numbering and digital coding are used almost exclusively. In contrast modern algorithmic languages allow more freedom through the use of identifiers.

II. 2) Data and Their Representation

The specification of the structure of data distinguishes the PK probably most from other algorithmic languages. In the PK the structures are systematically developed out of Yes - No - Values. In this way tree structures which can be as complicated as necessary can be generated. From the beginning it is assumed that these structures may be variable themselves (e.g. a list with a varying number of elements , an array (row) or in the case of an array (row) of characters a " string "). The term " List " is used in the PK in another sense, than in modern „ List - Procerring " .

The indeterminate strucure characters σ , α allow a variability of the structures and types of data such as can scarcely be achieved by the modern algorithmic languages. The operator $N(V_0)$ in the PK corresponds to the operator upb in Algol 68.

II. 2) b) Limitation of Data

The " limitation of data " allows an accurate definition of the elements within the set of different values of a specified structure. Similar formulations are used for instance in PASCAL.

II. 2) c) Types of Data

In distinguishing between types of data, the PK not only deals with pure syntax but also considers semantic aspects. However, in the following chapters only limited use has been made of this facility.

II. 2) d) Modes of Data

The introduction of the notation " modes of data " is advantagenous, sinse it does not enforce an accurate definition of the data structures in every case. This corresponds e.g. to the presently used notations " REAL " , "INT " for the specification of numbers.

II. 2) e) Components of Data

Since the PK uses tree structures, the components correspond to the possible branching points of the tree.

In the PK the components are always numbered upwards, beginning with zero. This corresponds to the indexing of the digits of integral numbers, where the position of the lowest value is assigned the index zero, (corresponding to $2^0 = 1$ in the binary system) . This method of starting to count with zero , also corresponds to the fact that registers in a calculator have a starting position corresponding to zero (all elements reset). This has the disadvantage, that the index for the highest element is n - 1 in a list of n elements. Modern languages allow more freedom in this respect. There the upper and lower bound of arrays can be chosen randomly (ALGOL 68) .

In the PK there are no mnemonic notations for components like Re(v), Im(v) as there are in Algol 68, for instance which select the real or the imaginary component of a complex number.

Since only tree structures are directly defined in the PK , it is consequently not possible, for instance to define both the components " row i " and " column j " of a matrix directly. If the matrix is defined as a set of rows then Ki means row i . If we want to select the column j then we must select the single elements of the matrix and then assemble them in the column , for example :

(ko , j ; Kl, j ; ... Kn, j)

The PK , however permits to write a special program for this task which is applicable to all matrices m x n .

II. 2) f) Representation of Data

The beginning of this section contains the important operation concatenation of data, which corresponds to the inversion of the operation „ selection " of components.

The sequence of the components is now of importance. The natural sequence begins with the component to which the lowest index refers. In contrast, in the normal notation of numbers the digits with the highest index are written first. Therefore, we must look for a compromise. This would not be necessary for an internal representation in a computer, because there the digit with the lowest index is also transferred first.

In line with its basic concept the PK uses simple characters for the determinate notation of Y-N- Values (-, +; or 0,L) . The algorithmic languages developed later did not systematically utilize Y-N- Values as their base. Only in revised versions did they supplement the mode Boolean with the values " true " and " false " . This may be sufficient , as long as Y-N-Values are only of minor importance in a program. The use of the symbol o as an indetermined Y-N-Value is sometimes advantageous, but avoidable. This method could be extended to random data structures by the assignment of a non - special character to a certain structure. This is frequently applied in algorithmic representations through for instance the use of the digit " 9 " as a representative for decimal digits and the letter " x " as a representative for letters (e.g. 999.99 for a decimal number with 3 digits before and two digits after the point in COBOL) .

II. 2) g) The Two–Dimensional Representation

An important feature of the PK is representation in different lines. Its advantage lies in the fact that it gives the human user a good survey. We have to remember , that the PK in its form displayed here was designed to realize a univocal representation of complicated algorithms. For implementation, the programs of the PK must be converted into the form of a linear series of characters. This can undoubtedly be achieved by representation methods which correspond to those of modern alogrithmic languages.

The representation in lines allows the introduction of a special line for the specification of structure characters (mode indication) . This corresponds to the " declarations " in modern program languages.

The data in the structure line are often redundant and may be omitted then.

II. 2) h) Constants

Constants may obviously also be directly defined in determinate from by a notation of their value.

II. 2) i) Supplementation of Data by Numbering of Components

The operator I ( ) simplifies the handling of lists in a similar way to the operator N ( ) . I ( ) corresponds to the operator elem N ( ) to the operator upb in Algol 68.

II. 2) j) Data of Fixed and Variable Structure

The term " variable structure " corresponds to the term arrays with flexible bounds in modern algorithmic languages.

II. 3) Chapter 3 – Fixed Programs

The distinction between fixed , quasi-fixed , and free programs in the PK is a consequence of its history . The first computers, that of Babbage and my models Z1 to Z4 , were consciously limited to fixed programs. Babbage, even , was already aware of the conditional instruction , but did not apply it in the model which he constructed. Today , nearly all programs in use are free programs in the sense of the PK. Therefore, it is not to be recommended to adapt the terminology of the PK. Fixed programs are now-a-days called strict sequential programs (without branches and loops) .

The rules and terms defined in chapter 3 are valid for quasi-fixed and free programs.

II. 3) a) Designation of Programs

As already mentioned, the programs are designated by indices in the PK. This has its advantage for syste – matic classification but has, without doubt, some disadvantages too. In may cases the modern use of identifiers for programs is of considerable advantage.

Some exceptions to its strict rules are also allowed in the PK.

II. 3) b) Variables of a Program

Classification into the four categories $\alpha - \delta$ facilitates the understanding of a program. In other program languages these differences are normally not so strongly emphazised, since random notations are allowed

there , instead of the PK characters V_i , Z_i , C_i , R_i . The role played by these values can frequently be understood only with the help of the program.

The values listed under α to δ correspond to the variables in common mathematical formulas.

II. 3) c) Range of Indices

In the PK the scope of the variables, intermediate values, etc. is limited to the program for which they are used. Such a simple and strict rule does not always exist in modern languages because of the use of a " block structure ". The difference between " local " and " global " parameters in modern languages has not been relevant form the start in the PK, since in the latter all mentioned parameters are local (e.g. FORTRAN or COBOL) .

II. 3) d) Marginal Data Extract (Randauszug)

In some way the " marginal data extract " corresponds to the parameter or specification list of a procedure or program module in modern programming languages.

The input values may be substituted by " global parameters " , if the associated program to which they belong is inserted into the frame of other programs.

The Data Extract itself is not part of the program , but only coordinated to it. By this convention, the " program " of the PK is confined to the instructions for the calculating process. This concept differs from the modern concept which includes the declarations as unexecutable statements in the term " program " .

II. 3) e) Assignment Statement and the Assignment Symbol

The symbol ⇒ of the PK corresponds to the symbol : = in modern programming languages . The rules are essentially the same for both symbols with the difference, however , that the symbol ⇒ must point to a following result, while the symbol : = must follow the result which then stands left of it.

The representation of the PK corresponds to the course of the actions during elaboration . On the left hand we have the old values, on the right hand the new ones.

The symbol : = has been chosen by mathematicians in accordance with the definition - symbol :: = of symbolic logic. Their concept is of advantage in that the calculated value is exposed and easily readable at the beginning of a line. But this difference between the two concepts is not very relevant.

II. 3) f) Subprograms

In principle, the PK does not distinguish between main-programs, subroutines, procedures, functions, etc. Any program can be used as a procedure.

The rule that any result of a program can be used as a function identifier is very useful. It allows the easy use of programs as subprograms within the frame of a main program.

Indenpendently of this general rule, it is possible to denote special subprograms (PZ,U...) in the PK . This has an effect only on the scope of the variables.

II. 3) g) Operation -Symbols, Function - Symbols

From a logical point of view there it is not necessary to treat operations and functions seperately from the programs. Also the PK provides the possibility of using the notations with which the mathematician is familiar.

II. 3) h) Remark

A special notation for " comment " is missing in the PK if it is compared to other algorithmic languages. Naturally, the following examples occasionally contain comments. However, since the PK is not designed to produce programs ready for complication, it is not necessary to mark the comments specially, in order to instruct the compiler to jump over them.

II. 4) Quasi - Fixed Programs

The introduction of the term " quasi - fixed programs " resulted from my concept at that time , of deve loping special machines for the composition of programs (Planfertigungsgeräte). In this sence I syste - matically investigated the possibility of program variations. No computers with essential facilities for varia tions of the course of the calculation existed then. The following chapter only is a typical example of merely theoretical desk work. Later, around 1955, the development had gone other ways.

II. 4) a) - e) I scarcely used the variations mentioned in these sections in later examples of programs

II. 4) e) Variable - Structure - Symbol

This section, interestingly, contain a remark about the storage of programs at the end of it. As already mentioned, this facility was self - evident for me. Nevertheless, the content of this section does not concern itself with the essential aspects of the problem .

II. 4) f) Variation of the Number of Components of a Structure

This variation of a program is, perhaps, the most important one. For this purpose, this facility is achieved by the introduction e.g. of the mode string in modern algorithmic languages.

II. 4) g) General Considerations about the Variation of Programs

The variations of programs mentioned in this chapter indeed furnish the PK with some of the characteristics of a calculus. In 1945, the term " algorithmic language " in its present meaning was not yet common. What I had in mind with " Rechenplanfertigung " in the PK is only hinted at there in principle. This holds true especially for the remark in section II. 4) b) on the coordination of structure - symbols to operation - symbols. To realize, that it would be necessary to fransform the formulas of the PK into a fixed sequence of symbols, in order to be able to perform symbolic calculations (see chapter 4 : Ope - rations with Algebraic Expressions) .

II. 5) Free Programs

As already mentioned, the " free programs " in the definition of the PK represent the common form of programs as they are used today. Fixed and quasi - fixed programs may be considered as a special form of free programs.

II. 5) a) Variable End Symbol

The Variable End Symbol is the simplest form of a branch instruction. It corresponds to the nowadays so called LEAVE or EXIT statements. It allows the univocal programming of a complicated calculation composed of many main programs and subprograms. In other algorithmic languages this function is achieved by conditional statements (IF, WHILE) or by GO TO - statements and labels.

The GO TO statements are often the source of errors and can even be always avoided. I assume that the PK Fin Symbol is more fool-proof.

It is also incorporated in the PK the method, now common of enclosing program parts between brackets to which the end symbol FIN refers. In some algorithmic languages symbols like BEGIN, END are instead of brackets to specify program parts. But their effect extends still further.

II. 5) b) Conditional Program Parts

This is again one of the most important types of conditional statements. In other algorithmic languages symbols like IF, THEN, ELSE, WHILE, FI are used for the same purpose. The PK knows only the simple condition, which is no alternative to THEN .... ELSE . Naturally , this is only a question of economy of representation. The extended modern form certainly has some advantages.

II. 5) c) Variable Indices

The Variables Indices represent another important form of variable instruction. Today, we also call this facility " transformation or computation of addresses " .

The broken lines are more a picture - writing than a representation by symbol. Remember, that the from of the PK presented here is primarily designed for human understanding in contrast to the Computer. In order to feed compilers etc. the PK must be converted into a sequence of symbols.

The representation used in the PK allows easy distinction between the structures of components and their indices.

II. 5) e) Computation of Programs

See the comment referring to II. 4)

II. 6) Repetitive Programs

Repetitive Programs of the PK correspond to the WHILE DO- statements in modern languages. In the PK the different possibilities are represented by W - instructions W_0 to W_6 . The following modern notations for example correspond to the formulas W_0 to W_5 :

Equivalent Constructions in ALGOL 68 :

$W_0(n)$ [P] ≡ TO n DO P

$W_1(n)$ [P (i)] ≡ FOR i FROM 0 TO n–1 DO P (i)

$W_2(n)$ [P (i)] ≡ FOR i FROM n–1 BY –1 TO 0 DO P (i)

$W_3(n,m)$ [P (i)] ≡ FOR i FROM n TO m–1 DO P (i)

$W_4(n,m)$ [P (i)] ≡ FOR i FROM n BY –1 TO m+1 DO P (i)

$W_5(n,m)$ [P (i)] ≡

IF n ≤ m THEN

FOR i FROM n TO m-1 DO P (i)

ELSE

FOR i FROM n BY -1 TO m+1 DO P (i)

FI

The modern instructions permit specification of the upper and the lower bounds and the step width. This is undoubtedly more elegant.

From a modern point of view the rules W_5 and W_6 seem to be too specialised.

II. 7) Programs of the Predicate Calculus

Mathematical logic, in particular the propositional calculus assisted me most effectively in the development of a switching algebra. This caused me to investigate , especially the Predicate Calculus. I soon recognized that this calculus in particular was suited to the simplification of programming tasks. This finding has been discovered again recently and has led to the use of this calculus for the description of data and structures of programs e.g. VDL (Vienna Definition Language) . The Predicate Calculus had already been applied inadvertently for data processing before the introduction of computers. This holds true for nearly all punched card operations. The selection of special cards from a stack corresponds to the operator.

$$\hat{x}\ P\ (x)$$

" Those x for which the predicate proves to be true " .

In a similar way, the other operators of the predicate calculus can effectively be used too. This becomes obvious in the development of chess programs especially. In them we very often have to formulate criteria as follows : " There is a square , occupied by a white officer which is under attack by a black piece " . To formulate such a task accurately we need the operators of the predicate calculus.

1) The " variable of third stage " of the PK does not correspont directly to the Referenceconcept for instance in Algol 68.

2) The first character R forms together with □ a special symbol R□ . The second R means „ result ".

3) The statement $R_o = V_o \in V_1$ is a comment.

4) In the following program the relation R□ is dependant on the predicate P (Z_o). Therefore the list Z_1 must be investigated from the beginning after the evaluation of μx .

5) The chapter " Computer Processable Programs " was not written any more .

II. 8) Miscellaneous

II. 8) a) The operators $\wedge R$, $\vee R$, ϵR , ΠR . In the abbreviated notation the initial statements

$+ \Rightarrow \underset{o}{R}$, $- \Rightarrow \underset{o}{R}$, $o \Rightarrow \underset{o}{R}$, $1 \Rightarrow \underset{o}{R}$ are omitted.

II. 8) b) Representation of Powers

Common mathematical representations are not only sequences of symbols, but rather a composition of symbols, picture - writing, and topological positioning of symbols (e.g. indexing and raising to powers). The form used in the PK is similar to picture - writing as the symbol $\urcorner$ proves. Modern algorithmic languages use the symbol $\uparrow$ or ** .

II. 8) c) The term " list " used in the PK corresponds best to the term " array " or " row " in modern algorithmic languages. Something different is now understood by " list - processing " , namely a method for the linking of different elements through additional information (pointer) about the adress of the following or the preceeding element.
Instead of

$$\begin{array}{l|l} v & V \Rightarrow \underset{o}{Z} \\ s & \square \times \sigma \end{array}$$

the following expression is possible too :

$$\begin{array}{l|l} v & V \Rightarrow \mu \underset{o}{Z} \\ s & \square \times \sigma \end{array}$$

II. 8) d) The Statement Symbol

The Statement Symbol " $\vdash$ " I took from Mathematical Logic. It is not included in the syntax of the PK , but is also relevant only for comments. It demonstrates, however, my original intention, of extending the PK beyond the borders of an algorithmic language.

Comments of Chapter 2

The general programs of chapter 2 represent a fundamental library. In modern algorithmic languages some of these programs can be realized only through special operators. Here the advantage of the PK is obvious : that all its data structures are developed systematically on the basis of Y - N –Values .

The following classifications of the programs are classified according to the structure of the input - values and results.

Normally, the programs are numbered sequentially. In some special cases names are added in form of a sequence of characters. As already mentioned, comments are not marked as such (e.g. P 1.18 , suppo - sition ...) .

P1.26, P1.27 The Formula $\underset{o}{V}+1$ is based on the supposition, that the operation addition belongs to the standard implementation.

P1.32, P1.33 The inclusion of the " signal " into the PK does not require special rules as for instance PL/1 does. The normal result R_o is only supplemented by an additional result R_1.

P1.36 Operators like Ub often belong to the standard statements of algorithmic languages today. The corresponding program then takes the following form :

```
bool a_i
int a_i
if a then c := b
     else c := o fi
```

There are already more elegant representations available for the same program , e.g.

c := (a | b | o) (Algol 68)

P1.39 Here we suppose that $2^{\neg} n$ is already defined respectevely that this operation belongs to the standard implementation . It would certainly be possible in this simple case to set up a program for the evaluation of m as a function of n .

The expression $\vdash R_o$ 1.9 (R_o) represents a comment.

P1.40 The series of bits v_o is shifted to the right in the direction of the higher indices, as many steps as v_1 as a binary number indicates.

The program is incorrect. R_1 means „ overflow " and is omitted in the marginal data extract.

P1.104 In the extended language of Algol 68, for instance, the operator Maj can be represented as follows :

Ro : = (Vo ≤ V1 | Vo | V1)

Maj, Min, Ord are semantic notations.

P2.8 Note that the letter R has unfortunately been used both as a symbol for Result and for Relation as well.

IV. Calculus of Lists

The use of the term " list " was already partly explained on page 34 The following programs are often presented both in implicit and in explicit form. The implicit form is a comment, the explicit form is the proper program.

P3.11 This program can also be presented without μ- operator according to P3.10.

P3.12 V_2 in this case is regarded as a binary number.

P3.26 Note that only the explicit form is the proper program.

P3.27 The symbol ⇔ means equal in the terminology of the Theory of Sets. V_o, Z_o and R_o are in fact not three different storage-capacity requiring lists. For such a program the introduction of a " transit - value T " replacing the three lists would be convenient. Then the statements :

$V_o \Rightarrow Z_o$ and $Z_o \Rightarrow R_o$

could be omitted in the program.

The program P3.27 is incorrect. For the condition

V K S	Z 1 σ	<	Z o ϵ σ

must be introduced an intermidiate value $\frac{Z}{2}$. (ϵ may be influenced by the first statement) .

P3.30 Note the different meaning of $\hat{x}$ and $\hat{\hat{x}}$ (chapter 1, PK page 45). The current value ϵ_i is part of the structure index; consequently it must be lifted to the main line.

P3.66, P3.67 In the PK the selection of specified sections of a list is performed by special programs. This is achieved in Algol 68 through special declarations , e.g. " trimming " . For the construction of a compiler the method of the PK may be more advantageneous, since it does not request special provisions to be made.

P3.68 Here the PK is applied as a true calculus to transform the implicit form into the explicit form.

P3.71 With the PK I tried to develop a " list - calculus ", distinct from the " theory of sets " in that it met the needs for the solution of practical calculating problems. It is an essential fact, that the elements of a list , which represents a set, are always stored in sequence.

V. 1) a) Graphic Representation by an Arrow - Diagram

This kind of representation has been extended lately to the Theory of Graphs. For practical problems this theory is mostly covered by the Calculus of Relations., as stimulated by the PK.

P4.43 The program is incomplete. It includes circles spezified by P4.41

P4.48 - P4.52 The following developments in particular demonstrate the application of the PK as a calculus . Implicit and explicit expressions can be represented by the same syntax.

I hoped that the application of the Relation - Calculus to computers would be very effective (see for instance the graph - theory) As a civil engineer, I had in mind to use the Relation Calculus for the formal representation of static constructions such as Framework . Nowadays it is every - day routine work of civil engineers not only to use computers for numerical calculations but also for the organisational operations referring to the structure of the system. Nevertheless, modern programs were not developed as solidly on the foundation of general programs for relations as is the PK.

The programs are partly incorrect Referring to μ' see page 45

45

P4.52 Here the advantages of the Marginal Data - Extract can be recognized. All results $R_o \div R_6$ can be inserted into other programs as individual function - symbols also.

P4.52 The encircled numbers do not belong to the program proper, but are only hints for the textual description. The elaboration of the program is univocally determinated by the bracketing of sub - programs and the Fin - symbols.

During the textual formulation of the program P4.52 I realized that the pointers (i) were very helpful for an understanding of the programs. I had in mind to introduce this method of programs control into the basic syntax of the PK' This would have corresponded to the GO TO statements

(i)

and the use of program labels. But I hesitated to take this step since, at that time, I did not have a satisfactory overview of the possible consequences.

We have, in fact, learned meanswhile that the GO TO statement can be highly dangerous. In principle, it is not necessary since it can be replaced by other perhaps stronger rules as is done in the PK with the FIN statement. On the other hand, the GO TO instruction in many cases allows very elegant solutions. Therefore, we would hardly wish to be without it in today's programming (see also P9.18), but this alone does not lead to "structured programming" or "software engineering".

Comments of Chapter 3

Programs for Arithmetic Operations

An effective programming language must be applicable to the programming of the arithmetic operations down to the smallest detail. This conviction of mine was a result of my experiences in the development with the aid of mathematical logic of circuit – diagrams for the computers Z1 to Z4.

I. 1) Scalars

In special cases irrational numbers can, nevertheless, be represented accurately by a finite series of symbols.

Hereafter where I use " computer V4 " I mean " computer Z4 ". Originally, I numbered my models V1 .... V4 (Versuchsgerät = test model 1 – 4). In order to avoid confusion with the " V – weapons ", I choose the designations Z1 to Z4 after the war.

II. General Introduction

The problem of the conversion of different representations of numbers is also a widely discussed subject in modern programming languages (e.g. in Algol 68).

The creation for the " overflow " is advantageous for the precise analysis of the calculating process. In the syntax and semantics of modern algorithmic languages, this problem is not specially dealt with, but usually regarded as part of the pragmatics of the language (realized by a hardware – interrupt). To me the reduction of programs in a operation which I called " melting " seemed to be very advantageous for the automation of programming. The program for a determinant of the order n for instance, the elements of which are constantly zero in some positions, can be reduced form the program for the complete matrix by melting, e.e. by omitting systematically all irrelevant operations, e.g. multiplications by zero.

Axiomatic Representation

Axiom systems for arithmetic operations only indirectly represent implicit solutions for numbers and operations with them. Indeed, axioms could generally only be satisfied by computers with an infinite number of digits. The PK, however, permits the design of programs, in which the number of positions (digits) n is a variable. With limes $n \to \infty$ the axioms can be satisfied.

P9.18 Extraction of the Square Root

I took special care with this operation. The computers Z1, Z3, Z4 were probably the only ones at that time, in which the square root operation was built directly into the hardware. This is possible in an especially elegant manner, if the binary system is used. The following development of the program P9.18 is omitted in the PK's English version. In case of special interest please refer to the German version.

P9.19 Extraction of the Cubic Root

I had elaborated a method for this operation analogous to the extraction of the square root; but it turned out to be too complicated to be built into the hardware of one of my computer models. Today, approximate methods are generally applied.

V. Operations with Positive Integer Decimal Numbers

V. 1) Structures of the numbers

At that time I was aware of the direct binary code for the conversion of decimal digits only. Later I was surprised at the very advantageneous solutions of Stibitz and Aiken.

VI. The Semi–Logarithmic Representation

This representation corresponds to the " floating point representation " of today. Originally I had in mind to construct computers with completely logarithmic representation. But this attempt failed because of the very complicated solutions for the addition. Consequently, I developed the " semi – logarithmic representation " which only in the integer part of the logarithm was used. The digits of the logarithm behind the point are replaced by a factor b, $1 \leq b < B$, (B = base of the numbersystem). In the modern floating point representation this factor lies mostly between 0.5 and 1.0 if the binary system is used.

The special values are only of limited practical importance. Since the model Z4 intentionally has no facilities for conditional instructions, I tried to incorporate as many of the variations as possible into the hardware. In this I was considerably supported by the tools which my algebra had given to me.

VI. 2) Operation with AΔ1

This section has been omitted in the English version of the PK. This example demonstrates that the PK is able to scope with very complicated calculations.

Comments of Chapter 4

Operations with Algebraic Expressions

Lastly, at this point there was the opportunity of dealing with normal numerical programs, e.g. solutions of linear equation systems, matrices etc. But to me these problems did not seem to be very urgent then, since it was obvious that numerical problems could be programmed with the PK without any difficulty. Moreover, in the isolated village Hinterstein / Allgäu I was more concerned with other problems. I wanted to investigate the then almost unexplored, more complex problems of calculating.

One of these problems seemed to be the algebraic handling of formulas called "symbolic calculations" today. Right form the start of my endeavours I had been attracted by the target of generalizing the term " calculating " beyond numerical problems.

I soun realized that I had entered a very wide and complex field. It was implossible to explore it in a short time to an extent which would suffice for practical use. As a consequence I concentrated on demonstrating that the PK was qualified for performing symbolic calculations.

In the following chapter only the rather simple propositional formulas are discussed and this only for some typical examples.

An extraction of this chapter was published in Archiv der Mathematik, Band 1, Heft 6 (1948/49), Verlag G. Braun GmbH Karlsruhe.

I. 1) g) Development of Programs

To me this problem appeared to be of the greatest importance. Instead of the term " compiler " I used the term " Rechenplanfertigung " (program production) . I intended to reach this goal by means of symbolic calculations. In my imagination it should have been possible to establish the rules for a compiler for the PK itself. Later the mathematicians went other ways. The first programming languages such as Cobol, Algol, and Fortran were not suited for this purpose. Only some recently developed languages, e.g. Algol 68, contain some of the required facilities.

As is well known, the development of compilers required tremendous investment in manpower, time and money. It would probably have been much more advantageneous to exploit the facilities of the PK form the very start of programming.

II. 1) Negation symbols have also to be regarded as operation symbol (monadic operation). Today it is a comman practice to take the symbol $\neg$ to avoid ambiguities.

II. 2) This is a recursive definition. The formal expression for Sa (x) demonstrates, that no " metalanguage " is necessary in the PK. For such definitions the " Backus–Naur–notation " for example is often used. This again shows that the PK incorporates a true calculus.

Page (175) The rules in terms of natural language are not precise enough in the sence of modern formal grammars.

Page (180) The value ϵ represents the " Klammerbilanz " (balance of brackets). Later Rutishauser introduced the term " Klammergebirge " (bracket mountain range) , for this.

Page (184) The order of " ranks " of the operation– symbols used here in contrast to the common habit of utilizing the term " Bindungsstärke " (priority of operations).

In programs, the higher rank corresponds to the block or program of a higher level. This was my reason to introduce the " order of ranks ". Besides of that it is comman today, that the symbol $\wedge$ has a higher priority than the symbol $\vee$.

Page (189) Remember that the errors in the original mansucript have intenionally not been eliminated in this publication.

Today, we know that the simplification of formulas is one of the most difficult problems of symbolic calculation. In most cases it is not possible to formulate an algorithm simple enough for practice. In such operations the experience of a mathematician plays an important role. Today we use the terms " Artificial Intelligence " and " General Problem " Solving" for the definition of such problems. The progress in this field is relatively slow.

Yet, I am convinced that these efforts will have great success in the future in spite of some disappointments. Perhaps it would be more effective to exploit first the facilities of the PK and thus obtain a solid foundation. Moreover, the computer hardware so far has not been designed to handle such problems conveniently. It should be worthwhile, therefore, to develop hardware which would be well adapted for the use of PK programs.

II. 4) Introduction of the " Machine Mode " (Maschinenform)

As already mentioned, I had in mind to extend the PK beyond its original scope, with the aim of compiling programs in particular. For this purpose the " machine mode " provides a good start. The later developments of algorithmic languages went this way, too.

But my relevant investigations began only sporadically and not systematically. Unfortunately, I had to stop this interesting work for lack of time when other problems began to occupy me completely.

Page (232) The form is analogous to the so called "Polish Notation". But that was unknown to me at the time. The form is not identical with the so called "Praefix–Notation".

Page (198) Pay attention that the symbol I (x) is used hear in another sense than on page 50

Comments of Chapter 5

Chess Programs

The formulation of schematic thinking processes associated with chess has been a task which stimulated from the very beginning of my investigations in general calculation about 1937. To scope with it, I learned to play chess. However, I never became a good player and, indeed, that was not the aim of my engagement.

I soon also realized, that the program for a good move is very difficult to design. First it was necessary to create a foundation for further proceedings. Therefore, I was hardly ever occupied with a " Theory of Games " of the kind developed by John von Neumann. By the way, I had in mind a method, which today is called " Minimax ", to evaluate the best move. But I realized very soon, that this method was of no practical importance, since even large electronic computers are not capable of performing the necessary volume of operations.

In 1945 , when I developed the PK in the isolated Village in the Alps, I did not even have a chessboard at my disposal. Moreover, I could not find anybody in the Village to play chess with me.

This situation may excuse some errors that I incurred, e.g. a false interpretation of the rule to strike " en passant " . But in the context of the PK this did not matter. I believe that the chess - programs alone prove the capability of the PK to handle very complex logical problems.

When I heard of progress in this field later, I was surprised that this had been possible without the use of a universal algorithmic language of the kind of the PK. As far as I know, chess programs were not established in one of the algorithmic languages than exsistant but directly in maschine code. I do imagine, that this is a rather complicated method and that a language like the PK could facilitate such programming most effectively. Certainly, the PK cannot furnish the key for the defeat of the world chess champion, like I had dreamed of in 1938.

Anyway, essential progress has been achieved recently. However in the final analysis we not only need more powerful hardware.

Undoubtedly, there are a lot of important goals to be aimed at other than that of playing chess with computers.

The Plankalkuel (PK)

(Programming Calculus)

Chapter 1

Introduction

Contents

Chapter 1

Introduction

Note : The reader should have a general knowledge of the author ' s

STATEMENTS OF A THEORY OF GENERAL CALCULATION *)

I. The Problem

The Plankalkuel ' s aim is the formal representation of any calculating program (Algorithm). These programs have to comply with the following conditions :

1) Input - variables and results must be clearly specified.

2) All statements for intermediate values and results must be given in explicit form so that the results can be calculated after the insertion of the input variables without additional tranformations that were not included in the program.

The programs can be of a great variety. In the PK " calculate " is defined as follows :

" To calculate new data form given data according to a program " .

In this book the whole field of calculating including " fixed programs " is investigated .

II. Establishment of the PK (Programming Calculus)

1) General Notations

Where numbers are used to distinguish between elements and to order them, they are written in the decimal system. In this, a notation is preferred, which allows a simple transformation of the numbers into the binary system, e.g. 0, 8, 16, 48, 64, 72, 128, ect. (all powers of 2.).

Subdivisions are marked by periods, e.g. 1.3, 2.13.1, for the specification of components.

In order to avoid the frequent repetition of the notation for a set of programs e.g. the chess – programs all sub - programs, data - structures etc. the symbol Δ is used as a substitute for the set of programs. P Δ. 13 for instance is the notation for program 13 of the set of programs Δ. If this special program is used outside of the specific set , then the symbol Δ has to be replaced by another notation.

Further, the relationless symbol □ is introduced for a blank position. In blank positions other suitable data my be inserted which are not necessarily related to other positions bearing the symbol □ . The □ symbols may, however, be filled in by digits and numbers in order to interrelate them within a program.

2) Data and their Representation

The occuring data can be of varying types, for example Y–N - Values, numbers, lists , etc. The term " algebraic dimension " was already introduced in the "STATEMENTS". (page 11)

*) see page 4

The distinction between the different data is characterized as follows :

a) Data Structures

Data Structure is a structure made up by component data wherein their meaning is not considered. There are data of fixed and of variable structure. Structure Symbol Si are attached to each notation. The formation of composed structures is then performed by means of " structure equations " , whereby already defined structures are used.

The symbol So is assigned to a single Y-N-Value. A sequence of Y-N-Values is given by the notation S1.n. The structure - equation reads :

$$S1.n. = n \times So$$

In this way it is always possible to analyse the composition of data , even if the structures of data are very complicated.

Another structure symbol is required for " indeterminate " . If, for instence we want to state, without specifying the structure of the elements, that a notation represents a list of n elements, then $n \times \sigma$ is its formal representation.

σ	then can be substituted by any structure symbol.
$\square \times \sigma$	is the general structure notation for a list (the length of the list and the structure of the elements are left open).
$\square \times 2\sigma$	represents the structure of a " list of pairs " wherein the structure of the elements is not fixed but is the same for the two elements of a pair.
$\square \times (\sigma,\tau)$	represents the structure of a list of pairs with different structures for the neighbouring elements of a pair.
$2 \times n \times \sigma$	is not a list of pairs but a " pair of lists " .

The notation $N (\underset{o}{V})$ specifies the number of elements N of a list $\underset{o}{V}$ (number of elements of the first level) .

b) Limitation of Data

Data Limitation is effective if the variability of a structure is not completely utilized for the representation of data. E.g. four binary digits are required to represent a decimal digit, but in this only ten of the sixteen possible variations of a series of four Y-N-Values are utilized. In such cases a limitation formula defines which types of data have to be considered. This limitation is represented by a B with an index.

If the Y-N-Values of a decimal digit are represented by a_0, a_1, a_2, a_3, the limitation formula reads as follows :

$$\overline{a_3} \vee \overline{a_1 \vee a_2}$$

This propositional expression is only valid for the binary numbers 0 to L00L.

Another limitation is the conversion of components into constants. It may often be convenient to supplement the elements of a list by their numbers within the list. So we produce a list of the

indices of the components of a data structure. This list is independent of the variations of the elements (see chess – programs page 217).

c) Types of Data

Notations of different meaning may be assigned to the same structures and limitations, e.g. the coordinates x and y. Generally , it is not necessary to distinguish between them. Should it be advantageous, however, then the type – symbols T_1 , T_2 ... are introduced.

d) Modes of Data (Angabeart)

A structure and possibly a limitation or type designation is designed to every mode of data. Independently of that, the components may have different meanings e.g. numbers in half – logarithmic notation.

All these notations can be combined by the mode – symbol Ai. If a notation is specified by a modes – symbol e.g. A10, then a separate notation of the structure is not necessary, since that is contained in the symbol A10. Mode – symbols can also be assigned to a group of different analogue structures. Numbers for instance can be represented by different structures (binary, decimal. etc.). We are then able to introduce a special symbol e.g. A8, see chapter 3, page 148 , which only states, that the data – mode refers to a number, without determination of its structure or type.

An indeterminate mode – symbol α can be now be introduced.

e) Components of Data

The parts of which data are composed are called components. The composition is represented by the structure notation :

S1.3 = 3XSo

This means that the structures S1.3 is composed of three components of the structure So (Y – N – Value). These components are designated by K_o, K_1, K_2.

The components may themselves be composed. The integer for decimal numbers is represented by the equation

nXS1.4 = nX4XSo

$K_o, K_1 \ldots K_{n-1}$ represent the single decimal digits. These are themselves composed as follows :

Ko.o Ko.1 Ko.2 Ko.3

K1.o K1.1 K1.2 K1.3

In this way the notation for the components can be subdivided to any level by the insertion of periods. The sequence of the symbols in the structure notation is relevant. For example :

nX4XSo ≠ 4XnXSo

The first notation represents a list of n elements in which each element is composed of four Y – N – Values. The second notation represents four series of n Y – N – Values.

In the first case K_0 has the structure S1.4, and in the second case S1.n.

In the foregoing examples the components were homogenuous. But this is not necessary. A sign may be attached to decimal numbers for instance. The structure is represented by the expression

(So, nXS1.4)

Now K_0 is of the structure So, and K_1 of the structure nX1.4. It is not possible to subdivide K_0 into components, but it is possible with K_1.

Note that, in a notation composed of n components, the highest index is n-1 since the numbering of the components starts with zero.

f) Representation of Data

The indeterminate representation of data is accomplished by letters with an added index, e.g. V_1 , Z_3 , etc. The composition of several data is indicated by bracketing and by setting commas between them.

(a, b) = c

In notations of this kind, the component with the lower index is always written first. Also when numbers are represented digit by digit in indeterminate form, the symbols for the lower powers are written first.

The determinate representation of Y-N-Values is normally achieved by the symbols "—" and " + " . For numbers digits can be used, (e.g. the decimal digits 0 to 9 or the binary digits 0 and L) . For this purpose, data have to be decomposed according to their structure definition. When numbers are represented by the symbols "—" and " + " the components with the lower index are again written first. However, when numbers are represented by digits, the digits with the highest index are written first, as is usual practice. Consequently, the following representations correspond to each other :

LL0 = – ++

L0L0 = –+–+

83 = L000,00LL= (++ ––, –––+)

This requires careful observation.

For the indeterminate form of an Y-N-Value the symbol " o " is introduced, but only in context with "–" and "+" . a = + – o for instance, means, that K_0 (a) is positive, K_1 (a) is negative , and that K_2 (a) may have any of the two values – or +.

The condition that a binary number x with four digits is even is then represented as follows :

x = – ooo

Only the component K_0 (x) is determinate here.

The symbol "0" can generally be taken as a substitute for a series of symbols " – " , even if this does not represent a number.

g) Representation in Lines

Seperate lines are used to characterize clearly the various notations which belong to the data, as for instance variable - index, component, structure etc.

First comes the main line for the representation of the expression in the usual form.

The next line serves to distinguish between the different variables by means of indices (V) .

Another line serves to specify the components K of the variables.

The expression K1 (V_3) (component 1 of variable 3), is written als follows :

```
                V
                3
                1
or K2.3 ( Z4 ) = Z
                 4
                 2.3
```

The last line serves for the specification of the structure, type, or mode of the data (S = index, or A = index) .

Example :

```
Z     variable Z4
4
2.3   component 2.3.
0     structure 0.
```

The structure - symbol refers to the component. The single lines are marked by preceding letters, V, K, S or A :

```
  | Z  ^  Z
V | 4     2
K | 2.3
S | 0     0
```

If a component is not derived from a variable, then the component index position is blank.

The preceding letter S can always be substituted by A ; inverse substitution is not permitted. In such a case , the indices already used for structures may not be used for modes as well. The structure - symbols So, S1.n for instance are identical with the symbols Ao, A1.n.

With line representation it is easily possible to distinghuish between the different modes and types of data. With it, it is not necessary to use different types of letters for different types of data, as is usually done, such as gothic letters for vectors. Such practice would not be appli - cable in the Plankalkuel, because of the large number of different data - types to be handled by it.

h) Constants

Constants with a special meaning can be assigned to the various types and structures. A constant is a special value of the set of possible variations of a variable of a given mode or structure. They are marked by a C with an index. The index normally refers to the structure or type.

i) Supplementation of Data by Numbering of Components

Any composed notation can be supplemented by a constant which is represented by a list of indices of the components. These values are specified by I ( ) . (Index of ....).

This is important, in order to learn where in a list an element with a special property is situated. (E.g. in chess problems : " The square occupied by the white king ") This problem can be solved by the supplementation of the notation by a list of the indices. But in practice it is not always necessary to perform this operation, since the symbol I ( ) is well defined.

j) Data of Fixed and Variable Structure

The whole set of all possible variations of a given mode, characterized by structure, type of limitation , forms the set of possible values . If the structure is fixed , then all elements of this set are of the same structure. If the structure of the data is variable, then the elements may have different structures. This is, the case with lists of varying length (e.g. in chess problems : List of the acting pieces) .

Mostly, the variability of the structure is confined by the number of its components. Then the structure – symbol is not a simple constant but is itself a variable. These variables are relevant in variable programs. The variable is then composed of the "proper variable" and the " structure - variable " . The structure - symbol here influences the course of the computation.

3) Fixed Programs

a) Notation of Programs

The programs are indentified by the letter P and an index (e.g. P1.10) .

Mostly, the index is composed, wherein the first component indentifies the programs group.

Programs may be of any size and may produce many results.

b) Variables of Programs

α) Input Values
These are identified by a V with an index.

β) Intermediate Values
These are of relvance only within the programs. They are identified by a Z with an index.

γ) Constants
Constants are part of the program. They are identified by a C and an index. One has to distinguish between general constants and special constants. The special constants are inden - tified by the characters Cp and an index. They are valid only for the specific programs.

δ) Results

Results are values which are calculated as functions of the input – values by means a program.

They are identified by a R and an index.

The identifiers V, Z, Cp, R are specifying variables and located in the second line (V = line).

c) Range of Indices

The range of the indices of the variables is confined to the specific program for which they are defined. The value Z_3, of program one, therefore, is not identical with the value Z_3 of program two. If results of a program are used in another program they then have to be identified by the index of the source – program. $R1.10\ (\ )_o$ for instance specifies the result with the index o of the program P1.10.

The numbering of the V – and R – values has to start form zero. The ranges of the indices of the program (Pi), and also the structure –, type –, and mode – symbols are valid within specific programs as well as outside them.

d) Marginal Data Extract

Marginal Data Extracts of a program represent the input and output values. Their structures and types are specified by marginal data extracts. On the left side of a marginal data extract an expression $R(V_o \ldots V_n)$, is located which lists all input values and on the right side is located a list of the results :

	R (V , V) ⇒ (R , R)			
V	o	1	o	1
S	1.n	1.n	1.n	o

This data extract means: The program has two input values V_o and V_1 of the structures S1.n and two results R_o and R_1 of the structure S1.n and So respectively.

Marginal Data extracts may be established for several programs in common. They are not part of the program, but only coordinated to it.

e) Program Equations and the Symbol " Results In "

A Program consist of a number of specific explicit program equations. On the left side an expression with input values or already defined intermediate values is located, on the right side an intermediate value or a result. It is also possible to calculate the various components of a result separately. The two sides of the equation are separated by the symbol " ⇒ ". This symbol may become identical with the symbol " = " (equal) or the symbol " ~ " (equivalence of propositions). For this we have the following rules :

α) Always the value on the right side of the symbol " ⇒ " is the one to be calculated. It never itself represents an operation.

β) If the symbols " = " or " ~ " appear within a program equation, then they represent operations

	V	=	V ⇒	Z
V	o		1	3
S	1.n		1.n	o

specifies, that Z_3 becomes positive if V_o equals V_1 .

γ) If the same values appear on both sides, then the values are not identical :

Z	+	1	⇒ Z
3			3

specifies, that the old value Z_3 increased by 1 results in the new value Z_3.

Such an equation can be replaced by a more precise equation as follows :

Z	+	1.	⇒ Z
3.i			3.i+1

Here the values on the two sides are distinguished by subindices.

δ) If the same value occurs in several equations repeatedly on the right side, then the last calculation of this value is good. Preceding values become invalid.

The items γ) and δ) correspond to the method of using storage cells repeatedly which the author already realized in his Computer V_4 .

The rule implies, that the sequence of program - equations may not be changed.

Concerning the use of brackets, the symbol " ⇒ " has the widest range (an exception is the symbol " ⇒ ", to which reference is made later) . Two program - equations standing side by side are separated by a vertical line.

f) Subprograms

Programs may be composed of subprograms, and these again may be composed of other sub - programs . In this way programs and subprograms can be nested manifoldly.

In principle, any program can be used as a subprogram. To define this, the results of the program which is to be used as subprogram are shown followed by brackets. Between these the variables V_o , V_1 , of the subprogram are substituted by those which are to be inserted into the sub - program within the frame of a main program. Thus we produce, an expression such as the following :

	R9.10(Z)	⇒	Z
V	o	o	1
S		1.n	1.n+1

which states that the result R_o of the program P9.10 calculated with the input-value Z_o results in Z_1 .

The term R_o9.10 ( ) is used as a function symbol with one blank position. The number of blanks is equal to the number of blanks in the corresponding program (in this instance P9.10) .

It is obvious that the structures of the variables must coincide with those of the program . In the example mentioned above the data extract reads :

	R(V)⇒(R , R)
V	o o 1
S	1.n 1.n+1 o

If the result R_o9.10 (Z_1) is also of interest, the two program equations can be combined as follows :

	R9.10(Z) ⇒ (Z , Z)
V	o 1 2
S	1.n 1.n+1 o

In this way any program can be used as a subprogram. The symbols for operations and functions also in some way represent subprograms.

If special programs are to be used as subprograms within a single main program only, then the following possibilities exist :

α) The subprogram is established as a normal program, the only difference being that the symbols " P " and " R " are replaced by " PZ " and " RZ " (analog to the intermediate values " Z ") . Indices 0, 1, 2 ... are attached to the symbols " PZ " and " RZ " to distinguish between different subprograms within a main program. The range of these indices is limited by the main program. Consequently, the program PZ1 of one main program is not identical with the subprogram PZ1 of another main program.

β) The subprogram is established as part of the main program. The notation of the variables of the main program are also valid within the subprogram. Accordingly, it is not necessary to change the notations.

Such programs are marked with a U and an index. They refer to a main program.

g) Operation Symbols, Function - Symbols

Operation symbols can be used in the usual way instead of the program symbols " P .... " or their result symbols " R .... " . This is convenient for the handling of programs of general importance, e.g. propositional operations, or arithmetic operations.

Corresponding to the predicate calculus the results of programs can also be identified by a series of characters, for example :

Pos(x) meaning " x is positive "

h) Remark

The programs discussed in section 3) are all fixed which means that the sequence of the operations is independent of the variation of the input values.

4) Quasi fixed Programs

a) Definition

Programs are termed quasifixed, if they allow some variations independent of the real variables. This means, that the variation of a program can be performed independently of the real calculation. The variation of the program results in a fixed program. Because of their properties such programs are called quasi-fixed.

The variation of the programs is a function of the " program variables " which consist of variable operation symbols, program symbols, structure symbols, etc. These " program - variables " in relation to the real variables represent another level of variation. Level 1 can be varied independently of level 2, but level 2 depends on level 1. It is possible to distinguish between the following cases :

b) Variable Operation Symbols

If several programs of the same structure exist, such as :

V	$\wedge$	V	$\Rightarrow$	R
o		1		1
V	$\vee$	V	$\Rightarrow$	R
o		1		o
V	$\rightarrow$	V	$\Rightarrow$	R
o		1		o

then it is possible to introduce a common variable operation symbol " ȯ " :

	V	ȯ	V	$\Rightarrow$	R
V	o		1		o
S	o		o		o

The symbol " ȯ " may be substituted by any symbol for dyadic operations with Y – N – Values, which represents propositional operations like $\wedge, \vee, \sim, \rightarrow$.

In this example the number of the operation symbols allowed results form the notation for the structure So of the variables $\overset{V}{\underset{o}{}}$ and $\overset{V}{\underset{1}{}}$.

If this number is to be limited to the operations for which the associative rule is good, then the best procedure is to list the operation - symbols allowed.

The data extract of such programs must also contain the operation - symbol as a variable.

	R	(	ȯ,	V	,	V	)	$\Rightarrow$	R
V				o		1			o
S				o		o			o

In order to specify the different levels of variation more brackets are used :

	(R (δ))		(V, V)	⇒	R	
V			o 1		o	
S			o o			

It is now clear that the pertaining program is primarely a function of δ , the result of which depends on V_o and V_1 .

In the above mentioned example no structure - symbol is assigned to the variable operation - symbol. The structure of operation symbols is normally S1.n. In complicated cases special de - finitions for the structures of operation - symbols have to be introduced. If there is more than one variable operation symbols are introduced to distinguish between them :

$$\delta_o , \delta_1 , \delta_i$$

These indices are inserted into the second line.

c) Variable Program Symbols

Variable subprograms in the same way as operation symbols can be established. A variable program - symbol "Π" is introduced.

d) Variable Negation Symbol

Frequently, programs differ only in that Y-N-Values appear negated or not negated. The two propositional operations, equivalence and disvalence, for instance can be expressed by the opera - tions, conjunction and disjunction , as follows :

$$(V \wedge V) \vee (\overline{V} \wedge \overline{V}) \Rightarrow R$$
$$o \quad 1 \quad o \quad 1 \quad o$$
$$(V \wedge \overline{V}) \vee (\overline{V} \wedge V) \Rightarrow R$$
$$o \quad 1 \quad o \quad 1 \quad o$$

These two programs can be combined by the introduction of a variable u of a higher level :

	((V ∧ (u ~ V) ∨ ((V ∧ (u ~ V))					⇒ R
V	o		1		o	1 o
S	o		o		o	o o

If u is substituted by " + " we then get the first formula if n is substituted by " – " then the second. The data extract for this program is :

	(R (u))	(V ,	V)	⇒	R
V		o	1		o
S	o	o	o		o

In this case the structure of u can be indentified by So.

A variable symbol V with index could be used instead of u . However, by taking u we make apparent that the variable is one of a higher level.

e) Variable Structure Symbols

In the " Statements ... " an example demonstrated that programs can assume different meanings by varying the algebraic dimension.

Determinant representation can be used advantageously for real numbers as well as for propositions. The two programs differ only with respect to structure symbols and operation symbols. The two variations have the following form :

α) Determinant of degree 2
for real numbers : $\Delta = \begin{vmatrix} V_0 & V_1 \\ V_2 & V_3 \end{vmatrix}$

```
  | R ( V, V, V, V ) ⇒ R
V |     o  1  2  3      o
A |     8  8  8  8      8

  | V X V – V X V ⇒ R
V | o   3   1   2   o
A | 8   8   8   8   8
```

The symbol A8 stands for " real number " .

β) Determinant of degree 2 for propositions :

```
  | R (  V, V, V, V ) ⇒ R
V |      o  1  2  3     o
S |      o  o  o  o     o

  | V ∨ V ∧ V ∨ V ⇒ R
V | o   3   1   2   o
S | o   o   o   o   o
```

These two programs can be combined by the introduction of two operation symbols " δ_o " and " δ_1 " and a variable mode - symbol " α ".

```
  | R ( α, δ, δ ))   ( V, V, V, V ) ⇒ R
V |          1  2       o  1  2  3      o
A |                     α  α  α  α      α
```

Table of variations :

α	δ_o	δ_1
So	∨	∧
A8	X	–

	(V δ V) δ (V δ V) ⇒ R
V	o o 3 1 1 o 2 o
S	α α α α α

f) Variation of the Number of Components of a Structure

Frequently, only the number of the components of the input values are varied. The programs then differ by a different number of repetitions of analog program parts.

The elementary case is that of a structure S1.n or of a list nxσ . The representation of the programs requires the assistance of repetitive programs. (This is reffered to later).

g) General Considerations about Variations of Programs

In the examples mentioned before, the variation of programs is effected by the insertion of variable symbols. Much more complicated variations are possible however .

For instance, in program P9.18 (extraction of a square root, referred to later) the number of positions of the result has to be evaluated before the program for the square root itself is cal - culated.

This type of program leads to the variable programs; Fixed and variable programs differ in that , variable programs use program variables as well as real variables.

5) Variable Programs

In variable programs the input variables influence the course of the calculation.

First, the program variables which were discussed in the paragraph covering the quasi fixed programs, such as variable operation - symbols, structure symbols, etc. can be functions of the real variables. It is possible that the type of an operation in a program - equation is only calculated during the course of a program. Such cases are analog to those in section 4) .

In addition , it is possible to distinguish between the following typical cases :

a) Variable End Symbol

There are programs, which can be terminated before being completely computed if the result is already clear after part of the program is executed, or , if a continuation of the compu - tation appears to be irrelevant . For instance, a disjunction with several components can be terminated as soon as one proposition turns positive and a conjunction as soon as one proposi - tion turns negative .

To indentify " end " the variable end - symbol Fin is introduced. It is put on the right side of a program equation. The expression to the left of the symbol ⇒ represents the criterion , that the computation may be terminated.

For instance :

```
V ∨ V ∨ V ⇒ R
o   1   2   o

V ⇒ Z | Z ⇒ Fin
o   o | o

Z ∨ V ⇒ Z | Z ⇒ Fin
o   1   o | o

Z ∨ V ⇒ R
o   2   o
```

This kind of representation is advantageous only if repetitive programs are applied, or, if the propositions are represented by complicated expressions.

Generally, the range of the symbol Fin covers the whole of the program within which it appears. If, however, it is indended to skip over only part of the program , then several program equations must be combined to form a part program which is marked by brackets. The range of the symbol FIN is then limited by the brackets.

Such program parts can be nested. The symbols Fin^1 , Fin^2 then determine the range to be limited by the first or by the subsequent enclosing brackets. This is especialy important when using repetitive programs.

Correspondingly, an expression following the symbol $\dot{\Rightarrow}$ is also considered as a part program with a fixed range (see section 5) b)) .

b) Conditional Program Parts

The computation of program parts can depend on conditions represented by an expression which depends on the variables. The condition and the conditional program part are separated by the symbol $\dot{\Rightarrow}$. The dot indicates that this symbol does not identify a propositional operation (implication) .

A simple example is the computation of Maj (V_o, V_1) (the larger of two values V_o and V_1) .

```
    | Maj  ( V,  V ) ⇒ R
V   |        o  1     o
K   |        8  8     8

V ⩾ V ⇒̇ ( V ⇒ R )
o   1     o   o

¬(V ⩾ V) ⇒̇ ( V ⇒ R )
o   1     1   o
```

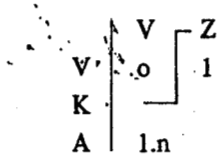
In this case either the first or the second program part is computed. Such conditional programs

may be multiply nested.

c) Variable Indices

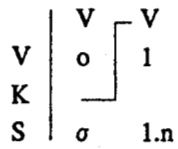
The indices of the symbols " V ", " Z ", " R ", " K ", etc. can be made to depend on variables. For instance, in the expression

the index of the component is variable. Now these variables can themselves be composed or supplemented by indices. The variables are wirtten on the main line and connected to their correct position by a broken line :



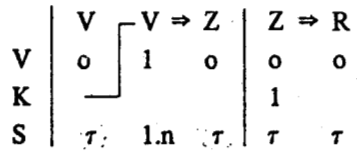
In this expression Z_1 indicates the component index of V_o . The structure notation for V_o then refers to its component ; the structure notation of Z_1 refers to Z_1 .

A typical application is the following : A function is represented by a list, in which every variable refers to a function value. This results in a list of pairs (pair list). If the variables are represented by the integers 0 to n−1, then they correspond to the indices of the list and then only the list of the function values is needed for the representation of the function values. If the structure on this list is specified by $nx\sigma$ and if V_1 is the variable, then the corresponding function value is represented by :



(see also : chess problems PΔ.62)

Not always does the overall index have to be varied. If, as described in the example above, the function values are composite and only the component 1 of it is wanted, then it is possible to define and intermediate value Z_o first as follows :



The two expressions can now be combined as follows :

	V	V.1	⇒	R
V	o	1		o
K				
S	τ	1.n		τ

The term $V.1_1$ is thought to be inserted into the position of the component index of V_o. The dot identifies the different levels of the components.

In the list V_o the element with the index V_1 shall be substituted by the element V_2 :

	R (V	, V	, V)	⇒	R	
V	o	1	2		o	
S	nXσ,	1.n	σ		nXσ	

	V ⇒ Z		V ⇒ Z		V	Z ⇒ R		
V	o	o	2	o	1	o	o	
K								
S	nXσ	nXσ	σ	σ	1.n	nXσ	nXσ	

The computation of R_o is performed via an intermediate value Z_o, which varies during the course of the computation (ref. rules for the symbol ⇒). Note that only the component V_1 of Z_o varies (for an example, see chess – problems PΔ.136).

The variation of the index can also be performed several times in succession (see chess – problems PΔ.202).

The demonstrated variation of the component index may also be performed with the variable – index. The two cases are analogical. The variation of the structure – index represents the case already discussed in section 4) e, f, that the structure – symbols depend on the real variations. This may refer to the basic method according to section 4) f (see arithmetic programs P9.72).

d) Data of Variable Size

Data of variable size are important for the list – calculus (see chapter 2, general programs). If list extract R_o has to be produced from the list V_o which contains only those components of V_o, which with a certain criterion comply, then the size of the list R_o is a function of V_o itself and not only of the size of V_o (in the latter case you have a quasi fixed program).

In such programs the structures are functions of the real input – variables and have to be determined for each computation (contrary to the quasi fixed programs, where this is determined independently of the real variables and only as a function of their structure).

It is not necessary, however, that the size of a list is always specified by the structure notation nXσ. Other size specifications are also possible. Two of these are of special importance :

α) The specification of lists by additional data. A part list such as V can be produced from a long list ticking additional data. In this case the ticked components only from the part list.

β) The setting of " begin " – and " end " – symbols provides a means of identifying the length of the list. These symbols may be blank positions. It has to be noted that in machine coding a special symbol has to be assigned to a blank position. In typewriters a special key is usually assigned to blanks.

Different programs solving the same problem may be represented differently depending on the kind of representation. By using various provisions such as repetitive programs and the μ function (reference to it will be made later) it is possible to avoid the differences and essentially to standardize the programs.

e) Computation of Programs

That the real variables of a program are of different levels, so that the variation of the program can be separately established as a function of the programmvariables only has already been discussed. ~~(see 4)~~): (see 4a, page 54)

In general however, this procedure can also depend on real variables.

If u_o , $u_1 \ldots u_n$ are the program variables of a quasi fixed program, and represent variable operation - symbols or structuresymbols and others, then the expression for the computation of a quasi fixed program is the following :

$$F (u_o, u_1, u_2) \Rightarrow P$$

P is composed of a series of program equations in which the real results are defined as functions of the real variables V_o , V_1 ... . The whole program then is represented as follows :

$$F (u, u, u \ldots) \Rightarrow P$$
$$\quad o\ 1\ 2$$
$$P$$

This states that the compiled program is computed after the computation of the program is completed. Here also, only parts of P or only program - variables may be defined by the expression F ( ) .

The program for $\sqrt[2]{V_o}$ takes the following form (see programs for arithmetic operations)

Marginal data Extract

	R (V) ⇒ R
V	o o
S	1.n 1.m

Marginal data extract with program - variables

	R (n, V) ⇒ (m, R)
V	o o
S	1.n 1.m

Program :

$$F(n) \Rightarrow m$$
$$(R(m,n))\ (V) \Rightarrow R$$

First m is evaluated as a function of n, then $\sqrt[2]{\frac{V}{o}}$ may be evaluated by a quasi fixed program.

6) Repetitive Programs

Repetitive Programs are programs which are executed several times in succession, wherein the number of repetitions in special cases may also be zero or one only. Generally, a program equation or a series of them is solved, as soon as the listed instructions are executed. The non - formulated rule exists wherein you proceed to the next equation after execution of the current one. Consequently, the sequence of the equations on paper must correspond to the programmed sequence of their solution.

The rule for repetitive programs is that the start of the program is returned to until the signal " Fin " terminates the repetitions.

This is only meaningful, if the instructions of the repetitive program are subject to variations which result form the repetitive computations themselves. Very often these variations serve to process a list for the computations. Normally, this is accomplished by changing a component - index i to i+1. Repetitive programs, therefore, generally operate with control variables i, ϵ , which control the variations of the repretitive executions of a program. They are called " variation - values " of a repretitive program.

The repetitive program must contain an instruction for the termination of the program execution as well as the instruction for the variations. This may , have to happen either after a predetermined number of repetitions is completed or as soon as a set of values is worked off. To terminate the symbol Fin, but of the second degree Fin^2 , has to be used. This is necessary since the entire computation consists of a sequence of equal repetitions of the program. The single symbol Fin within a repetitive program would only cause the termination of the variation currently running without returning to the start of the repetitive program.
Repetitive programs are put in brackets and marked by a pre-set symbol W.
Such a W- program then takes the following general form :

$$W \begin{bmatrix} F \rightarrow P \\ \overline{F} \rightarrow Fin^2 \end{bmatrix}$$

F represents a propositional expression which is a function of the variation – values and of the variables of the program as well .
P is the real repetitive program. It contains the program including the instruction for its variation. If " F " is positive then P is executed , if " F " is negative then the whole process of repetitions is terminated. For simplification, the expression $\overline{F} \Rightarrow Fin^2$ is left out and it is understood that it must always be regarded as a complement to a W - program. A W- program then takes the

form :

$$W\,[\ F \Rightarrow P\ ]$$

This can also be composed of several expressions :

$$W \begin{bmatrix} F_o & \Rightarrow P_o \\ F_1 & \Rightarrow P_1 \\ \vdots & \\ F_n & \Rightarrow P_n \end{bmatrix}$$

In such cases , the end symbol must be supplemented by an expression :

$$\bar{F}_o \wedge \bar{F}_1 \ \ldots \wedge \bar{F}_n \Rightarrow Fin^2$$

which indicates, that the process is terminated if the condition for its execution is positive for none of the program parts. Besides this general W - instruction, some other special ones are introduced for frequently recurring cases.

First a program is investigated in which a control variable i or ϵ runs through a series of numbers. Wo (n) denotes n - repetitions of a numbers.
For instance : Raising to a power :

$$V_o^{V_1} \Rightarrow R_o$$

	R (V, V) ⇒ R
V	o 1 o
A	8 9 8

A8 = real number

A9 = pos. integer number

$$1 \Rightarrow Z \ \Big|\ Wo\ (V) \ \begin{bmatrix} Z & \times & V & \Rightarrow & Z \\ o & & o & & o \end{bmatrix} \Big|\ Z \Rightarrow R$$

If $V_1 = o$ then the W - program is not executed. The result R_o is 1.

If $V_1 = 1$ then the program is executed once. The result R_o is V_o, etc.

W1 (n) . This instruction is used, if the program varies in value form o to n-1, e.g. if an operation has to be performed with each component of a list.

Example : General negation (see chapter 2) .
Negation of all elements in a series of Y - N - Values

$$\begin{array}{c|l} & R(V) \Rightarrow R \\ V & \quad o \quad\quad o \\ S & \quad 1.n \quad\quad 1.n \end{array}$$

$$\begin{array}{c|l} & W1(n) \quad \left[\begin{array}{cc} \overline{V} \Rightarrow R \\ o \quad o \\ i \quad i \\ o \quad o \end{array}\right] \\ V & \\ K & \\ S & \end{array}$$

Wo	(n) [P]	o ⇒ ϵ | W[ϵ < n ⇒ [P | ϵ + 1 ⇒ ϵ]]
W1	(n) [P(i)]	o ⇒ i | W[i < n ⇒ [P(i) | i + 1 ⇒ i]]
W2	(n) [P(i)]	n−1 ⇒ i | W[i ⩾ o ⇒ [P(i) | i − 1 ⇒ i]]
W3	(n,m) [P(i)] (n ⩽ m)	n ⇒ i | W[i < m ⇒ [P(i) | i + 1 ⇒ i]]
W4	(n,m) [P(i)] (n ⩾ m)	n−1 ⇒ i | W[i ⩾ m ⇒ [P(i) | i − 1 ⇒ i]]
W5	(n,m) [P(i)]	$n \Rightarrow i \mid W\left[i \neq m \Rightarrow \left[\begin{array}{l}[P(i) \\ m > n \Rightarrow (i+1 \Rightarrow i) \\ m < n \Rightarrow (i-1 \Rightarrow i)\end{array}\right]\right]$

W2 (n) corresponds to W1 (n) but with the difference that the control variable i runs from n−1 to o .

This W–program is important for instance for operation with algebraic expressions which are re - presented by a series of symbols. With these reverse programs execution frequently has to be applied.

The limits of the control variables in W1 and W2 are selcted in such a way that the number of elements of composed data (list) can be substituted for " n " . Then the variation runs from 0 to n-1 .

If, for instance general negation has to be evaluated using data of variable structure , then the formula may be written as follows :

$$W1(N(v)) \quad \left[\begin{array}{cc} \overline{v} \Rightarrow R \\ o \quad o \\ i \quad i \end{array}\right]$$

W3 (n,m) to W5 (n,m) are programs in which the variation runs from n inclusive to m exclusive and always begins with n . The limits are m ⩾ n for W3, m ⩽ n for W4. W5 is applicable in both cases.

W–programs can also be used to from a seris of values. The series of the number 0 to n-1 is the result of the following program :

$$\begin{array}{l|l|l|l} & 0 \Rightarrow R & W1(n) & \left[\begin{array}{ll} R+1 \Rightarrow & R \\ o & o \\ i & i+1 \end{array}\right] \\ V & \quad o & & \\ K & \quad o & & \end{array}$$

W–programs can be multiply nested. The correlation of the symbols W and the program – variables has then to be indicated by indices. For instance P3.3 : the examination of a list for recurring elements.

Each element has to be compared with every other element.

$$\begin{array}{l|l} & R(v) \Rightarrow R \\ V & \quad o \quad o \\ S & \quad n \quad o \end{array}$$

$$+ \Rightarrow Z$$
$$o$$

$$\begin{array}{l|l|l|l} & W1(n) & W3\ (i+1,n) & \left[\begin{array}{llll} v & i \neq v & i \wedge Z \Rightarrow & Z \\ o & o \quad o & 1 \quad o & o \\ \sigma & \sigma & o & o \end{array}\right] \\ V & o & & \\ K & & & \\ S & & & \end{array}$$

$$\begin{array}{ll} Z \Rightarrow & R \\ o & o \end{array}$$

The intermediate value Z_o is used for the current development of the conjunction of all single conditions.

The first W–program takes the form

$$\begin{array}{l} W1(n) \\ o \end{array}$$

The index 1 on the main line specifies the kind of W–program. The index 0 on the V–line refers to the attached control variable written i_o . The first W–program runs form 0 to n-1; consequently through all components of V_o .

The second W–program is of the W2 kind with i_1 indices attached, but it runs from $i_o + 1$ to $n-1$. From this follows, that the lower limit is a function of the variable i_o of the first W–program. In each run, one component of V_o is compared with all other components following it, thus duplicate operations and comparisons of elements with themselves are avoided.

Taking such numberings of W–programs and their corresponding indices into occount the general instruction for W1 reads as follows :

$$\begin{array}{l|l|l|l} & W1(n) & \left[\begin{array}{l} P(i) \\ j \end{array}\right] \equiv & \left[\begin{array}{ll} o \Rightarrow & i \\ & j \end{array}\right. \left| \begin{array}{l} W\left[\begin{array}{l} i < n \\ j \end{array}\right. \Rightarrow \left[\begin{array}{l} P(i) \\ j \end{array}\right. \end{array}\right. \left| \begin{array}{ll} i+1 \Rightarrow & i \\ j & j \end{array}\right]\Bigg] \\ V & j & & \end{array}$$

Corresponding notations can be established with other W–programs.

W6 (n,m) An especially advantageous form of a W–program is the following:
The first element is to be selected out of a list Z_o . Then Z_o is to be newly formed after the selected element has been left out. Thus the list Z_o is being limited through other operations or supplemented or newly established entirely.
We write :

$$\begin{array}{l|l} & W6 \\ V & \\ K & \end{array} \left[\begin{array}{c|c|c} \begin{array}{cc} Z \Rightarrow Z & \\ 0 \quad 1 & \\ 0 & \end{array} & P(Z) & R(Z,\square) \Rightarrow Z \\ & 1 & 0 \quad 0 \end{array}\right]$$

This is realized by the following program (" $\in$ " see).

$$\begin{array}{l|l} & W \\ V & \\ K & \end{array} \left[\begin{array}{c|c} N(Z) \neq 0 \rightarrow & \left[\begin{array}{c|c} \begin{array}{cc} Z \Rightarrow Z & \\ 0 \quad 1 & \\ 0 & \end{array} & \hat{x}\,(x \in Z \wedge I(x) \neq 0 \Rightarrow Z \\ & 0 \quad 0 \\ P(Z) & R(Z,\square) \Rightarrow \hat{z} \\ 1 & 0 \quad 0 \end{array}\right] \end{array}\right]$$

The W–program consists of three parts:

a) Computation of $\underset{1}{Z}$ and its removal form $\underset{o}{Z}$.

b) Program to be computed with $\underset{1}{Z}$.

c) New formation of the list $\underset{o}{Z}$ as a function of the former $\underset{o}{Z}$ and other variables.

The program is repeated until the list $\underset{o}{Z}$ is used up (for example see chapter 2, P3.9) .

7) Programs for the Predicate Calculus

In mathematical logic certain operation symbols play an important role : The " All " – and " Existence " operators , as well as the operators " that one which " , " those which " , " the next " .
Now will be demonstrated, how these operators can be represented in the Plankalkuel and what importance they deserve.

a) The " All " – and " Existence – Operators "

The expression (x) R (x) specifies that the predicate R is valid for all x. " All x " identifies the number of elements, the insertion of which into the predicate R is meaningful. This definition can also be adapted to the Plankalkuel. The number of values to be substituted for x is then the number of data characterized by the structure or mode symbols of x. In the expression

$$\begin{array}{l|l} & (x)R(x) \\ V & \\ S & 1.n \quad 1.n \end{array}$$

all data with the structure 1.n can be inserted for x.

In the expression :

	(x)	R (x)
V		
A	9.10	

Bitte eingeben! 144 oder 146 ?

only the positive integer decimal numbers can be inserted for x (see X) . 143

Generally, the following rule has to be assumed : in the case of a restriction – formula B (see 46) , only the values defined by this formula can be inserted . This restriction is already cotained in the term A9.10 (see 148).

To establish the " All " operator , a first expression is required to specify that x belongs to a certain data – type :

$$A(x) = \alpha \quad \text{respectively} \quad A(x) = \sigma$$

Herein " α " and " σ " are random mode – symbols or structure – symbols. An expression is then required to denote the list of all values which satisfy the modes α or structure σ :

$$L(\alpha), \text{ then } L(\sigma) \text{ respectively .}$$

If σ = So then the list $L(\sigma)$ consists of values " – " and " + ". If σ = 1.n the series of integer binary numbers from 0 to n-1 results. The program for the formation of this list has already been established (see page 65) . In a similar way it is possible to establish the program for the evaluation of a randomly composed structure σ for the list $L(\sigma)$. A restriction – formula can be applied to achieve a corresponding restriction of the list.

In principle, it is possible to establish a general program for the formation of the list $L(\sigma)$ for any mode of data .

This will not be realized presently, for two reasons :

α) This program would be a function of the structure– notation of a structure notation, and thus a variable of third degree, so to speak (see 61) . This would render the problem very complicated.

β) Generally, it is not necessary to develop a list of all possible cases.

If the list $L(\alpha)$ is exceptionally required for a certain α , then the special program for the development of $L(\alpha)$ has to be established. Thus , in general, the expression

$$(x)\, R\square(x) \Rightarrow R \qquad 2)$$

can be represented by the program :

	L (A (x)) ⇒ Z			+ ⇒ Z		W1 (N (Z))		R□ (Z) ∧ Z ⇒ Z				Z ⇒ R					
V			o		1		o			o	1	1		1	o		
K										i							
S		σ	□Xσ		o			o	σ	o	o			o	o		

The same cosiderations are valid for the "Existence" operator

$$(Ex)\, R\square(x) \Rightarrow R$$

at least one element with the property $R\square$ exists. The corresponding program is the following:

	$L(A(x)) \Rightarrow$	Z	$-\Rightarrow$	Z	$Wl(N(Z))$	$R\square(Z) \wedge Z \Rightarrow Z$	$Z \Rightarrow R$
V		o		1	o	o 1 1	1 o
K						i	
S	σ	$\square X\sigma$		o		o σ o o	o o

Now, in practice, it is not necessary to extend the variation of x over the whole range of the structure. Mostly the expressions are of the form, " All elements of the list V_o are of the property $R\square$ ", or, "in the list V_o an element of the property $R\square$ exists".

We formulate these sentences differently :

"For all x is true : If x is an element of V_o then it is also of the property $R\square$", or
"An x exists for which is true : It is an element of the list V_o and it is of the property V_o"
Symbolic notation is now required to express that x is an element of the list V_o. In relation to the theory of sets it is stated :

	$x \in V$
V	o
S	σ $\square X\sigma$

which is read : "x is an element of $\underset{o}{V}$".

The symbol "$\in$" binds expressions more closely together than any other symbol does.
The corresponding program reads :

	$R(V, V) \Rightarrow R$	$R = V \in V$	3)
V	0 1 0	0 0 1	
S	σ n$\times\sigma$ 0		

	$-\Rightarrow Z$	$Wl(n)$	$V = V \vee Z \Rightarrow Z$	$Z \Rightarrow R$
V	0		1 0 0 0	0 0
K			i	
S	0		σ σ 0 0	0 0

The expression for the operators can now be expressed as follows :

$$(x)(x \in \underset{o}{V} \rightarrow R\square(x))$$

$$(Ex)(x \in \underset{o}{V} \wedge R\square(x))$$

In these expressions it is no longer necessary to extend the variation over the whole range of the structure. For the " All " operator the following is true :

Either x is not an element of V_o, in which case the implication is true in any case ($a \rightarrow b$ eq $\bar{a} \vee b$).

These cases need therefore not be investigated.

Or x is an element of V_o, then $R\square(x)$ must be true .

For the " Existence " operator the following is true :

Either x is not an element of V_o then the expression within brackets is false in any case.

Or x is an element of V_o, then $R\square(x)$ must be true.

Therefore, it is sufficient in any case to confine the variation to the elements of V_o.

In consequence , the resulting expression reads :

$$(x)(x \in V_o \rightarrow R\square(x)) \Rightarrow R_o$$

and the corresponding program :

	+ ⇒ Z	W1 (N (V))	[R□ (V) ∧ Z ⇒ Z]	Z ⇒ R
V	0	0	0 0 0	0 0
K	[illegible]	[illegible]	1 [illegible]	[illegible]

In mathematical logic the following statement exists :

$$(x)F(x) \rightarrow (Ex)F(x)$$

This is true, because empty sets are excluded. In the PK representation developed above, the set of elements x is restricted to the list V_o, and consequently the logic statement is <u>not</u> true in all cases :

$$(x)(x \in V_0 \rightarrow R\square(x)) \rightarrow (Ex)(x \in V_0 \wedge R\square(x))$$

Reason : the set of elements of V_o may be empty. The corresponding statement of the PK reads :

$$(x)(x \in V_0 \rightarrow R\square(x)) \rightarrow (Ex)(x \in V_0 \rightarrow R\square(x))$$

respectively :

$$(x)(x \in V_0 \wedge R\square(x)) \rightarrow (Ex)(x \in V_0 \wedge R\square(x))$$

These two statements are generally true .

For the expression :

$$(Ex)(x \in V_0 \wedge R\square(x)) \Rightarrow R_0$$

the corresponding program reads :

$$\begin{array}{l|l|l|l|l} & -\Rightarrow Z & W1(N(V)) & R\square\ (V) \vee Z \Rightarrow Z & Z \Rightarrow R \\ V & 0 & 0 & 0\quad 0\quad 0 & 0\quad 0 \\ K & & & i & \end{array}$$

b) The Operator. " Those which "

In mathematical logic the expression

$$\hat{x}\, R\square(x)$$

specifies the set of elements for which the predicate $R\square$ is true .

To arrive at a general formula, the variation of x has to be extended over the whole range of its structure.
We then obtain the expression

$$\hat{x}\, R\square(x) \Rightarrow \underset{o}{R}$$

and the corresponding program :

$$\begin{array}{l|l|l} & L(A(x)) \Rightarrow Z & 0 \Rightarrow \epsilon \\ V & 0 & \\ S & \sigma\quad \square X\sigma & \end{array}$$

$$\begin{array}{l|l|l|l} & W1(N(Z)) & R\square(Z) \rightarrow Z \Rightarrow R & \epsilon + 1 \Rightarrow \epsilon \\ V & 0 & 0\quad 0 & \\ K & & i\quad \epsilon & \\ S & \square X\sigma & \sigma\quad \sigma & \end{array}$$

From the list $L(A(x))$, which contains all variations of the structure σ, those of the property R are extracted. The value ϵ serves for the current numbering of the elements of the result.

The problems occuring in practice are mostly of the following type :

" Form a list of those elements of the list $\underset{o}{V}$, which are of the property $R\square$ ".

Correspondingly, this can be written :

$$\hat{x}\,(x \in \underset{o}{V} \wedge R\square(x))$$

Now we can say : If x is not an element of $\underset{o}{V}$, then the expression within brackets is false in any case. If x is an element of $\underset{o}{V}$, then $R\square(x)$ must be true. Again, only the elements of $\underset{o}{V}$ have to be investigated.

But then, the following difference exists : if several elements in the list $\underset{o}{V}$ are equal to each other, then they have to be listed, but only once. It is therefore sufficient to investigate all elements of the list $\underset{o}{V}$ in regard to the property $R\square$, but elements already extracted in addition.

This is accomplished through a value Z_o which represents the extraction list. In the beginning the list Z_o equals the empty set which is denoted by the symbol " $\emptyset$ ".

For the expression :

$$\hat{x}\ (x \in V_o \wedge R\square(x)) \Rightarrow R_o$$

the corresponding program reads :

$$0 \Rightarrow \epsilon \qquad \emptyset \Rightarrow Z_0$$

V	W1	(N(V))	[	R□(V)	∧	$\overline{V \in Z}$	→	[V	⇒	Z	∉ +1	⇒ ε]]
K		0		0		0 0		0		0		
S		□Xσ		i		i		i		ε		
				σ		σ		□Xσ		σ		

V	Z	⇒	R
K	0		0
S	□Xσ		□Xσ

The expression :

$$\hat{x}\quad (x \in V_o)$$

then represents the list of all extracted elements of V_o, in which each element is listed only once.

The expression :

$$V_o = \hat{x}\,(x \in V_o)$$

therefore, specifies, that V_o does not contain multiple elements. Very often a list of all elements of the list V_o of the property $R\square$ is required, in which the elements are listed as frequently as they are extracted.

(For example : out of the list of all instructions in a machine–ready program only the instructions for operations have to be extracted. They represent the extract list for the control of the arithmetic unit.) .

Now the symbol $\hat{\hat{x}}$ is introduced and the following expression formed :

$$\hat{\hat{x}}\,(x \in V_0 \wedge R\square(x)) \Rightarrow R_0$$

the corresponding program reads :

$$\begin{array}{l} \quad 0 \Rightarrow \epsilon \\ \left|\begin{array}{l} W1\,(N(\,V\,)) \\ \quad\;\; 0 \end{array}\right. \left[\begin{array}{l} R\square\,(\,V\,) \rightarrow \left[\begin{array}{ll|l} V \Rightarrow R & & \epsilon + 1 \Rightarrow \epsilon \\ 0 & 0 & \\ i & \epsilon & \end{array}\right. \\ \quad\;\; 0 \\ \quad\;\; i \end{array}\right] \\ \begin{array}{l} V \\ K \end{array} \end{array}$$

It has to be mentioned that in both cases $(\hat{x}, \hat{\hat{x}})$, the sequence of the members of the extracted list corresponds to that of the input–list V_o.

In the PK the operators have, therefore, a somewhat different meaning as compared to mathematical logic, where they only define sets. In the PK they are expressions for a series of values with a determined sequence of these elements.

It is generally true that :

$$\hat{\hat{x}}\,(x \in V)_{\sigma} = V_o$$

The meaning of $\hat{x}$ and $\hat{\hat{x}}$ is demonstrated by an example : Given is a list V_o consisting of a series of numbers.

Ger (x) means " x is an even number ".
V_o = (0, 3, 5, 4, 3, 3, 6, 12, 6, 4)

Then it is true, that

$$\hat{x}\,(\,x \in V_o\,) = (\,0, 3, 5, 4, 6, 12\,)$$

$$\hat{x}\,(\,x \in V_o \wedge \mathrm{Ger}\,(x)) = (0, 4, 6, 12)$$

$$\hat{\hat{x}}\,(\,x \in V_o \wedge \mathrm{Ger}\,(x)) = (0, 4, 6, 12, 6, 4)$$

c) <u>The Operator " That One which "</u>

In mathematical logic the expression

$$x'\,R\square\,(x)$$

states, " that element for which the predicate $R\square$ is true ".

It is a condition for the application of this operator, that one and precisely that particular element of the specified property does exist. By variation over the whole range of the structure of x the following program is produced :

$$\begin{array}{l} \quad L(A(x)) \Rightarrow Z \\ \quad\quad\quad\;\; o \\ W1\,(N\,(Z)) \left[\begin{array}{l} R\square\,(Z) \rightarrow \left[\begin{array}{ll|l} Z \Rightarrow R & & Fin^3 \\ o & o & \\ i & & \end{array}\right. \\ \quad\;\; o \\ \quad\;\; i \end{array}\right] \\ \quad o \end{array}$$

As soon as an element of the property $R\square$ appears, this results in R and the end symbol for the whole process terminates it. The end – symbol in a W – program is Fin^2 if the whole repititive process is to be terminated. But, since the end symbol stands to the right of the symbol $\rightarrow$ in a closed program, it must be increased to the third degree (Fin^3). For this operator too, the expressions mostly take the following form :

$$\overset{\prime}{x}\,(\underset{o}{x} \in V \wedge R\square(x)) \Rightarrow \underset{o}{R}$$

the corresponding program reads :

$$\begin{array}{l|l} V & W1\,(N\,(V)) \\ K & \underset{o}{} \end{array} \left[R\square(v) \rightarrow \left[\begin{array}{ll} v & \Rightarrow R \\ o & o \\ i & \end{array} \middle| \; Fin^3 \right] \right]$$

d) The operator " The Next One "

The operator

$$\mu x\, R\square(x)$$

was introduced into logic by Hilbert, (Hilbert, Bernays, " Grundlagen der Math. 1. Band " page 395). It specifies: " The next element of the property $R\square$; if this does not exist then the expression equals zero " .

We will take over this operator in somewhat different form. In the PK, the problem of systematically investigating a set of values is mostly encountered. In this case, the list is investigated for elements with the property $R\square$, until it is exhausted. If there is no such element, then it is meaningless to set $x = o$ since this would lead to errors. An example out of the chess - programs shows this :

There is the " field - occupation " $\underset{o}{V}$ (see 216). First , out of V_o the list V_1 containing all squares occupied by white pieces is extracted. Then from theses squares, i.e. from the list V_1 all those squares are extracted which are under attack by black :

$$\underset{o}{Agr\,(V)}$$

Wanted, therefore, is the list :

$$\hat{x}\,(\underset{1}{x \in V} \wedge Agr\,(x))$$

However, immediately after the extraction of each of the respective squares, they must serve as input – values for another program , namely one investigating the freedom of movement. This is performed in that the next element of the property Agr is always extracted from the list V_o . Then the program P is executed with this element. For the first element we can establish :

$$\mu x\,(\underset{0}{x \in V} \wedge Agr\,(x)) \Rightarrow \underset{0}{Z} \;\Big|\; \underset{0}{P(Z)}$$

If no such element exists, then a substitution of x by zero would produce a false result. For this would mean, that the square (0,0) (the square al) is occupied by a white piece attacked by black. Differently from Hilbert we therefore define :

$$\mu x \, R\square(x)$$

as the next element of the property $R\square$. If this does not exist, the program is terminated by the end symbol. If in a repitive program the values $xR\square(x)$ are formed successively, then the values already extracted are discarded. In this way it is possible to investigate a list systematically. We then obtain an expression taking the form :

$$W\left[\begin{array}{lll} \mu x\ (x\in & V & \wedge\ R\square(x))\Rightarrow Z \\ & 0 & 0 \\ & m\times\sigma & \sigma \end{array}\ \middle|\ \begin{array}{l} P(Z) \\ 0 \\ \sigma \end{array}\right]$$

This expression can be substituted by the following program : [4)]

$$\begin{array}{l|l|l|l} & W1(N(V)) & \left[\begin{array}{ll} (V,-)\Rightarrow & Z \\ 0 & 1 \\ & i \\ & (\sigma,0) \end{array}\right] & \left|\begin{array}{l} +\Rightarrow Z \\ 2 \\ \\ 0 \end{array}\right. \\ V & 0 & & \\ K & i & & \\ S & \sigma & & \end{array}$$

$$\begin{array}{l|l} & W\left[\begin{array}{lll} (Ex)\left[\begin{array}{lll} x & \in Z & \wedge\ \overline{X} \\ & 1 & \\ & & 1 \\ (\sigma,0) & m\times(\sigma,0) & 0 \end{array}\right. & \wedge Z & \Rightarrow \\ & & 2 & \\ & & & \\ & & 0 & \end{array}\right. \\ V & \\ K & \\ S & \end{array}$$

$$\begin{array}{l|l} & W(N(V))\left[\begin{array}{lll} -\Rightarrow & Z & \left|\begin{array}{lll} Z & \wedge\ R\square(Z)\Rightarrow \\ 1 & & 1 \\ i.1 & & i.0 \\ 0 & & \sigma \end{array}\right. \\ & 2 & \\ & & \\ & 0 & \end{array}\right. \\ V & 0 \\ K & m\times\sigma \\ S & \end{array}$$

$$\begin{array}{l|l} & \left[\begin{array}{lll} Z & \Rightarrow Z & \left|\begin{array}{lll} +\Rightarrow Z & \left|\begin{array}{lll} +\Rightarrow Z & P(Z) & FIN^3 \\ 2 & 0 & \end{array}\right. \\ 1 & & \end{array}\right. \\ 1 & 0 & \\ i.0 & & \\ \sigma & \sigma & \end{array}\right. \\ V & \\ K & \\ S & \end{array}$$

In this program an auxilitary list Z_1 is first produced from the list V_o by supplementing each element by a Y–N–Value, which indicates whether this element has already been investigated for a transfer to the list Z_o . Consequently, these supplementary data are all negative at the start of the operation.

The real program then consists of a main W – program and a subordinated W–program. The first one serves the repitive formation of Z_o and is repeated as long as the following conditions are fulfilled :

α) In the list Z_1 elements must exist that have not yet been investigated.

β) If repetitions other than the first are under way, then the preceding evaluation must have resulted in a Z_o . Otherwise the repetition of the same investigation is meaningless. This is

specified by the auxiliary value Z_2 which has therefore to be positive to start with.

The investigation of the next element of Z_o is performed by the subordinated W – program. In the list Z_1 those elements are investigated which have not yet been specified :

	$(\bar{Z})$
V	1
K	i.1

As soon as the predicate $R\square$ becomes true for such an element this is made equal to Z_o and specified in the list Z_1 :

	$+ \Rightarrow Z$
V	1
K	i.1

Further, Z_2 becomes positive and with Z_o the program P (Z) is computed. Then the subordinated W – program is terminated (Fin^3) .

Generally , $R\square$ as well as Z_1 are not influenced by P (Z_o). This implies that the list Z_1 remains the same in all repetitions, with the exception of the marking of the already extracted elements. Thus the criterion for the extraction of the elements also remains the same for all variations.

It is then not necessary to extend the investigation for every Z_o over the entire range of the list Z_1 in each repetition. Only the elements not yet investigated have to be considered. This corresponds to a systematic investigation of Z_1 and of V_o respetively.

First the following program can be established :

$$W1\,(N(V))_o \left[R\square(V)_{o\,i} \rightarrow \left[V \Rightarrow Z_{o\,o\,i} \mid P(Z)_o \right] \right]$$

P4.49, chapter 2),page129 gives an example in which this formula must be applied, since $R\square$ is currently varying.
There $R\square$ has the form :

	$x = Z$	$\vee$	$x = Z$	
V	1		1	
K	o		1	

Z_1 has a different value in each repetition. In order to stress the fact, the symbol μ was provided with a dash

e) The operator $\lambda(x)$.

The following is stated :
the operator $\lambda(x)$ corresponds to the operator $\mu(x)$ with the only difference that with it , the investigation of V_o starts with the last expression :

$$W\left[\begin{array}{c|c} \lambda x\,(x \in V \wedge R\square(x)) \Rightarrow Z & P(Z) \\ 0 \qquad\qquad\qquad 0 & 0 \end{array}\right]$$

corresponds the program:

$$\begin{array}{l} N(v) \Rightarrow \epsilon \\ \quad\; 0 \end{array}$$

$$W\left[\begin{array}{l} \epsilon \neq 0 \;\dot{\rightarrow}\; \left[\begin{array}{l} - \Rightarrow \begin{array}{c} Z \\ 1 \end{array} \\ W\left[\begin{array}{l} \epsilon \neq 0 \;\dot{\rightarrow}\; \left[\begin{array}{l} \begin{array}{c} R\square(V) \\ 0 \\ \epsilon \end{array} \;\dot{\rightarrow}\; \left[\begin{array}{c|c|c} \begin{array}{l} v \Rightarrow Z \\ 0 \quad 0 \\ \epsilon \end{array} & \begin{array}{c} + \Rightarrow Z \\ \quad\; 1 \end{array} & Fin^3 \end{array}\right] \\ \epsilon - 1 \Rightarrow \epsilon \end{array}\right] \end{array}\right] \\ \begin{array}{l} Z \;\dot{\rightarrow}\; P(Z) \\ 1 \qquad\;\; 0 \end{array} \end{array}\right] \end{array}\right]$$

Corresponding considerations can be applied to combinations of several λ – and μ– operators in one W–program. More about that in the chapter "machine–ready programs ". 5)

f) <u>μ –Operator on the right side of the symbol ⇒</u>

In the expression:

$$\begin{array}{l|l} & W\left[\begin{array}{l} F \Rightarrow \mu\; R \\ \qquad\quad\;\; o \end{array}\right] \\ V & \end{array}$$

the equation within brackets specifies, "F results in the next element of the result R_o ", which consists of a list of several successively evaluated elements.

The following expression substitutes the program :

$$\left.\begin{array}{l|c|c|c} & 0 \Rightarrow \epsilon & W\left[\begin{array}{l} F \Rightarrow R \\ \qquad\;\; o \\ \qquad\;\; \epsilon \end{array}\right] & \epsilon + 1 \Rightarrow \epsilon \\ V & & & \\ K & & & \end{array}\right]$$

g) <u>Designation of Variables and Intermediate Values</u>

In the expressions dealt with in paragraphs a) to f), x represents a bounded variable. If several of such bounded variables appear in one expression, then they have to be designated differently. If there are only two, then the symbols x and y can represent them advantageously. If there are more than two, then it is better to use the symbols x_o, x_1, x_2.

Similarily, the intermediate values Z_o, Z_1 ... have to be distinguished by sub–indices, or by a change of the designations in the case of interlinking of operators. This also applies to the variables of the program within which the operator appears. For more about that see chapter " machine–ready programs ". 5)

h) Implicit and Explicit Expressions with Operators of the Predicate – Calculus

The discussed expressions take one of the following forms:

$$(x)\ R\square\,(x) \Rightarrow R \qquad (0 \quad 0)$$

$$(x)\ (x \in V \rightarrow R\square\,(x)) \Rightarrow R \qquad (0 \quad 0)$$

$$(Ex)\ R\square\,(x) \Rightarrow R \qquad (0 \quad 0)$$

$$(Ex)\ (x \in V \wedge R\square\,(x)) \Rightarrow R \qquad (0 \quad 0)$$

$$\hat{x}\ R\square\,(x) \Rightarrow R \qquad (0 \quad 0)$$

$$\hat{x}\ (x \in V \wedge R\square\,(x)) \Rightarrow R \qquad (0 \quad 0)$$

$$\hat{\hat{x}}\ (x \in V \wedge R\square\,(x)) \Rightarrow R \qquad (0 \quad 0)$$

$$x'\ R\square\,(x) \Rightarrow R \qquad (0 \quad 0)$$

$$x'\ (x \in V \wedge R\square\,(x)) \Rightarrow R \qquad (0 \quad 0)$$

$$W\left[\mu x\ (x \in V \wedge R\square\,(x)) \Rightarrow Z \;\middle|\; P(Z)\right] \qquad (0 \quad 0 \quad 0)$$

$$W\left[\lambda x\ (x \in V \wedge R\square\,(x)) \Rightarrow Z \;\middle|\; P(Z)\right] \qquad (0 \quad 0 \quad 0)$$

At first glance the values $\underset{o}{R}$ and $\underset{o}{Z}$ seem to be represented explicitly in these expressions, because they are located on the right side of the symbol $\Rightarrow$.

The programs to be substituted for these expressions, indeed, in any case provide a systematic computing method. But often this systematic method is redundant and has then to be substituted by a more effectivw one..

The implicit expression:

$$x^2 + ax + b = o$$

for instance, can be transformed into the following explicit form:

$$x'\ (x^2 + ax + b = o) \Rightarrow \underset{o}{R}$$

In this case, the systematic process would request the systematic variation of x over all possible values, in order to investigate, if the proposition within the brackets is fulfilled. Whereas the well-known explicit solution is the following :

$$-a/2 + a^2/4 - b \Rightarrow \begin{matrix} R \\ 0 \\ 0 \end{matrix}$$

$$-a/2 - a^2/4 - b \Rightarrow \begin{matrix} R \\ o \\ 1 \end{matrix}$$

In statistics , however, expressions in the general form given above often already represent the explicit solution because a better method does not exist.

The systematic method corresponds to the sorting procedure with punched cards. The data on the cards correspond to the elements of the property $R\,\square$, several runs are necessary if the criterion is complicated. An interesting example is the following :

Chess-programs P Δ.32.
List of elements between V_o and V_1

$R\,\Delta 29\,(V_o, V_1, V_2)$ specifies : V_o is situated between V_1 and V_2.

Then can be stated :

$$\hat{x}\ [\ R\,\square\,\Delta 29\,(x,\ V_o,\ V_1)\ ] \Rightarrow R_o$$

In this case, the systematic procedure would consist of an investigation of all 64 squares of the chess-field to find out if the criterion $R\Delta 29\,(x, V_o, V_1)$ is fulfilled for them or not. The volume of this computation is still tolerable, but another method is much more efficient. Starting with V_o in the direction of the element V_1 the elements are generated. Then we obtain the list of the wanted elements in a direct unbroken sequence. In a corresponding way in PΔ34 those elements are generated which are in knight-relation to the member V_o .

This method of constructing the wanted values will be called " generating method " in contrast to the " systematic method " . The rules for the generating method have to be established specially in each case.

Therefore, it is not possible to decide easily whether an expression corresponding to that on page X is implicit or explicit. There are programs and various expressions equivalent in themselves available which reach the target in different ways at different expense.

8) Miscellaneous

a) The Operators $\wedge R$, $\vee R$, ΣR, πR

For the expressions :

	V	$\wedge$ V	$\wedge$ V	$\wedge$ V	$\Rightarrow$ R
V	0.0	0.1	0.i	0.n-1	0
S	0	0	0	0	0

	V	$\vee$ V	$\vee$ V	$\vee$ V	$\Rightarrow$ R
V	0.0	0.1	0.i	0.n-1	0
S	0	0	0	0	0

	V	+ V	+ V	+ V	$\Rightarrow$ R
V	0.0	0.1	0.i	0.n-1	0
A	8	8	8	8	8

	V	$\times$ V	$\times$ V	$\times$ V	$\Rightarrow$ R
V	0.0	0.1	0.i	0.n-1	0
A	8	8	8	8	8

the following programs can be established in accordance with the rules discussed hitherto :

	+ $\Rightarrow$ Z	W1(n)	V $\wedge$ Z $\Rightarrow$ Z	Z $\Rightarrow$ R
V	0		0.i 0 0	0 0
S	0		0 0 0	0 0

	– $\Rightarrow$ Z	W1(n)	V $\vee$ Z $\Rightarrow$ Z	Z $\Rightarrow$ R
V	0		0.i 0 0	0 0
S	0		0 0 0	0 0

	0 $\Rightarrow$ Z	W1(n)	V + Z $\Rightarrow$ Z	Z $\Rightarrow$ R
V	0		0.i 0 0	0 0
S	8		8 8 8	8 8

	1 $\Rightarrow$ Z	W1(n)	V $\times$ Z $\Rightarrow$ Z	Z $\Rightarrow$ R
V	0		0.i 0 0	0 0
A	8		8 8 8	8 8

For the foregoing programs the following abbreviated notation can be introduced :

W1 (n) (V $\Rightarrow \wedge$ R) 0.i 0 W1 (n) (V $\Rightarrow \vee$ R) 0.i 0

W1 (n) (V $\Rightarrow \Sigma$ R) 0.i 0 W1 (n) (V $\Rightarrow \Pi$ R) 0.i 0

The symbols $\wedge$, $\vee$, Σ , Π have to be pronounced :

conjunction element of ....
disjunction element of .....
addend element of ....
factor of ....

It is advantageous to employ the foregoing symbols if elements in a series of operations are generated successively. In this case it is not necessary for all of them to appear in a closed W–program. Rather expressions of the following form are also possible:

$$F_o \Rightarrow \wedge R_o$$
$$F_1 \Rightarrow \wedge R_o$$
$$\vdots$$
$$F_n \Rightarrow \wedge R_o$$

But in a program only one of the four operators $\wedge$, $\vee$, Σ , Π may be applied to the same value (see chapter 4, page).

b) Representation of Powers

To position all variables on the main line the values representing powers can also be positioned on the main line. The original position of the values can then be indicated by a broken line analogical to the representation of composed indices (see page 59) .

$$V^{V_1} = V^{V_0 \, V_1}$$

By this method any randomly complicated expressions can be represented as powers :

	V^{V_0}	V^{V_1}
V	1	2
K	1.3	0
A	8	9

c) Empty-List and Variable List with only One Element

All data composed of a series of elements of the same structure can be defined as lists ($S = m \times \sigma$, see chapter 2, 98) . If the number of elements of the list is variable, then the special case can occur, that this number becomes zero. Such a list is generally specified by the symbol $\emptyset$.
The expression :

$$\emptyset \Rightarrow Z_o \, \square \times \sigma$$

denotes, that the list Z_o represents an intermediate value which is empty at the start. This is the case if for instance the result of a program is a list, the elements of which are generated successively. At the beginning this growing list is empty. It may also happen, that such a list has only one element at the beginning. We then state :

	V	⇒	Z
V	o		o
S	σ		□Xσ

This specifies that V_o is the only element of the list Z_o. In this exceptional case, variables of different structure may appear on the two sides of the symbol ⇒ . This is permitted only in this case.

d) Statement Symbol

If an expression represents an identity, i.e. a generally true proposition , then the statement symbol is positioned before the expression.

$\vdash F$

Chapter 2

2. Korrektur

General Programs

Contents

I. Operations with Data of the Structure So (Yes-No-Value)

1) Operations with one Operand

Negation : $\overline{V}_o$ in the meaning of the calculus of propositions

2) Operations with two Operands

$V_o \; \delta \; V_1$. Values for δ : $\vee, \wedge, \rightarrow, \sim, \not\sim$.

in the meaning of the calculus of propositions .

Function table :

V	V	Operation				
o	1	$\vee$	$\wedge$	$\rightarrow$	$\sim$	$\not\sim$
–	–	–	–	+	+	–
–	+	+	–	–	–	+
+	–	+	–	+	–	+
+	+	+	+	+	+	–

II. Operations with Data of the Structure S1.n

$$S1.n = n \times So$$

1) Programs with one Input Variable of the Structure S1.n

Propositions on S1.n

	R (V)	⇒ R
V	o	o
S	1.n	o

P1.0 General Disjunction ~~$(\overline{V})$~~ $\overline{o}$ $\nabla(V)_o$

(At least one element is positive)

	(Ex)	(x ∈ V ∧ x)	⇒ R
V		o	o
S	o	1.n , o	o

Explicit form

	– ⇒ Z	W1 (N (V))	[Z ∨ V ⇒ Z]	Z ⇒ R
V	o	o	o o o	o o
K			i	
S	o	1.n	o o o	o o

Alternative representation

$$W1 \left[\begin{array}{c|c|c} (N (V)) & [V \Rightarrow VR] \\ o & o \; o \\ & i \\ 1.n & o \; o \end{array} \right]$$

P1.1 General Conjunction $\Omega_{o}^{(V)}$

(All elements are positive)

V	(x)	(x ∈ V → x)	⇒	R
		o		o
S	o	o 1.n o		o

Explicit form

V	+ ⇒ Z	W1	(N (V))		Z ∧ V ⇒ Z		Z ⇒	R
	o		o		o	o	o	o	o
K						i			
S	o		1.n			o		o	o

Alternative representation

W1	(N (V))		V ⇒	R
	o		o	o
			i	
			o	o

P1.4
P1.5
P1.6
P1.7

V	W1 (N (V) –1)		V δ V ⇒ Z			Ω_{o}^{Z}	⇒	R
	o		o	o	1	1		o
K			i	i+1	i			
S	1.n		o	o	o	1.n-1		o

values for δ :

program notation	δ	Meaning
P1.4	∨	of two neighbouring elements, at least one is positive
P1.5	~	All elements are equal to each other
P1.6	↯	All elements have alternating values
P1.7	→	Left of a symbol " – " , there is no symbol + "

P1.8 Symmetry :

```
o ⇒ i | n-1 ⇒ j | + ⇒ Z
                  o

   |W [ i < j ⇒ [ ( V ~ V ) ∧ Z ⇒ Z | i+1 ⇒ i [ j-1 ⇒ j ] ] ]
V  |  [          [   o   o     o   o |         [       ]   ]
K  |  [          [   i   j          |         [       ]   ]
S  |  [          [   o   o     o   o |         [       ]   ]

   Z ⇒ R
   o   o
```

P1.9 Just one element is positive

```
   | – ⇒ Z | W1 [ ( N ( V ) ) [ V ∧ Z ⇒ ∧ R | V ∨ Z ⇒ Z ] ] | Z ⇒ ∧ R
V  |     o |    [       o   [ o   o     o | o   o   o ] ] | o     o
K  |       |    [           [ i             | i         ] ] |
S  |     o |    [       1.n [ o   o     o | o   o   o ] ] | o     o
```

P1.9 Alternative Representation

```
     – ⇒ Z | + ⇒ Z
         o |     1

   | W1 ( N ( V )) [ Z ∧ ( V → Z̄ ) ⇒ Z | Z ∨ V ⇒ Z ] | Z ∧ Z ⇒ R
V  |               [ 1     o   o     1 | o   o   o ] | o   1   o
K  |               [       i           |     i     ] |
```

Examples for P1.0 to P1.9

V o	R1.o o	R1.1 o	R1.4 o	R1.5 o	R1.6 o	R1.7 o	R1.8 o	R1.9 o
–	–	–					+	–
+	+	+					+	+
––	–	–	–	+	–	+	+	–
–+	+	–	+	–	+	+	–	+
+–	+	–	+	–	+	–	–	+
++	+	+	+	+	–	+	+	–
+++	+	+	+	+	–	+	+	–
+–+	+	–	+	–	+	–	+	–
–––	–	–	–	+	–	–	+	–
++––	+	–	–	–	–	–	–	–
++–++	+	–	+	–	–	–	+	–
––+++	+	–	–	–	–	+	–	–
––+–––	+	–	–	–	–	–	–	+
+–––––	+	–	–	–	–	–	–	+
––––+	+	–	–	–	–	–	–	+
+–+–+	+	–	+	–	+	–	+	–

2) Programs with one Input Variable of the Structure S1.n and a Result of the Strukture S1.n

	R (V) ⇒ R
V	o o
S	1.n 1.n

P1.16 General Negation

$$W1\,(N\,(V)) \begin{bmatrix} \bar{V} \Rightarrow R \\ o \quad o \\ i \quad i \end{bmatrix}$$

Operation symbol: $\ominus$ V (o)

P1.17 Mirage

$$W2\,(N\,(V)) \begin{bmatrix} V \Rightarrow \mu\, R \\ o \quad o \\ i \end{bmatrix}$$

P1.18 Upward Implication

$$- \Rightarrow Z \;\Big|\; W1\,(N\,(V)) \begin{bmatrix} Z \vee V \Rightarrow Z \\ o \;\; o \;\; o \\ \quad i \end{bmatrix} \;\Big|\; \begin{matrix} Z \Rightarrow \mu R \\ o \quad o \end{matrix}$$

supposition:

(x) R1.7 (R1.18(x))

P1.19 Downward Implication

R1.17 (R1.18 (R1.17 (V)) ⇒ R1.19
o o o o

supposition :

(x) R1.7 (R1.17(R1.19(x))

P1.20 Identification of the First Positive Element from the Left

$$- \Rightarrow Z \;\Big|\; W1\,(N\,(V)) \begin{bmatrix} \bar{Z} \wedge V \Rightarrow R \\ o \;\; o \;\; o \\ \quad i \quad i \end{bmatrix} \;\Big|\; \begin{matrix} Z \vee V \Rightarrow Z \\ o \;\; o \;\; o \\ \quad i \end{matrix}$$

supposition:

$$(x) \begin{bmatrix} R1.9\,(R1.20\,(x)) \vee x = 0 \\ o \quad o \end{bmatrix}$$

P1.21 Identification of the First Positive Element from the Right

```
R1.17 (R1.20 (R1.17 ( V ))) ⇒ R
o      o      o      o      o
```

supposition:

```
(x) [R1.9 (R1.21 ( x)) ∨ x = 0]
    [o    .                   ]
```

P1.22 Shifting to the Right

```
  |  –⇒R | W1 ( N ( V ) –1) | V ⇒ R |
V |    o |         o       | o   o |
K |    o |                 | i   i+1 |
```

P1.23 Shifting to the Left

```
  |  –⇒R | W1 ( N ( V ) –1) | V ⇒ R |
V |    o |         o       | o   o |
K |  n–1 |                 | i+1 i |
```

P1.24 Circulation to the Right

```
  | V⇒R  | W1 ( N ( V ) –1) | V ⇒ R |
V | o  o |         o       | o   o |
K | n–1 o |                 | i   i+1 |
```

P1.25 Circulation to the Left

```
  | V⇒R  | W1 ( N ( V ) –1) | V ⇒ R |
V | o  o |         o       | o   o |
K | o  n–1 |                | i+1 i |
```

P1.26 Counting Forward

```
  | +⇒Z | W1 ( N ( V )) | V ↛ Z ⇒ R | V∧Z ⇒ Z |
V |   o |       o       | o   o   o | o o   o |
K |     |               | i       i | i       |
```

P1.27 Counting Backward

```
  | –⇒Z | W1 ( N ( V )) | V ~ Z ⇒ R | V∨Z ⇒ Z |
V |   o |       o       | o   o   o | o o   o |
K |     |               | i       i | i       |
```

Alternative Representation for P1.26, P1.27

P1.26 `V + 1 ⇒ R`
`o     o`

P1.27 `V – 1 ⇒ R`
`o     o`

Examples for P1.16 to P1.27

V o	R1.16	R1.17	R1.18	R1.19	R1.20	R1.21
–	+	–	–	–	–	–
+	–	+	+	+	+	+
––	++	––	––	––	––	––
–+	+–	+–	–+	++	–+	–+
+–	–+	–+	++	+–	+–	+–
++	––	++	++	++	+–	–+
————	++++	————	————	————	————	————
———+	+++–	+———	———+	++++	———+	———+
——+–	++–+	–+——	——++	+++–	——+–	——+–
–+–+	+–+–	+–+–	–+++	++++	–+——	———+
+———	–+++	———+	++++	+———	+———	+———
+–++	–+——	++–+	++++	++++	+———	———+
+++–	———+	–+++	++++	+++–	+———	——+–
++++	————	++++	++++	++++	+———	———+

V o	R1.22	R1.23	R1.24	R1.25	R1.26	R1.27
–	–	–	–	–	+	+
+	–	–	+	+	–	–
––	––	––	––	––	+–	++
–+	––	+–	+–	+–	++	+–
+–	–+	––	–+	–+	–+	––
++	–+	+–	++	++	––	–+
————	————	————	————	————	+———	++++
———+	————	——+–	+———	——+–	+——+	+++–
——+–	———+	–+——	———+	–+——	+–+–	++——
–+–+	——+–	+–+–	+–+–	+–+–	++–+	+——+
+———	–+——	————	–+——	———+	–+——	————
+–++	–+–+	–++–	++–+	–+++	–+++	——++
+++–	–+++	++——	–+++	++–+	———+	–++–
++++	–+++	+++–	++++	++++	————	–+++

3) Various Programs with an Input Variable of the Structure S1.n

P1.32 Counting forward with signal in case of overflow

	R (V)	⇒ (R ,	R)	
V	o	o	1	
S	1.n	1.n	o	

	+ ⇒ Z	W1	(N (V))	V↛	Z	⇒ R	V ∧ Z	⇒ Z		Z	⇒ R				
V	o		o	o	o	o	o	o	o		o	1			
K				i		i	i								

P1.33 Counting backward with signal in case of underflow

	R (V)	⇒ (R ,	R)	
V	o	o	1	
S	1.n	1.n	o	

	– ⇒ Z	W1	(N (V)	V ~ V	⇒ R		V ∨ Z	⇒ Z		$\overline{Z}$	⇒ R				
V	o		o	o	o	o	o	o	o	o	1				
K				i		i	i								

P1.36 Transfer controlled by V :

Ub	1

	Ub	(V ,	V)	⇒ R											
V		o	1	o											
S		1.n	o	1.n											

	V ↛ o	⇒ R		V ↛ V	⇒ R										
	1	o		1	o	o									

P1.37 Number of positive elements

	R (V)	⇒ R	m ⩽ n	
V	o	o		
S	1.n	1.m		

	o ⇒ Z	W1	(n)	V ↛	(Z + 1	⇒ Z)				Z	⇒ R				
V	o			o	o	o				o	o				
K				i											

P1.39 Assignement of data $\overset{R}{o}$ of the property R1.9 ($\overset{R}{o}$) to data of the structure S1.n , interpreted as binary number

	R (V)	⇒ R	2 ⌐ n	⇒ m	⊢ R1.9 (R)										
V	o	o			o										
S	1.n	1.m													

	o ⇒ Z	+ ⇒ Z	⌐ V		Z ⇒ R										
V	o	o	o		o	o									
K															
S	1.m	o	1.n												

Examples

V o	0	1	2	3	4	5	6	7	
––	+	–	–	–					0
–+	–	–	+	–					2
++–	–	–	–	+	–	–	–	–	3
+++	–	–	–	–	–	–	–	+	

P1.40 Shifting upward by a given number of positions with signal in case of overflow

	R (V , V) ⇒ (R , R)	m < n
V	o 1 o 1	
S	1.n 1.m 1.n o	

	V ⇒ Z V ⇒ Z
V	1 o o 1
S	1.m 1.m 1.n 1.n

	W [Z ≠ 0 → [– ⇒ Z | W1 (n–1) [Z ⇒ Z]]]
V	o 2 1 2
K	o i i+1
S	1.m 1.m o o o

	Z ⇒ Z | Z ⇒ VR | Z –1 ⇒ Z
V	2 1 | 1 1 | o o
K	n–1
S	1.n 1.n | o o | 1.m 1.m]]

	Z ⇒ R
V	1 o
S	1.n 1.n

P1.41 Shifting downward alanlogous to P1.40 (see P1.23 , page 87)

4) Propositions on two Data of the Structure S1.n

P1.64
P1.65 V
P1.66 S
P1.67
P1.68

	R (V, V) ⇒ R
	o 1 o
	1.n 1.n o

W1 (n) [V 0 V ⇒ ∧ R]
o 1 o
i i

Substitutions for o

	ŏ
P1.64	∨
P1.65	∧
P1.66	→
P1.67	↛
P1.68	~

For R1.68 (V_o, V_1) can be written :

$$V = V$$
$$o \quad 1$$

P1.72

	V ⩽ V ⇒ R
V	o 1 o
S	1.n 1.n o

	W1 (n)	V ⇒ Z	V ↛ V ⇒ Fin^2	$\bar{Z}$ ⇒ R
V		o o	o 1	o o
K		i	i i	
S		o o	o o	o o

P1.73

V ⩾ V

o 1 as P1.72 , but exchange V by V

1.n 1.n o 1

P1.74

	V < V ⇒ R
V	o 1 o
S	1.n 1.n o

	W1 (n)	V ⇒ Z	V ⇒ Z	V ↛ V ⇒ Fin^2	$\bar{Z}$ ∧ Z ⇒ R
V		o o	1 1	o 1	o 1 o
K		i i	i	i i	
S		o o	o o	o o	o o o

P1.75

V > V as P1.74, but exchange V by V

o 1 o 1

5) Operations with Data of the Structure S1.n

	R (V, V) ⇒ R
V	o 1 o
S	1.n 1.n 1.n

		W1 (n)	V	Ô	V ⇒ R
P1.96			o		1 o
P1.97	V				
P1.98	K		i		i i
P1.99	S		o		o o
P1.100					

	Ô	Op.–symbol
P1.96	∨	⩒
P1.97	∧	⩓
P1.98	→	⇉
P1.99	↛	⇏
P1.100	~	≈

P1.104 Maj (V, V) ⇒ R (The higher of the two values V and V)
o 1 o o 1
1.n 1.n 1.n

V	⩽	V ⇒ Z	$\bar{Z}$ → V ⇒ R	Z → V ⇒ R
o		1 o	o o o	o 1 o

P1.105 Min (V, V) ⇒ R (The smaller of the Values V and V
o 1 o o 1

V	⩽	V ⇒ Z	Z → V ⇒ R	$\bar{Z}$ → V ⇒ R
o		1 o	o o o	o 1 o

P1.106 Ordering of two data, the lower value first
Ord 0

	R (V , V) ⇒ (R , R)
V	o 1 o 1
S	1.n 1.n 1.n 1.n

Ord 0 (V , V)
o 1

V	⩽	V ⇒ Z	Z →	V ⇒ R	V ⇒ R
o		1 o	o	o o	1 1

			$\bar{Z}$ →	V ⇒ R	V ⇒ R
			o	1 o	o 1

P1.107 As P1.106 , the higher value first

V	⩾	V ⇒ Z	Z →	V ⇒ R	V ⇒ R
o		1 o	o	o o	1 1

			$\bar{Z}$ →	V ⇒ R	V ⇒ R
			o	1 o	o 1

6) Relations between 3 Data of the Structure S1.n

V, V, V interpreted as positive integer binary numbers.
o 1 2

	R (V , V , V) ⇒ R
V	o 1 2 o
S	1.n 1.n 1.n o

P1.128 V' lies between V and V
o 1 2

Zw (V, V, V)
o 1 2

(V < V ∧ V < V) ∨ (V > V ∧ V > V) ⇒ R
1 o o 2 1 o o 2 o

P1.129 V lies outside of V and V
o 1 2

(V < V ∧ V < V) ∨ (V > V ∧ V > V) ⇒ R
o 1 o 2 o 1 o 2 o

III. Programs with Pairs of Data

Structure of the input $2\times\sigma$, (σ, τ) respectively.
In the first case the structure of the front element is the same as that of the back element. In the second case the two elements of the pairs are of different structures. General structure –symbol of a pair:

$$S2 = (\sigma, \tau)$$

This is mostly substituted by special symbols

$$2\times\sigma;\ (\sigma, \tau)\ ;\ 2 \times S1.n$$

In the case of 2×S1.n, the operations of section II, 4, 5 can be applied to the singel elements of the pair.

1) General Programs

P2.1 One element of the first pair is equal to one element of the second pair (General Coherence)

	R (V , V) ⇒ R
V	o 1 o
S	$2\times\sigma$ $2\times\sigma$ o

	V = V ∨ V = V ∨ V = V ∨ V = V ⇒ R								
V	o	1	o	1	o	1	o	1	o
K	o	o	o	1	1	o	1	1	
S	σ	σ	σ	σ	σ	σ	σ	σ	o

P2.2 The front elements or the back elements are equal to each other (coherence with different structure of the front and back elements) .

R (V ,	V)	⇒ R
o	1	o
(σ, τ)	(σ, τ)	o

	V	= V	∨ V	= V	⇒ R
V	o	1	o	1	o
K	o	o	1	1	
S	σ	σ	τ	τ	o

P2.3 One pair is the mirror image of the other

	R (V ,	V)	⇒ R
V	o	1	o
S	2Xσ	2Xσ	o

	V	= V	∧ V	= V	⇒ V
V	o	1	o	1	o
K	o	1	1	o	
S	σ	σ	σ	σ	o

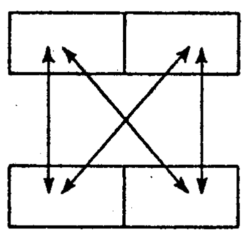
P2.4 The pairs are equal , or one pair is the mirror image of the other

	V	= V	∨ R2.3 (	V	, V)	⇒ R
V	o	1	o	o	1	o
S	2Xσ	2Xσ	o	2Xσ	2Xσ	o

P2.8 The realtion Rx is true, at least between one element of the front pair and one element of the back pair (General coherence by Rx) .

	(R (Rx))(V ,	V)	⇒ R
V	o	1	o
S	2Xσ	2Xσ	o

	Rx (V,	V) ∨	Rx (V,	V) ∨	Rx (V,	V) ∨	Rx (V,	V)	⇒ R
V	o	1	o	1	o	1	o	1	o
K	o	o	o	1	1	o	1	1	
S	σ	σ	σ	σ	σ	σ	σ	σ	o



P2.9 Linkage

One element of the front pair is equal to one element of the back pair ; the others are not equal.

	R (V , V) ⇒ R
V	o 1 o
S	2Xσ 2Xσ o

	V ≠ V ∧ V ≠ V ∧ [(V = V ∧ V ≠ V) ∨ (V = V ∧ V ≠ V)] ⇒ R
V	o o 1 1 o 1 o 1 o 1 o 1 o
K	o 1 o 1 o o 1 1 o 1 1 o o
V	∨ (V = V ∧ V ≠ V) ∨ (V = V ∧ V ≠ V)
	o 1 o 1 o 1 o 1
K	1 o o 1 1 1 o o

2) Relations between Pairs of the Structure 2XS1.n, interpreted as areas charcterized by binary numbers

a) Propositions on not ordered pairs

	R (V , V) ⇒ R
V	o 1 o
S	2X1.n 2X1.n o

P2.16 The elements of each pair are in themselves ordered :

	V ⩽ V ∧ V ⩽ V ⇒ R
V	o o 1 1 o
K	o 1 o 1
S	1.n 1.n 1.n 1.n o

P2.17 The pairs are ordered in relation to the first bound.

	R2.16 (V , V) ∧ V ≤ V ⇒ R
V	o 1 o 1 o
K	o o
S	1.n 1.n 1.n 1.n o

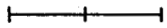
P2.18 Both areas are equal to zero :

	V = V ∧ V = V ⇒ R
V	o o 1 1 o
K	o 1 o 1 o

P2.19 The areas are separated

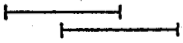
	Maj (V, V) < Min (V, V) ∨ Min (V, V) > Maj (V, V) ⇒ R
V	o o 1 1 o o 1 1 o
K	o 1 o 1 o 1 o 1 o

P2.20 The areas are adjacent



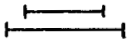
```
   | R2.9 ( V ,    V      ) ⇒ R
V  |      o      1        o
S  |      2X1.n   2X1.n    o
```

P2.21 The areas are overlapping



```
   | [R1.128 ( V,V,V )   R1.129 ( V,V,V ) ] ∨
V  | [o       o 1 1      o       o 1 1   ]
K  | [          o o 1               1 o 1 ]

   | [R1.128 ( V,V,V ) ∧ R1.129 ( V,V,V ) ] ⇒ R
V  | [o       o 1 1      o       o 1 1   ]   o
K  | [          1 o 1               o o 1 ]
```

P2.22 The first area lies within the second .



```
   | Zw ( V,V,V ) ∧ Zw1 ( V,V,V ) ⇒ R
V  |      o 1 1          o 1 1      o
K  |      o o 1            1 o 1
```

P2.23 One area lies within the other .

```
   | R2.22 ( V,    V      ) ∨ R2.22 ( V,    V      ) ⇒ R
V  | o       o    1          o       1    o       o
S  | o       2X1.n 2X1.n     o       2X1.n 2X1.n    o
```

P2.24 The areas have more than one point in common

```
   | V  ≠  V  ∧ R2.19 ( V,    V      ) ⇒ R
V  | o     1    o      o     1        o
S  | 2X1.n 2X1.n o      2X1.n 2X1.n    o
```

Alternative representation :

```
R2.21 ( V,V )   R2.23 ( V,V ) ⇒ R
o       o 1     o       o 1   o
```

b) Propositions on ordered pairs

Supposition :

```
   | R2.16 ( V      ) ∧ R2.16 ( V      )
V  |        o                   1
S  |        2X1.n                2X1.n
```

Marginal Values

```
   | R ( V,    V    ) ⇒ R
V  |     o    1      o
S  |     2X1.n 2X1.n  o
B  |     R2.16 R2.16
```

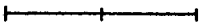
P2.32 The pairs are equal

	V = V	∧ V = V	⇒ R
V	o 1	o 1	o
K	o o	1 1	

P2.33 The areas are separated

	V < V	∨ V < V	⇒ R
V	o 1	1 o	o
K	1 o	1 o	

P2.34 The areas are neighbouring



	V = V	∨ V = V	⇒ R
V	o 1	1 o	o
K	1 o	1 o	

etc. corresponding to page 95-96

IV. Calculus of Lists

S3.m = m X σ

1) Quasi fixed Programs

The structure of the results are only functions of the number of elements of the input value, but not of their actual variation (see chapter 1, page 54)

a) Propositions on Lists

P3.0 Element of a list

	R (V, V)	⇒ R
V	o 1	o
S	σ mXσ	

abbreviated representation :

V ∈ V (see chapter 1, page 68)
o 1

	W1 (N (V))	[V = V ⇒ V R]
V	1	o 1 o
K		i

Marginal values for P3.1; P3.2

	R (V)	⇒ R
V	o	o
S	mXσ	o

P3.1 All elements are equal to each other

Implicit form

	(x) (x ∈ V → x = V) ⇒ R
V	o o o
K	o
S	σ σ mXσ σ o

Explicit form W1 (m) [V = V ⇒ ∧R]
o o o
o i

P3.2 All elements differ form each other

Implicit form

(x) (Ey) (x ∈ V ∧ y ∈ V ∧ I (x) ≠ I (y) → x ≠ y) ⇒ R
o o o

Explicit form

	W1 (m)	W3 (i+1,m)	V' i ≠ V i ⇒ ∧ R
V	o	1 o	o o o 1 o
K			
S			σ σ o

(see chapter 1, page 64)

Marginal values for P3.4 to P3.9

	(R (R□)) (V) ⇒ R
V	o o
S	mXσ o

P3.4 There is a pair of elements for which the relation R□ is true

Implicit form

(Ex) (Ey) (x ∈ V ∧ y ∈ V ∧ I (x) ≠ I(y) ∧ R□ (x,y)) ⇒ R
o o o

Explicit form

	W1 (m)	W1(m)	i ≠ i → R□ (V i, V i) ⇒ V R
V	o	1	o 1 o o o 1 o
K			
S			1.n 1.n σ σ o

P3.5 There is a pair of adjarent elements for which the relation R□ is true

Implicit form

(Ex) (Ey) (x ∈ V ∧ y ∈ V ∧ I (x) +1 = I (y) ∧ R□ (x,y)) ⇒ R
o o o

Explicit form

$$\begin{array}{l|l} V & W1\ (m-1) \\ K & \end{array} \left[\begin{array}{l} R\square\ (V,V\ ) \Rightarrow \vee\ R \\ \quad o\ \ o \qquad\quad o \\ \quad i\ \ i+1 \end{array} \right]$$

P3.6 For all adjarent elements the relation $R\square$ is true

$$\begin{array}{l|l} V & W1\ (m-1) \\ K & \end{array} \left[\begin{array}{l} R\square'(V,V\ ) \Rightarrow \wedge\ R \\ \quad o\ \ o \\ \quad i\ \ i+1 \end{array} \right]$$

P3.7 For every element, another one exists for which the relation $R\square$ is true

Implicit form

$$(x) \left[\begin{array}{l} x \in V \rightarrow (Ey) \\ \quad o \end{array} \left[\begin{array}{l} y \in V \wedge I\ (x) \neq I\ (y) \wedge R\square(x,y) \\ \quad o \end{array} \right] \right] \begin{array}{l} \Rightarrow\ R \\ o \end{array}$$

Explicit form

$$\begin{array}{l|l} V & W1\ (m) \\ K & o \end{array} \left[\begin{array}{l} - \Rightarrow Z \\ \quad o \end{array} \left[\begin{array}{l|l} W(m) & i \neq i+1 \rightarrow\ R\square\ \ (V\lceil i,\ \ V\lceil i+1\rceil \Rightarrow \vee Z \\ 1 & o\ \ 1 \qquad\quad o\rfloor o\ \ 1\rfloor 1 \qquad\quad o \end{array} \right] \right] \left| \begin{array}{l} Z \Rightarrow \wedge R \\ o \qquad o \end{array} \right]$$

P3.8 For every element and all elements following it the relation $R\square$ is true

Implicit form

$$(x)\ (y) \left[\begin{array}{l} x \in V \wedge y \in V \wedge I(x) < I(y) \rightarrow\ R\square(x,y) \\ \quad o \qquad o \end{array} \right]$$

Explicit form

$$\begin{array}{l|l} V & W1(m) \\ K & o \end{array} \left[\begin{array}{l} W3(i+1,m\ ) \\ 1 \quad o \end{array} \left[\begin{array}{l} R\square\ (V\lceil i,\ \ V\lceil i+1\ ) \Rightarrow\ R \\ \quad o\rfloor o\ \ o\rfloor 1 \qquad o \end{array} \right] \right]$$

P3.9 Coherence exists between all elements according to the relation $R\square$

Implicit representations by the predicate $R\square'\ (x,y)$

" Between x and y the relation $R\square$ is true directly or indirectly".

Recursive definition of $R\square'$:

$$R\square(x,y)\ \vee\ R\square(x,y) \rightarrow R\square'(x,y)$$

$$R\square'(x,y) \wedge (Ez) \left[R\square(y,z) \vee R\square(z,y) \right] \rightarrow R\square'(x,z)$$

$$(x)\ (y) \left[\begin{array}{l} x \in V \wedge y \in V \wedge I(x) \neq I(y) \rightarrow R\square'(x,y) \\ \quad o \qquad o \end{array} \right]$$

Explicit form :

```
      V ⇒ V
V  |  o   o
K  |  o
S  |  σ   □Xσ

      W6 [  Z ⇒ Z
V  |        o   1
K  |          o
S  |          σ   σ

V  |     LZ [ Z , x̂ [ x ∈ V ∧ ¬(x ∈ Z) ∧ R□(Z,x) ∨ R□(x,Z) ] ] ⇒ Z
K  |           o       o        o        1          1          o
S  |         □Xσ  σ    σ    mXσ      □Xσ      σ σ        σ σ    □Xσ

      (x) (x ∈ V → x ∈ Z) ⇒ R
V  |          o       o     o
K  |
S  |  σ   σ    mXσ σ   □Xσ   o
```

LZ(x,y) means:
Concatenation of the lists x and y (see page 109)

110

b) Operations with a List, which again produces a list

P3.10 Addition of an element

LZ New element last

```
      LZ (V , V) ⇒ R
V  |     o   1    o
S  |     mXσ σ   (m+1)Xσ

      W1 (m) [ V⇒  R ] | V ⇒ R
V  |          [ o   o ] | 1   o
K  |          [ i   i ] |     m
```

P3.11 New element first

```
LZ       LZ (V, V) ⇒ R
      V  |     o 1    o
      S  |     σ mXσ  (m+1)Xσ

         V ⇒ μR  |  W [ μV  ⇒ μR ]
      V  o   o   |    [ 1     o  ]
```

P3.12 Insertion of the element V_1 in position V_2 of the List V_o

Condition : $0 \leqslant V_2 \leqslant m$

	R (V_o , V_1 , V_2)	⇒	R	
V	o		o	
S	$m\times\sigma$	σ	$1.n$	$(m+1)\times\sigma$

	W1 (V_2)	[V_o ⇒ R_o ; V_i ⇒ R_i]	V_1 ⇒ R_o	V_2	W3 (V_2 , m)	[V_o ⇒ R_o ; V_i ⇒ R_{i+1}]
V						
K						

P3.13 Substition of the element in position V_2 by V_1 in list V_o

Condition : $0 \leqslant V_2 < m$

	R (V_o, V_1 , V_2)	⇒	R	
V	o		o	
S	$m\times\sigma$	σ	$1.n$	$m\times\sigma$

	W1 (V_2)	[V_o ⇒ R_o ; V_i ⇒ R_i]	V_1 ⇒ R_o	V_2	W3 (V_2+1,m)	[V_o ⇒ R_o ; V_i ⇒ R_i]
V						
K						

alternative representation

	V_o	⇒	Z_o	V_1	⇒ Z_o	V_2	Z_o	⇒	R_o
V									
K									
S	$m\times\sigma$		$m\times\sigma$	σ	σ	$1.n$	$m\times\sigma$		$m\times\sigma$

P3.14 Cancellation of the element in position V_1 in the list V_o

	R (V_o , V_1)	⇒	R	
V	o		o	
S	$m\times\sigma$	$1.n$		$(m-1)\times\sigma$

	W1 (V_1)	[V_o ⇒ R_o ; V_i ⇒ R_i]	W3 (V_1, m−1)	[V_o ⇒ R_o ; V_{i+1} ⇒ R_i]
V				
K				

P3.15 Splitting of a list

Supposition : The structure of the individual elements is composed.

$$K\kappa(\sigma) = \tau$$

Meaning: " the component κ of σ has the structure τ ".

```
   | Spκ ( V ) ⇒ R
V  |      o     o
S  |     mXσ    τ
```

```
   | W1(m) [ V ⇒ R ]
V  |       [ o   o ]
K  |       [ i.κ i ]
S  |       [ τ   τ ]
```

P3.16 Inversion of Sequence

```
   | R ( V ) ⇒ R
V  |     o     σ
S  |    mXσ   mXσ
```

```
   | W1 (m) [ V ⇒ R   ]
V  |        [ o   o   ]
K  |        [ i   m−1−i ]
S  |        [ σ   σ   ]
```

c) Programs for sorting

Applicable for lists with elements for which the relation $x < y$ is defined. (" x lesser then y " or " x ranges before y " respectively).

P3.24

```
   | μx [ x ∈ V ∧ (y) (y ∈ V → x ⩽ y ) ] ⇒ R
   |    [ o                 o           ]   o
```

```
   | Min ( V ) ⇒ R
V  |      o     o
S  |     mXσ    σ
```

```
   | V ⇒ Z | W1(m) [ Min(Z,V) ⇒ Z ] | Z ⇒ R
V  | o   o |      [   o o     o ] | o   o
K  | o     |      [     i       ] |
S  | σ   σ |      [   σ σ     σ ] | σ   σ
```

P3.25

$$\mu x \left[\underset{o}{x \in V} \wedge (y) \left(\underset{o}{y \in V} \rightarrow \underset{o}{x \geqslant y} \right) \right] \Rightarrow \underset{o}{R}$$

	Max (V)	⇒	R
V	o		o
S	mXσ .		σ

	V	⇒	Z	Wl(m)	[Maj	(Z, V)	⇒	Z]	Z ⇒ R
V	o		o			o o		o	o o
K	o					i			
S	σ		σ			σ σ		σ	σ σ

P3.26 Proposition : " The list is ordered " .

Ord 0 Predicate symbol : Ord 0 (V)
o

	(R3.8(R1.72))	(	V	),	⇒	R
V	o o		o			o
S			mX1.n			o

Implicit form

$$(x) (y) \left[\underset{o}{x \in V} \wedge \underset{o}{y \in V} \wedge I (x) < I (y) \rightarrow x \leqslant y \right] \Rightarrow \underset{o}{R}$$

This implies

$$(x) (y) \left[\underset{o}{x \in V} \wedge \underset{o}{y \in V} \wedge I (x) + 1 = I (y) \rightarrow x \leqslant y \right] \Rightarrow \underset{o}{R}$$

This is identical with

(R3.6	(R1.72	))(	V	) ⇒	R
o	o		o		o

Explicit form

	Wl (N (V) –1)	[V ⩽ V	⇒	∧R]
V	o	o o		o
K		i i+1		
S	mX1.n	1.n 1.n		o

P3.27 Ordering of a list. Lesser elements first .

Ord 1

	Ord 1	(V)	⇒	R	
V		o		o	
S		mXσ		mXσ	

For implicit representation we first need a criterion stating that the two lists contain the same elements regardless their sequence .

	V ⊖ V
V	o 1
S	mXσ mXσ

Definition of ⊖ :

$$V \ominus V = Df\,(x)\left[x \in V \underset{o}{\rightarrow}\left[N\left[\hat{y}(y \in V \wedge y = x)\right] = N\left[\hat{y}(y \in V \wedge y = x)\right]_{1}\right]_{o}\right]$$

In consequence of that we get the following implicit expression for P3.27 :

P3.27 :

$$\dot{x}\,(x \ominus V \wedge Ord\,0\,(x)) \Rightarrow R$$

The following method is applied : the sorting is accomplished step by step by a sorting of the elements o to i . After the list is sorted up to the element i, then the element i+1 is inserted into the already pre–sorted list as follows, the element i+1 is selected and substituted for Z_1 . Z_1 will then be continually compared to the element ϵ , beginning with ϵ = i. If Z_1 is less than the element ϵ, then this will result in the new element $\epsilon + 1$, and ϵ will be lowered by 1. Otherwise Z_1 results in the new element $\epsilon + 1$ and the insertion process for the element $\epsilon + 1$ will be terminated. If the element, which is to be inserted, is less than all other elements up to i, then $\epsilon = -1$. In this case this results in the new element 0. Z_o is the list, which has to be currently transformed. At the beginning it equals V_o, in the end it results in R_o.

The variation of the main W–program must run from i = 0 to i = m – 2 . (m – 1 is the highest index). At the beginning the list is ordered up to this element i = 0. Therefore, the first element to be inserted is the element i + 1 . In the last variation we have i + 1 = m – 1 , therefore , i = m – 2. According to the rule, chapter 1, page 64 the corresponding variation is

P3.27

```
      W1(m-1)
         Ord 1 ( V ) ⇒ R
                 mXσ   mXσ

      |  V   ⇒  Z
V     |  o      o
S     |  mXσ    mXσ

      |  W1 (m-1)  [ Z   ⇒   Z  |  i  ⇒  ε
V     |              o       1  |
K     |              i+1        |
S     |              σ       σ  |  1.n   1.n
      |              W[ ε ⩾ 0 → [ Z < Z → [ Z ⇒ Z | ε-1 ⇒ ε ]]]
V     |                              1   o   o   o
K     |                                  ε   ε   ε+1
S     |                              σ   σ   σ   σ
      |                              Z < Z → [ Z ⇒ Z | Fin³ ]
V     |                              1   o   1   o
K     |                                  ε       ε+1
S     |                              σ   σ   σ   σ
      |              ε = -1 → Z ⇒ Z
V     |                       1   o
K     |              1.n  1.n σ   σ  ]

      |  Z   ⇒   R
V     |  o       o
S     |  mXσ     mXσ
```

P3.28 As P3.27 , however the highest values first .
" > " instead of " < ".

```
Ord 2 ( V ) ⇒ R
        o     o
```

Example for P3.27 V = (3, 2, 1, 7)
o

		Z_o				Z_1
i	ε	0	1	2	3	
0	0	3	2	1	7	2
0	–1	3	3	1	7	2
1	1	2	3	1	7	1
1	0	2	3	3	7	1
1	–1	2	2	3	7	1
2	2	1	2	3	7	7
		1	2	3	7	

d) Programs for counting

$N(V_o)$ = " number of elements of the list V_o "

P3.29

$$(N(R\square))(V)_o \equiv N(\hat{x}(x \in V \wedge R\square(x)))$$

Explicit form

$0 \Rightarrow \epsilon$

$$W1\,(N(V))_{o,i}\ \left[\, R\square(V)_o \rightarrow (\epsilon+1 \Rightarrow \epsilon)\,\right]\ \Big|\ \epsilon \Rightarrow R_o$$

P3.30 Number of elements that are different from each other

$$R(V_o) \Rightarrow R_o \qquad m\times\sigma \quad 1.n$$

$$N(\hat{x}(x \in V_o)) \Rightarrow R_o$$

Explicit form

	V ⇒ Z	m	⇒	ϵ
V	o o			2
S	m×σ □×σ	1.n		1.n

	W	ϵ ≠ 0 →	Z ⇒ Z		0 ⇒ ϵ		0 ⇒ ϵ	
V		2	o	2		o		1
K			o					
S		1.n	σ	σ	1.n	1.n	1.n	1.n

		W3(1,ϵ)	Z = Z →	(ϵ + 1 ⇒ ϵ)						
V		2	2	o	o		o			
K				i						
S			σ	σ						
			Z = Z →	Z ⇒ Z		ϵ	ϵ + 1 ⇒ ϵ			
V			2	o	o	1	1		1	
K				i	i					
S			σ	σ	σ	σ				
			(Z,ϵ) ⇒	μR	ϵ ⇒	ϵ	Z	(ϵ×σ) ⇒ Z	(ϵ×σ)	
V			2 o	o	1	2	1	2	o	2
K										
S			σ	σ	1.n	1.n				

e) Propositions on two lists

Marginal values for P3.32 to P3.36

```
    | R (   V ,    V ) ⇒ R
V   |       o      1      o
S   |       mXσ    nXσ
```

P3.32 Identity of two lists

```
    | V = V ⇒ R
V   | o   1   o
S   | mXσ mXσ o
```

```
    | W1(m) [ V = V ⇒ ∧ R ]
V   |        | o   1     o |
K   |        [ i   i       ]
```

P3.33 Lists of the same composition (see also page 104)

```
    | V = V ⇒ R
V   | o   1   o
S   | mXσ mXσ o
```

$$\mathrm{Ord}\ \underset{o}{1}\ (\ V\ ) = \mathrm{Ord}\ \underset{1}{1}\ (\ V\ ) \Rightarrow \underset{o}{R}$$

P3.34 " At least one element is contained in both lists "

Implicit form

$$(\ \mathrm{Ex}\ )\ (\ \underset{o}{x \in V} \wedge \underset{1}{x \in V}\ ) \Rightarrow \underset{o}{R}$$

Explicit form

```
    | W1 (m) [ W1(n) [ V ┌i = V ┌i ⇒ VR ] ]
V   | o        | 1    | o └o   o └1    o | |
K   |          [      [                   ] ]
```

P3.35 For every element of the first list there exists an equal element in the second list (sequence and number of repetitions may be different , however) .

Implicit form

$$(\ x\ )\ \left[\underset{o}{x \in V} \rightarrow \underset{1}{x \in V} \Rightarrow \underset{o}{R}\right]$$

In observation of the rules of chapter 1, page the following expression results :

```
    | +⇒Z [ W1(m) |– ⇒ Z | W1 ( n ) [ V ┌i = V ┌i ⋁Z ⇒ Z ] ] Z ] | Z ⇒ Z |
V   |   o  | o    |     1 |  1       | o └o   1 └1   1   1 | | o |   1   o |
K   |        [      |       |          [                     ] ]   |         |
```

$$\underset{o}{Z} \Rightarrow \underset{o}{R}$$

This can be transformed according to the rule of chapter p, page 19 into the form :

```
  | W1(m) ┌ – ⇒ Z | W1 ( n )  ┌ V ┌ i  = V ┌ i  ⇒ V Z ┐ | Z ⇒ Λ R ┐
V |       │     1 |            │ o ┘ o    1 ┘ 1      1 │ | 1     o │
K |       └       |            └                      ┘ |         ┘
```

P3.36 All elements of the first list are also contained in the second list, and vice versa.

Implicit form :

```
Ord 1 ( V ) ⇒ Ord 1 ( V ) ⇒ R
        o             1     o
```

respectively

```
R3.35 ( V, V ) Λ R3.35 ( V, V ) ⇒ R
        o  1             1  o     o
```

This expression can be simplified. But this will not be discussed here.

f) Propositions on two lists for which a relation R□ is true.

Marginal values for P3.40 to P3.44

```
  | ( R ( R□ ))( V   ,  V   ) ⇒ R
V |            o      1         o
S |            mXσ    nXσ       o
```

P3.40 For every two elements of the same index the relation R□ is true.

Supposition : m = n

(The lists can be projected on each other without a change of sequence of the elements) .

```
  | W1 ( m ) ┌ R□ ( V,  V ) ⇒ Λ R ┐
V |          │      o   1       o │
K |          └      i   i         ┘
```

P3.41 " The lists can be projected on each other by a change of the sequence of elements " .

```
R3.40 ( Ord 1 ( V ) , Ord 1 ( V ))⇒ R
o                 o             1   o
```

P3.42 According to P3.34, P3.35, P3.36

P3.43 } however, instead of the relation " = " the relation " R□ " .

P3.44 }

g) Production of a new list from two given lists :

P3.48 Horizontal composition of two lists with the same number of elements. The elements with the same index are combined .

```
Qz  |  Qz ( V   , V   ) ⇒ R
    |      o       1       o
V   |
S   |      mXσ     mXτ     mX(σ,τ)

    |  W1 ( m )      [ ( V V ) ⇒ R ]
    |                [   o 1     o ]
V   |                [   i i     i ]
K   |                [   σ τ     (σ,τ) ]
S   |
```

P3.49 Composition of two lists, the elements of which are of the same structure.

```
    |  Lz( V   , V   ) ⇒ R
    |      o       1       o
V   |
S   |      mXσ     nXσ     (m+n)Xσ

    |  W1 ( m )  [ V ⇒ R ]  |  W1 ( n )  [ V ⇒ R ]
    |            [ o   o ]  |            [ 1   o ]
V   |            [ i   i ]  |            [ i   m+i ]
K   |            [ σ   σ ]  |            [ σ   σ ]
S   |
```

The operation symbol Lz may also be applied to more than two lists , respectively to single elements (see page 101).

```
Lz  ( V   , V   , ... , V   )
      □Xσ   □Xσ         □Xσ
```

P3.50 Horizontal composition with a constant

Qz (Constant first)

```
    |  Qz ( V,  V   ) ⇒ R
    |      o   1       o
V   |
S   |      σ   mXτ     mX(σ,τ)

    |  W1 ( m )      [ ( V,V ) ⇒ R ]
    |                [   o 1     o ]
V   |                [     i     i ]
K   |                [   σ τ     (σ,τ) ]
S   |
```

P3.51 Horizontal composition with a constant

(Constant second)

```
   | Qz  ( V   , V   ) ⇒ R
V  |       o     1     o
S  |       mXσ   τ     mX(σ,τ)
```

```
   | W1 ( m )  [ ( V, V ) ⇒ R ]
V  |           [   o  1    o ]
K  |           [   i        i ]
S  |           [   σ  τ    (σ,τ) ]
```

The operation symbol Qz is also applicable to several lists with an equal number of elements respectively single elements.

```
Qz  ( V   , V   , V   ,... , V   )
      o     1     2          n
      mXσ0  mXσ1  mXσ2       mXσn
```

P3.52 Numbering of elements

```
Nr |  Nr( V   ) ⇒ R
V  |      o      o
S  |      mXσ     mX(1.n,σ)
```

```
   |  W1 ( m )  [ ( i , V ) ⇒ R ]
V  |            [     o     o ]
K  |            [     i     i ]
S  |            [ 1.n  σ    (1.n,σ) ]
```

Examples for Lz, Qz and Nr

V_o = (13, 24, 01, 53)

V_1 = (12, 17)

V_2 = (03, 13, 12)

$Lz_{o\,.\,1}(V, V)$ = (13, 24, 01, 53, 12, 17)

$Lz_{1\;o}(V, V)$ = (12, 17, 13, 24, 01, 53)

$Lz_{o\;1\;2}(V, V, V)$ = (13, 24, 01, 53, 12, 17, 03, 13, 12)

V_o = (a, c, b, f)

V_1 = (7, 5, 3, 1)

V_2 = (α, β, γ, δ)

$Qz_{o\;1}(V, V)$ = (a7, c5, b3, f1)

$Qz_{2\;1}(V, V)$ = (α7, β5, γ3, δ1)

$Qz_{o\;2\;1}(V, V, V)$ = (aα7, cβ5, bγ3, fδ1)

V_3 = a

$Qz_{3\;o}(V, V)$ = (aa, ac, ab, af)

$Qz_{2\;3}(V, V)$ = (αa, βa, γa, δa)

$Nr_{o}(V)$ = (0a, 1c, 2b, 3f)

2) Free Calculus of Lists

(The results are lists of variable size)

P3.64 $\hat{x}_{o}\,(x \in V_{o} \wedge R\square(x)) \Rightarrow R_{o}$

P3.65 $\hat{\hat{x}}_{o}\,(x \in V_{o} \wedge R\square(x)) \Rightarrow R_{o}$

see chapter 1, page 70 -72

P3.66 Partial list from i = o to i = V
1

T61

```
        |  T1 1 ( V   ,  V   ) ⇒ R  ┌─ ( V+1 ) X σ
V       |        o      1      o    │      1
S       |        mXσ    1.n        ─┘
```

Implicit form

$$\hat{x}\ (\ x \in V\ .\ I\ (x) \leq V\ ) \Rightarrow R \qquad (o \quad 1 \quad o)$$

Explicit form

```
        |  W1 ( V+1 )   ⌈V ⇒ R⌉
V       |      1        | o  o |
K       |               ⌊i  i ⌋
```

P3.67 Partial list from i = V to i = m – 1
1

T62

```
        |  T1—2 ( V   ,  V   ) ⇒ R  ┌─ ( m – V )Xσ
V       |        o      1      o    │       1
S       |        mXσ    1.n        ─┘
```

Implicit form

$$x\ (\ x \in V \wedge I\ (x) \geqslant V\ ) \Rightarrow R \qquad (o \quad 1 \quad o)$$

Explicit form :

```
        |  W3 ( V,m )   ⌈V ⇒ R⌉
V       |      1        | o  o |
K       |               ⌊i  i ⌋
```

P3.68 Evaluation of the number of identical elements

```
R ( V   ) = R
    o       o
    mXσ     □X(σ,1.n )
```

Implicit form

$$\hat{x}\ (\ x \in V\ ) \Rightarrow Sp0\ (\ R\ )\ (o \quad o)$$

```
        |  ( x )  [ x ∈ R → x = N [ ŷ ( y ∈ V ∧ y = x ) ] ]
V       |             o             o                    o
K       |                1
```

Explicit form

```
    | W1 ( N ( Z ))  [ N ( x̂ ( x ∈ V ∧ x = Z ))  ⇒ Z ]  | Qz ( Z, Z ) ⇒ R
V   |       o        [                o    o     1 ]  |      o 1     o
K   |                 [                     i     i ]  |
S   |       □Xσ       [ 1.n    σ   mXσ    σ     1.n ]  |      σ 1.n   □X(σ, 1.n)
```

Z_o = Sp0 (R) means : the list of the elements contained in V_o without repetitions .

Z_1 = Sp1 (R) specifies how frequently the corresponding element is contained in V_o .

Example : V_o = (14, 13, 1, 14, 1, 1, 9, 4)

R_o =
14,2
13,1
1,3
9,1
4,1

P3.69 Combination of the elements, which cohere through the relation R□ . The cohering elements are each compiled to a group. These are numbered and supplemented by a group number.

```
    | R ( V ) ⇒ R
V   |    o     o
S   |   mXσ   □X(σ,1n)
```

see P3.9 , page 99. (100)

Meaning of the intermediate values :

Z_o List for compiling the group just investigated

Z_1 Element which is just being tested for coherence with other elements

Z_2 List for compiling the result

Z_3 Remainder list of the elements not yet assigned to groups

ϵ Group number

P3.69

```
   |  V     ⇒ Z     | ∅ ⇒ Z
V  |  o       3     |     2
S  |  mXσ    □Xσ    |    □X(σ,1.n)

   |  W6 [ Z ⇒ Z     | 0 ⇒ ε
V  |       3   o    |
K  |       o         |
S  |       σ   □Xσ  |      1.n

   |       W6 [ Z ⇒ Z
V  |          |   o   1
K  |          |   o
S  |          |   σ   σ
   |          | Lz [ Z , x̂ ( x ∈ Z ∧ x ∈ Z ∧ (R□ ( Z,x ) ∨ R□ (x,Z )) ] ⇒ Z
V  |          |      o        3        o        1          1        o
S  |          |     □Xσ       σ      □Xσ  σ   □Xσ       σ σ      σ σ   □Xσ

   |       Lz( Z ,      Qz ( Z   , ε )) ⇒ Z     | ε+1 ⇒ ε
V  |          2            o          2        |
S  |         □X(σ,1.n)    □Xσ  1.n   □X(σ,1.n)  | 1.n   1.n

   |       x̂ ( x ∈ Z ∧ ¬(x ∈ Z) ) ⇒ Z
V  |             3        o        3
S  |          σ  □Xσ  σ   □Xσ      □Xσ

   |  Z              ⇒  R
V  |  2                 o
S  |  □X(σ,1,n)        □X(σ,1.n)
```

Example : V = (3, 5, 7, 15, 4, 9, 16, 5, 3, 3,)
o

R□ (x,y) ≡ | x − y | ≤ 2

R = o
[3, 0
  5, 0
  7, 0
  4, 0
 15, 1
 16, 1
  9, 2]

Group 0 = (3, 5, 4)
Group 1 = (7, 9)
Group 2 = (15, 16)

P3.70 Union of two lists

	V	∪	V	⇒ R
V	o		1	o
S	mXσ		nXσ	□Xσ

$\hat{x}\ (\ x \in V \vee x \in V\ ) \Rightarrow R$

o 1 o

P3.71 Section of two lists

	V	∩	V	⇒ R
V	o		1	o
S	mXσ		nXσ	□Xσ

$\hat{x}\ (\ x \in V \wedge x \in V\ ) \Rightarrow R$

o 1 o

Note :

In the calculus of lists the commutative law for the operations ∪ and ∩ is not true. Therefore , it is not generally true that :

V ∪ V = V ∪ V

o 1 1 o

V ∩ V = V ∩ V

o 1 1 o

The two cases differ in the sequence of the operands. The first list is competent for the sequence.

But it is generally true that

V ∪ V ⇒ V ∪ V

o 1 1 o

(see page 104) 105

Example

$V_o = (1, 3, 3, 7, 3, 2, 1, 5)$, $V_1 = (1, 5, 5, 6, 8, 2. 3)$

$V_o \cup V_1 = (1, 3, 7, 2, 5, 6, 8,)$, $V_1 \cup V_o = (1, 5, 6, 8, 2, 3, 7)$

$V_o \cap V_1 = (1, 3, 2, 5)$, . $V_1 \cap V_o = (1, 5, 2, 3)$

V. Programs with Lists of Pairs

(Calculus of Relations)

1) General

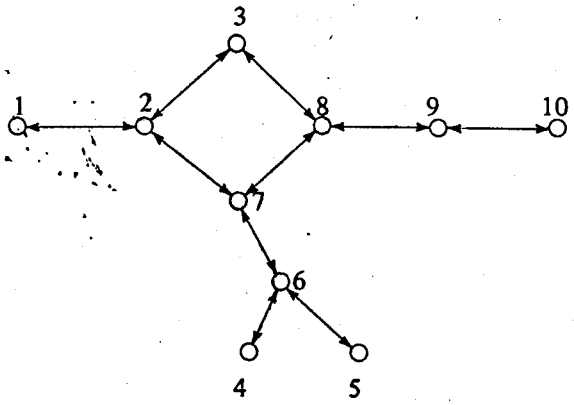
In mathematical logic a relation represents a predicate with two variables, for instance a < b. In general representation , a relation may be represented by an operation symbol (for instance " < ") , or by a predicate symbol (e.g. Verb (a,b) ≡ a is connected to b) .

In the special representation mainly three forms can be distinguished :

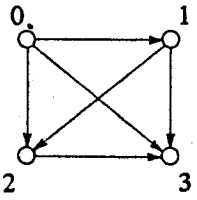
a) Graphic representation by an arrow-diagram

Example :

" Pole x is connected conductively with pole y "



Since this relation is symmetrical, it is represented by antidirentional arrows. For the set of elements 0, 1, 2, 3 the relation a < b has the following diagram :



b) Representation by a matrix :

For the two examples of a) the matrices have the following form :

	1	2	3	4	5	6	7	8	9	10
1	+	+								
2	+	+	+				+			
3		+	+					+		
4				+		+				
5					+	+				
6				+	+	+	+			
7		+				+	+	+		
8			+				+	+	+	
9								+	+	+
10									+	+

	0	1	2	3
0		+	+	+
1			+	+
2				+
3				

The matrix of the symmetrical relation is symmetrical to the main diagonal.

c) Representation by a list of pairs .

The pair list contains all pairs of elements for which the relation R□ is true. For the two examples mentioned above the lists have the following form :

1	–	2	0	–	1
2	–	1	0	–	2
2	–	3	0	–	3
3	–	2	1	–	2
3	–	8	1	–	3
8	–	3	2	–	3
8	–	9			
9	–	8			
9	–	10			
10	–	9			
2	–	7			
7	–	2			
7	–	8			
8	–	7			
7	–	6			
6	–	7			
4	–	6			
6	–	4			
4	–	5			
5	–	4			

Since the relation " conductively connected to " is generally symmetrical (if rectifiers are excluded) then the duplicate listings can be replaced by the single ones following :

1	–	2
2	–	3
3	–	8
8	–	9
9	–	10
2	–	7
7	–	8
7	–	6
4	–	6
4	–	5

Representation by a list of pairs is best suited for computations. Consequently, the theory of lists of pairs will be discussed first .

The general structure symbol for lists of pairs is S.4

$$S.4 = m \times (\dot{\sigma}, \tau)$$

The first element of a pair is called " front element " the second " back element " . Generally , the front elements may be of another structure than the back elements. However , with symmetrical

relations $\sigma = \tau$ is true. The structure of the list then is $m \times 2\sigma$. Generally all programs for lists can be applied to lists of pairs , wherein the elements of the list are identical with the pair . As an example the expression

	V ⇔ V
V	o 1
S	4 4

states , that $\overset{V}{o}$ and $\overset{V}{1}$ represent the same relation, possibly however, in a different sequence of the members.

2) Propositions on lists of pairs (see also section 5, page 124)

a) Front elements and back elements are of the same structure Marginal values for P4.1 to P4.10

	R (V) ⇒ R
V	o o
S	$m \times 2 \times \sigma$ o

P4.1 Coherence of all pairs

	(R3.9 (R2.8))(V) ⇒ R
V.	o o o o
S	$m \times 2 \times \sigma$ o

To a list of pairs, for which R4.1 is true, an arrow diagram coherent in all elements exists , e.g. figures on page 116.

P4.2 The list contains symmetric pairs

	(Ex) (x ∈ V ∧ x = x) ⇒ R
V	o o
K	o 1
S	2σ $\square \times 2\sigma$ σ σ o

P4.3 The list contains pairs of pairs of which one is the mirror image of the other

	(Ex) (Ey) (x ∈ V ∧ y ∈ V ∧ I(x) ≠ i(y) ∧ x = y ∧ x = y) ⇒ R
V	o o o
K	o 1 1 o

Explicit form according to P3.4 , page 98 | G9,

P4.4 " The list contains no repetitions and no mirror image "

	R3.2 (V) ∧ R4.3 (V) ⇒ R
V	o o o o o
S	o $\square \times 2\sigma$ o $\square \times 2\sigma$ o

P4.6 "For each pair there exists a mirror image (condition for symmetry).

```
    |  (x) [ x ∈ V →
V   |          o
K   |
S   | 2σ     [ 2σ   mX2σ
```

```
    |        (Ey) [ y ∈ V  ∧  I(x) ≠ I(y) ∧ x = y ∧ x = y ] ⇒ R
V   |              .  o                                      o
K   |                                    o  1   1   o
S   |           2σ    [ 2σ  mX2σ  2σ     2σ  σ   σ   σ   σ ]  o
```

(see P3.7, page 97)

P4.7 " All front elements are different from all back elements ".

```
    |  (x) (y) [ ( x ∈ V  ∧   y ∈ V     → x ≠ y ) ] ⇒ R
V   |               o          o                      o
K   |                                  o  1
S   | 2σ  2σ      [ 2σ  mX2σ  2σ  mX2σ  σ  σ ]       o
```

P4.8 General incoherence of the pairs

```
(R3.4 (R2.8)) ( V ) ⇒ R
o     o       o     o
```

P4.9 Condition for reflexivity

" Each front or back element mentioned is listed as a pair with itself".

```
    |  (x) [ x ∈ Sp0 ( V ) ∧ x ∈ Sp1 ( V ) → (Ey) [ y ∈ V ∧ y = x ∧ y = x ] ] ⇒ R
V   |              o                 o                  o                      o
K   |                                                      o  1
S   | σ  [ σ   mXσ  mX2σ  σ   mXσ  mX2σ   2Xσ  [ 2σ  mX2σ                ] ]
```

P4.10 Condition for transitivity

```
    |  (x) (x)  [ x ∈ V  ∧  x ∈ V  ∧  x = x → (Ex) [ x ∈ V  ∧  x = x ∧ x = x ] ] ⇒ R
V   |  o   1    [ o   o     1   o     o   1   2     [ 2   o     o   2   1   2 ]   o
K   |                                 1   o                   o   o   1   1
S   |  2σ  2σ   [ 2σ  mX2σ  2σ  mX2σ  σ   σ   2σ    [ 2σ  mX2σ  σ   σ   σ   σ ] ]  o
```

b) Propositions on lists of pairs

Front and back elements of different structure

Marginal values for P4.12 to P4.15

```
    | R (V  ) ⇒ R
V   |    o      o
S   |    mX(σ,τ)o
```

P4.12 Coherence of all pairs

```
( R3.9 ( R2.2 ) )( V ) ⇒ R
  o      o        o     o
```

P4.13 All front elements are equal to each other

```
R3.1 ( Sp0 ( V ) )⇒ R
         o      o
```

P4.14 All back elements are equal to each other

```
R3.1 ( Sp1 ( V ) )⇒ R
         o      o
```

3) Programs for ordering of lists of pairs

Supposition : $x < y$ is defined for the structures of the pairs ' elements

Marginal values for P4.24 to P4.27

```
   | R ( V       ) ⇒ R
V  |     o         o
S  |     mX(σ,τ)    mX(σ,τ)
```

(see also P3.27 page 103)

P4.24 Ordering of the pairs refering to the front elements (see page ——)

```
Ord 2 | Ord 2 ( V      ) ⇒ R
    V |        o        o
    S |        mX(σ,τ)

      | V      ⇒ Z
    V | o      . o
    S | mX(σ,τ)  mX(σ,τ)

      | W1 ( m−1 ) ⌈ Z          ⇒ Z     | i ⇒ ϵ                          ⌉
    V |            | o            1     |
    K |            | i+1                |
    S |            | (σ,τ)        (σ,τ) | 1.n  1.n                       |
      |            | W⌈ϵ ⩾ 0 → ⌈Z < Z → ⌈Z    ⇒ Z   | ϵ −1  ⇒ ϵ ⌉⌉⌉
    V |            |    |       1  o     | o     o                        |
    K |            |    |       o  ϵ.o   | ϵ     ϵ+1                      |
    S |            |    |       σ  σ     | (σ,τ) (σ,τ)                    |
      |            |    |  Z < Z → ⌈Z    ⇒ Z   | Fin³                   ⌉⌉
    V |            |    |       1  o     | 1     o                        |
    K |            |    |       o  ϵ.o   |       ϵ+1                      |
    S |            |    |       σ  σ     | (σ,τ) (σ,τ)                    |
      |            |  ϵ = −1 → ⌈Z        ⇒ Z   ⌉
    V |            |           | 1          o   |
    K |            |           |                |
    S |            | ( 1.n  1.n| (σ,τ)      (σ,τ)⌉                         ⌉

      | Z          ⇒ R
    V | o            o
    S | mX(σ,τ)      mX(σ,τ)
```

P4.25 Ordering referring to the back elements

```
Ord 3 | Ord 3 ( V      ) ⇒ R
    V |        o        o
    S |        m(σ,τ)    mX(σ,τ)
```

As P4.24 ; however, $\begin{matrix} Z & < & Z \\ 1 & & o \\ 1 & & \epsilon.1 \\ \tau & & \tau \end{matrix}$ instead of $\begin{matrix} Z & < & Z \\ 1 & & o \\ 1 & & \epsilon.o \\ \sigma & & \sigma \end{matrix}$

P4.26 Ordering reffering to front and back elements. Front elements have preference however .

```
Ord 4          | Ord 4  ( V      ) ⇒ R
          V    |                     o
          S    |          mX(σ,τ)    mX(σ,τ)

     |  V        ⇒ Z
V    |  o          o
S    |  mX(σ,τ)    mX(σ,τ)

     |  W1 (m−1)  [ Z        ⇒ Z     |  i     ⇒ ε                    ]
V    |             o          1       |                               |
K    |             i+1                |                               |
S    |             (σ,τ)      (σ,τ)   |  1.n    1.n                   |
     |                                                                 |
     |             W[ε ⩾ 0 → [ Z < Z   ( Z = Z   Z < Z ) ⇒ Z        ] ]
V    |                       | 1   o     1   o   1   o     2        |  |
K    |                       | o   ε.o   o   ε.o 1   ε.1             |  |
S    |                       | σ   σ     σ   σ   τ   τ     o        |  |
     |                       |                                        |  |
     |                       | Z →[Z     [    ⇒ Z     | ε−1  ⇒ ε ]    |  |
V    |                       | 2  | o     |      o                    |  |
K    |                       |    | ε     |      ε+1                  |  |
S    |                       | o  | (σ,τ) [      (σ,τ)               |  |
     |                       |                                        |  |
     |                       | (Z)→[Z    |    ⇒ Z     | Fin³ ]       |  |
V    |                       | 2  | 1    ⫽|      o                    |  |
K    |                       |    |      ⫽|      ε+1                  |  |
S    |                       | o  | (σ,τ) |      (σ,τ)               |  |
     |                       |                                        |  |
     |             ε =  −1 → Z ⇒ Z                                    |  |
V    |                       | 1   o                                  |  |
K    |                       |     o                                  |  |
S    |             1.n  1.n  | σ   σ                                  |  |

     |  Z        ⇒ R
V    |  o          o
S    |  mX(σ,τ)    mX(σ,τ)
```

P4.27 Ord 5 (V) as P4.26. Here however , back elements have preference.
o

Ord 5 Alternative expression for Z
2

```
     |  Z < Z ∨ ( Z = Z ∧ Z < Z ) ⇒ Z
V    |  1   o     1   o   1   o     2
K    |  1   ε.1   1   ε.1 o   ε·o
S    |  τ   τ     τ   τ   σ   σ     o
```

P4.28 Ordering within the pairs

```
Ord 6        | Ord 6 ( V      ) ⇒ R
          V  |       o          o
          S  |       mX2Xσ      mX2σ

             | W1 ( m ) [ ( R1.106 ( V    ) ) ⇒ R ]
          V  |          [           o        o   ]
          K  |          [           i        i   ]
          S  |          [  2σ       2Xσ      2Xσ ]
```

4) Field, Front Area and Back Area of a List of Pairs, respectively a Relation

Marginal values for P4.32 ; P4.33; P4.34

```
   | R ( V      ) ⇒ R
 V |     o        o
 S |     mX2Xσ    □Xσ
```

P4.32 Field of a list of pairs

```
   | x̂ [x ∈ Sp0 ( V    ) ∨ x ∈ Sp1 ( V    )] ⇒ R
 V |           [      o                 o    ]   o
 S |           [σ  mXσ  mX2σ   σ  mXσ   mX2σ]   □Xσ
```

Function Symbol : Ca (V) (Campus)
o

~~(Compus)~~

P4.33 Front Area of a list of pairs

```
   | x̂ ([x ∈ Sp0 ( V )]) ⇒ R
 V |           [      o ]   o
 S |           [σ  mXσ  mX2σ] □Xσ
```

Function Symbol Vb (V)
o

P4.34 Back Area of a list of pairs

```
   | x̂ [x ∈ Sp1 ( V )] ⇒ R
 V |    [      o    ]   o
 S |    [σ  mXσ  mX2σ]   □Xσ
```

Function Symbol Nb (V)
o

5) Programs for Structures of Relations represented by Lists of Pairs

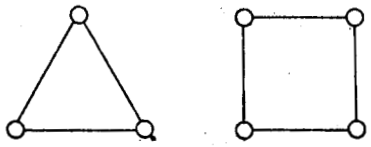
P4.40 Subprogram : Number of the connections of an element of a pair.

```
   | R ( V     , V )      ⇒ R
V  |   o       1          o
S  |   mX2σ     σ          1.n

   | x̂ [ x ∈ V  ∧  x ≠ x ]  ⇒ Z
V  |     [ o                ]  o
K  |     [      o    1      ]
S  |     [ 2σ  mX2σ  σ   σ  ]  □X2σ

   | N [ x̂ [ x ∈ Z    ∧ ( x = V ∨ x = V ] ] ⇒ R
V  |     [   o          1       1     ] ]  o
K  |     [             o        1     ] ]
S  |     [ 2σ  □X2σ     σ       σ   σ ] ]  1.n
```

Marginal values for P4.41 to P4.45

```
   | R ( V ) ⇒ R
V  |   o     o
S  |   mX2σ  o
```

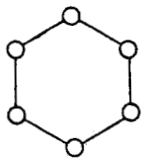
P4.41 The relation consists of individual circles with at least 3 elements



```
   | R4.4 ( V ) ∧ ( x )  [ x ∈ Ca( V ) → R4.40 ( V , x ) = 2 ]  ⇒ R
V  |  o      o          [      o          o       o          ]    o
S  |  o      mX2σ   σ   [ σ  □Xσ         1.n     mX2σ  σ  1.n ]    o
```

P4.42 The relation consists of a single circle

```
   | R4.1 ( V ) ∧ R4.41 ( V ) ⇒ R
V  |  o      o     o      o     o
S  |  o      mX2σ  o      mX2σ  o
```



P4.43 The relation consists of individual chains

```
R4.4 ( V ) ∧ ( x )  [ x ∈ Ca ( V ) → [ R4.40 ( V,x ) = 1 ] ]  ⇒ R
o      o            [      o         [ o      o          ] ]    o
                    [                [ ∨R4.40 ( V,x ) = 2 ] ]
                    [                [ o      o          ] ]
```

P4.44 The relation consists of a single chain.

```
R4.1 ( V ) ∧ R4.43 ( V ) ⇒ R
o      o     o      o     o
```

P4.48 All elements are coherent :

(The arrow diagram is a coherent figure) .

```
   | (R3.9 ( R2.1 ))( V      ) ⇒ R
V  |  o      o       o       o
S  |                 mX2σ    o
```

Explicit form

```
   | V   ⇒ Z
V  | o     o
K  | o
S  | 2σ    □X2σ

   | W6 [ Z    ⇒ Z  ] | Lz [ Z    , x̂ [ x ∈ V     ∧ x ∈ Z     ∧ R2.1 (Z,  x )] ] ⇒ Z
V  |    [ o      1  ] |    [ o           [   o         o         o    1       ]   ]   o
K  |    [ o           ] |    [                                                    ]
S  |    [ 2σ     2σ  ] |    [ □X2σ      [ 2σ  □X2σ    2σ  □X2σ    σ    2σ  2σ ]   ]   □X2σ

   | (x) [ x ∈ V   →   x ∈ Z ] ⇒ R
V  |     [   o         o   ]   o
S  | 2σ  [ 2σ  □X2σ    σ  □X2σ ]   o
```

P4.49 Univocal coherence of all elements

(one way principle) :

```
   | R ( V   ) ⇒ R
V  |   o      o
S  |   mX2σ   o

   | (x)( x ∈ V → x ≠ x )
V  |        o
K  |              o   1
S  |    2σ        σ   σ
```

Definition :

If the front and back elements of the pairs contained in V_o are called points, then x and y of V_o is true for any two different points.

There does exist just one list Z, the elements of which are points of V_o with the following propeity.

The first element of Z equals x
The last element of Z equals y .

All pairs composed of adjacent elements of Z are elements of V_o.

In other words : for any two points x, y of V_o there exists a chain of pairs of points which are elements of V_o. The first pair of the chain contains x and the last pair y, wherein the adjacent elements of the chain are coherent.

The exact formulation of this statement is complicated.
Another defintion is given preference:
Recursive defintion:

A single non-symmetrical pair is a univocally coherent list of pairs. By supplementing a univocally coherent list of pairs by a new non-symmetrical pair, another univocally coherent list is produced, if just one and only one point of the new pair is equal to a point of the given list of pairs. In this way the following program is generated:

(1) The first pair of V_o forms the first stage of the growing list Z_o.

Z_1 is the list of the remaining pairs of V_o, i.e. of those not contained in Z_o.

(2) The list Z_1 is investigated for the next element, the front element or the back element of which is contained in the field of Z_o. If both are contained therein, then ambiguity exists and R_o must be negative. The investigation is then terminated.

The investigation (2) has to be repeated as long as there are elements of Z_1 available which fulfill the request for coherence.

(3) If Z_1 does not contain coherent elements, then incoherence exists and R_o must be negative.

If finally Z_1 is empty then R_o is postive.

The corresponding programs are as follows:

```
    | V ⇒ Z
V   | o   o
K   | o
S   | 2σ  □X2σ

    | W [ μ'x [ x ∈ V   ∧ ¬(x ∈ Z) ∧ (Ey) y ∈ Ca(Z) ∧ (y = x ∨ y = x) ] ] ⇒ Z
V   |       o       o                 o                                   1
K   |                                                 o     1
S   |       2σ  mX2σ  2σ  □X2σ        σ   □Xσ       σ  σ  σ  σ          2σ

    |  ¬(Z = Z) ⇒ ΛR  |  Z = Z ⇒ Fin³  |  Lz [ Z   , Z ] ⇒ Z
V   |  1  1     o     |  1  1          |      o     1    o
K   |  o  1           |  o  1          |
S   |  σ  σ     o     |  σ  σ          |      □X2σ  2σ   □X2σ

    | N(Z) = N(V) ⇒ ΛR
V   |    o      o     o
S   |    □X2σ   mX2σ  o
```

This program is very complicated for computation, since the remaining pairs have to be investigated for every new point of a coherent pair, and their front and back elements have to be compared to all pairs already connected. This procedure can be avoided if out of the already connected points (field of Z_o) one point is occasionally selected and then the list of the remaining elements V_o is searched for all elements which are coherent to that point.

```
   | V ⇒ Z
V  | o   o
K  | o
S  | 2σ  □X2σ

   | W [ μx( x ∈ Ca ( Z )) ⇒ Z | x̂ ( x ∈ V ∧ x ∈ Z ∧ ( x=Z ∨ x=Z' )) ⇒ Z
V  |                o    1   |        o        o        1    1     2
K  |                                                  o    1
S  |     σ   □Xσ  m X2σ  σ   |    2σ  m X2σ  2σ  □X2σ   σ σ  σ σ   □X2σ

   | (x) [ x ∈ Ca ( Z ) → ( Ey) ( y ∈ Ca ( Z )∧y ≠ Z ∧ y = x )] ⇒ Z
V  |     [       2                          o       1              3
S  |     [ σ   □Xσ                     σ   □Xσ     σ   σ   σ   σ  o

   | Z ⇒ Λ R | Z  ⇒ Fin³ |  Lz( Z , Z ) ⇒ Z
V  | 3     o | 3         |     o     2     o
S  | o     o | o         |    □X2σ  □X2σ  □X2σ

   | N ( Z ) = N ( V ) ⇒ Λ R
V  |     o         o       o
S  |    □X2σ      m X2σ    o
```

In this program the expression for Z_2 and Z_3 can be combined if an investigation is made as to whether the condition Z_3 is true for this element immediately after the generation of a new element Z_2.

```
   | V ⇒ Z
V  | o   o
K  | o
S  | 2σ  □X2σ

   | W[ μx ( x ∈ Ca ( Z )) ⇒ Z                                          ]
V  |                o    1                                             |
S  |     σ   □Xσ  □X2σ   σ                                             |
   |                                                                    |
   |  W[ μ'x [ x ∈ V ∧ x ∈ Z ∧ ( x = Z ∨ x = Z )] ⇒ Z                   ]
V  |          o       o          1     1     2                        |
K  |                                 o     1                          |
S  |  2σ                             σ  σ  σ  σ  ] 2σ                   |
   |                                                                    |
   |  (Ex) [ x ∈ Ca ( Z ) ∧ x ≠ Z ∧ x = Z ∨ x = Z ] ⇒ Z                 |
V  |                o       1      2     2     3                      |
K  |                                 o     1                          |
S  |  σ   □Xσ          σ  σ  σ  σ  σ  σ  ] o                          |
   |                                                                    |
   |  Z ⇒ Λ R | Z̄ ⇒ Fin⁵ | Lz( Z , Z ) ⇒ Z                             |
V  |  3     o | 3        |     o     2     o                            |
S  |  o     o | o        |    □X2σ  □X2σ  □X2σ                         ]

   | N ( Z ) = N ( V ) ⇒ Λ R
V  |     o         o       o
S  |    □X2σ      m X2σ
```

This program can also be simplified. It is not necessary to extend the variation of x in the expression for Z_3 over the whole area of the field of Z_0; it is sufficient to do so only for those points for which the coherent pairs have not yet been connected.

Now, an auxiliary value Z_4 is introduced. This is equal to the list of the already connected points, but it has not yet been investigated for coherence. So the final form of P4.49 is developed.

P4.49

```
  | V ⇒ Z     | Z ⇒ Z
V | o   o     | o   4
K | o         | o.o
S | 2σ  □X2σ  | σ   □Xσ

  | W⌈μx ( x ∈ Z  ) ⇒ Z                                                  ⌉
V | |          4      1
S | |    σ    □Xσ     σ
  | |
  | | W⌈μ'x⌈x ∈ V  ∧  x̄ ∈ Z  ∧ ( x = Z ∨ x = Z )⌉  ⇒ Z                  ⌉
V | | |    |    o        o          1       1  |    2
K | | |    |                    o       1      |
S | | |    ⌊2σ  mX2σ  σ  □X2σ   σ   σ   σ   σ  ⌋    2σ
  | | |
  | | | x  ⌈x ∈ Z ∧ x ≠ Z⌉ ⇒ Z
V | | |    |    2       1|   5
S | | | σ  ⌊σ   2σ  σ   σ⌋   σ
  | | |
  | | | ( Ex ) ⌈( x ∈ Z  ∧  x = Z )⌉ ⇒ Z
V | | |        |      4         5 |   3
S | | |    σ   ⌊  σ  □Xσ    σ   σ ⌋   o
  | | |
  | | | Z ⇒ Λ R | Z̄ ⇒ Fin^5 | Lz( Z  ,  Z ) ⇒ Z
V | | | 3     o | 3         |     o     2      o
S | | | o     o | o         |     □X2σ  2σ     □X2σ
  | | |
  | | |                       Lz( Z  ,  Z ) ⇒ Z
V | | |                           4     5      4
S | ⌊ ⌊                           □Xσ   σ      □Xσ                      ⌋⌋

  | N ( Z    ) = N ( V    ) ⇒ R
V |     o            o        o
S |     □X2σ         mX2σ     o
```

P4.49

Legend of the Intermediate Values :

Z_o Compiling list of the connected pairs

Z_1 Point just investigated for coherence

Z_2 Pair to be newly connected

Z_3 " The new point (Z_5) is univocally connected " .

Z_4 List of connected points not yet investigated for coherence.

Z_5 Newly connected point.

In this program another auxiliary value Z_5 is introduced , which is equal to the newly connected point (that point of the newly connected pair Z_2 which is not equal to Z_1) . Then the expression for Z_3 becomes especially simple. Instead of $x \in Ca(Z_o) \wedge x \neq Z_1$ it reads : $x \in Z_4$, since Z_4 in the partial list of $Ca(Z_o)$ with those elements, for which the investigation is only relevant. Because of the μ - rule , Z_1 is no longer contained in Z_4, so that the expression $x \neq Z_1$ can be omitted.

Furthermore, the expression

$$x = Z_{2o} \vee x = Z_{21}$$

can be substituted by : $x = Z_5$.

The connected point Z_5 has to be added to the list Z_4 in each evaluation.

P4.50 Combination of coherent elements in proups.

V S	R (V$_o$) ⇒ R$_o$	(R3.69 (R2.1)) (V) ⇒ R
	mX2σ mX(2σ,1.n)	$_o$ $_o$ $_o$ $_o$

P4.51 Investigation for univocal coherence and ordering into coherent groups. Merger of programs P4.49 and P4.50 . In addition, production of a list of the degree of determination of the groups.

	R (V) ⇒ (R, R , R)
V	o o 1 2
S	mX2σ o mX(2σ,1.n) □X2X1.n

Supposition : no symmetrical pairs (the same , as with P4.49) .

Legend of the results :

R_o " The list of pairs is univocally coherent " .

R_1 List of pairs ordered according to their coherence with the numbers of the group.

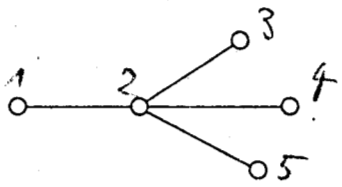
R_2 List of the degree of determination of the groups with group numbers.

R Number of group , R = Degree of determination.
2 2
o 1

For a group univocally coherent in itself, the degree of determination is 1. Each duplicate pair and each pair redundant with regard to coherence increases the degree of determination ϵ by 1.

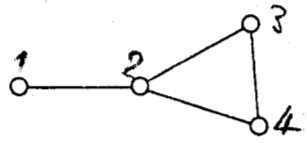
Example :

1 – 2 $\epsilon = 1$
2 – 3
2 – 4
2 – 5



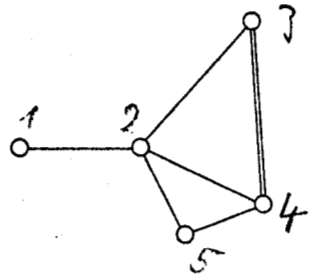
---

1 – 2 $\epsilon = 2$
2 – 3
2 – 4
3 – 4



---

1 – 2 $\epsilon = 4$
3 – 4
2 – 3
2 – 4
3 – 4
4 – 5
5 – 2



P4.51 is developed out of P4.49 (see page 125). The ivestigation, in P4.49 applied to the entire list of pairs, concerns a single group only in P4.51. Therefore , a main W – program is required to investigate the various groups step by step.

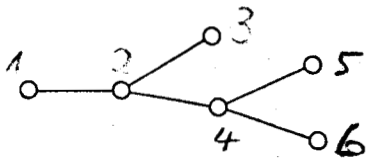
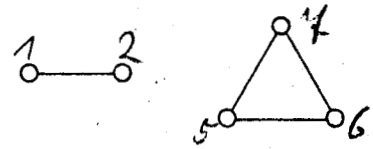
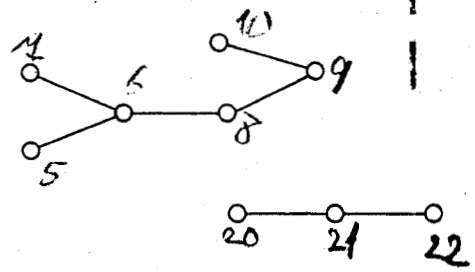
Here the entire list V_o is replaced by the remainder list Z_{10}, which represents the list of the pairs of V_o not yet connected after completion of a group. Consequently, Z_{10} must be equal to V_o at the beginning. Z_o represents only the compiling list of the just investigated group. Therefore, another compiling list Z_{10} is required for the production of R_1 (list of the sorted pairs). After completion of a group, the pair list of the last group Z_o is supplemented by the group number ϵ_o.

Until the evaluation of Z_3 everything proceeds analogous to P4.49. In the case of Z_3 , ϵ_1 must be increased by 1, since it is then a redundant pair with regard to coherence .

After completion of a group, in addition to the evaluation of Z_{11} the next element R_2 which is composed of the two ϵ – values has to be generated.

The condition for R_o (N(...)) standing at the end of P4.49 is replaced by the condition that in the end $\epsilon_1 = 1$, if R_o is positive, i.e. of V_o consists only of one single coherent group.

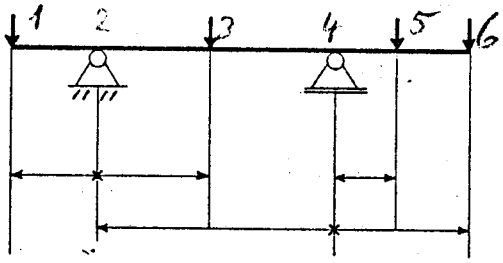
Examples for P4.51

V_o	R_o	R_1	R_2	Coordinated diagram of relations
1 – 2	+	1 – 2 , 0	0 , 1	
2 – 3		2 – 3 , 0		
2 – 4		2 – 4 , 0		
4 – 5		4 – 5 , 0		
4 – 6		4 – 6 , 0		
1 – 2	–	1 – 2 , 0	0 , 1	
5 – 7		5 – 7 , 1	1 , 2	
5 – 6		5 – 6 , 1		
6 – 7		6 – 7 , 1		
1 – 2	–	1 – 2 , 0	0 , 3	
2 – 3		1 – 3 , 0	1 , 1	
1 – 3		0 – 1 , 0	2 , 1	
3 – 4		2 – 3 , 0		
2 – 3		2 – 3 , 0		
20 –21		0 – 4 , 0		
0 – 1		3 – 4 , 0		
0 – 4		20 –21 , 1		
22 –21		22 –21 , 1		
10 – 9		10 – 9 , 2		
8 – 9		8 – 9 , 2		
5 – 6		6 – 8 , 2		
7 – 6		5 – 6 , 2		
6 – 8		7 – 6 , 2		

An important application of P4.49 and P4.51 is the following problem :

Measurement on an axis .

Example :



List of dimensions :

1 – 2
2 – 3
2 – 4
4 – 5
4 – 6

In this example the measurement is a univocal one .

Another example : telephone networks .

If they are univocal then only one connection is possible between any pair of subscribers

P4.52 Extension of P4.51

General analyses of a random list of pairs in regard to coherence. Symmetrical pairs are to be extracted separately. Further, the multiple pairs and the mirrored pairs are specially selected.

Marginal Values .

	R (V) ⇒	R ,	R	, R	, R ,	R ,	R ,	R
V	0	0	1	2	3	4	5	6
S	4	0	□X(2,9)	□X2X9	0	4	0	□X(2,9)

Legend of the structure symbols :

S_0 = Yes–No–Value

S_2 = 2Xσ pair of data

S_3 = □Xσ list

S_4 = □X2σ list of pairs

S_9 = positive integer number

Meaning of the result values :

R_o The list of pairs V_o is univocally coherent.

R_1 List of pairs ordered by groups, and supplemented by a group number.

R_2 List of the groups and of their degrees of determination.

R_3 " The list contains symmetrical pairs " .

R_4 List of the symmetrical pairs.

R_5 The list contains pairs, which are equal to another pair or to the mirror image of it.

R_6 List of pairs according R_5 .

Legend of the intermediate values :

	Z	List of remainders of V_o after completion of a group
V	o	
S	4	
	Z	Compiling list of the group just being developed
V	1	
S	4	
	Z	List of the points already connected to Z_5
V	2	
S	3	
	Z	List of the already connected points of the just evaluated group
V	3	
S	3	
	Z	Last newly connected pair
V	4	
S	2	
	Z	Point, the connections of which are just being investigated for coherence
V	5	
S	σ	
	Z	Last point connected to Z_5 via Z_4
V	6	
S	σ	
	Z	Criterion for " multiple pair "
V	7	
S	o	
	Z	Criterion for ambiguous coherence
V	8	
S	o	

	Z	Compiling list of the result R_1
V	10	
S	□X(2,9)	

	ϵ	Group–number
V	o	
S	9	

	ϵ	Degree of determination of the just evaluated group
V	1	
S	9	

P4.52

	①	②	③
	V ⇒ Z	0 ⇒ Z	o ⇒ ϵ
V	o o	10	o
S	4 4	□X(2σ,9)	9

		④		⑤		⑥	
	W6	Z ⇒	Z	1 ⇒	ϵ	0 ⇒	Z
V		o	3		1		1
K		o.o					
S		σ	3		9		4

		⑦			⑧	
	W	μx (x ∈	Z) ⇒	Z	0 ⇒	Z
V			3	5		2
S			3	σ		3

		⑨								
	W	μ'x	x ∈ Z ∧ $\overline{x \in Z}$ ∧ (x = Z ∨ x = Z)							⇒ Z
V			o		1		5		5	4
K						o		1		
S			2	4	2	4	σ	σ	σ	σ 2

		⑩						
		Z = Z →	+ ⇒ V	R	Z ⇒ μ	R	Fin^2
V		4 4		3	4	4	
K		o 1					
S		σ σ		o	2	4	

		⑪				
		x'	x ∈ Z ∧ x ≠ Z			⇒ Z
V			4		5	6
S		σ	σ	2	σ	σ

V S

(12) $(Ex)\begin{bmatrix} x \in Z \wedge x = Z \Rightarrow Z \\ 2 \quad 6 \quad 7 \\ \sigma \;\; 3 \;\; \sigma \;\; \sigma \quad o \end{bmatrix}$

(13) $Z \vee (Ex)\begin{bmatrix} x \in Z \wedge x = Z \Rightarrow Z \\ 3 \quad 6 \quad 8 \\ \sigma \;\; 3 \;\; \sigma \quad \sigma \quad o \end{bmatrix}$

(14) $Z \to \begin{bmatrix} + \Rightarrow R & (Z,\epsilon) \Rightarrow \mu R \\ z \quad 5 & 4\,o \quad 6 \\ o \quad o & 2\,9 \quad \square X(2,9) \end{bmatrix}$

(15) $\overline{Z} \to \begin{bmatrix} Lz\,(Z,Z) \Rightarrow Z \\ 2\,6 \quad 2 \\ 3\,\sigma \quad 3 \end{bmatrix}$

(16) $Z \to (\epsilon+1 \Rightarrow \epsilon)$

8 1 1

o 9 9

(17) $\overline{Z} \Rightarrow \wedge R$

8 o

o o

(17a) $\overline{Z} \to \begin{bmatrix} Lz(Z, Z) \Rightarrow Z \\ 3\,6 \quad 3 \\ 3\,6 \quad 3 \end{bmatrix}$

(18) $Lz(Z, Z) \Rightarrow Z$

1 4 1

4 2 4

(19) $(\epsilon, \epsilon) \Rightarrow R$

o 1 2

9 9 4

(20) $Lz\begin{bmatrix} Z & , & Qz(Z,\epsilon) \Rightarrow Z \\ 10 & & 1\,o \quad 10 \\ \square X(2,9) & & 4\,9 \quad \square X(2,9) \end{bmatrix}$

(21) $\hat{x}(x \in Z \wedge \overline{x \in Z}) \Rightarrow Z$

o 1 o

2 4 4 4

(22) $\epsilon+1 \Rightarrow \epsilon$

o o

9 9

(23) $(\epsilon = 1) \Rightarrow R$

1 o

9 o

(24) $Z \Rightarrow R$

10 1

$\square X(2,9)$ $\square X(2,9)$

Formulation of P4.52 in words :

(1) The given input of pair $\begin{matrix} V \\ o \end{matrix}$ represents the first stage of the list $\begin{matrix} Z \\ o \end{matrix}$.

(2) The list Z_{10} is empty at the start

(3) ϵ_o is zero at the start

(4) The first point of the first pair of Z_o results in Z_3. If Z_o is empty, then go to (23)

(5) Set $\epsilon_1 = 1$

(6) Set $Z_1 = \phi$ (empty list). In the beginning Z_1 is empty. In case of repetitions the hitherto formed elements of Z_1 must be cancelled.

(7) The next not yet considered element of the list Z_3 result in Z_5.
If no such element exists, then go to (4)

(8) Set $Z_2 = \phi$ (empty list).

(9) Select out of the set of pairs the next pair contained in Z_o, but not contained in Z_1, in which one element is equal to Z_5. This results in Z_4. If no such element exists then go to (7)

(10) If the front element of Z_4 is equal to the back element, then set R_3 positive Add Z_4 to the list Z_4 (if this is still empty, then Z_4 represents the first element). Go to (9)

(11) That element of the pair Z_4 which is not equal to Z_4, results in Z_6.

(12) If an element which is equal to Z_6 exists in the list Z_2, then Z_7 becomes positive; otherwise it is negative.

(13) If an element equal to Z_6 exists in the list Z_3, then Z_8 becomes positive; otherwise it is negative.

(14) If Z_7 is positive, then
R_5 becomes positive
and Z_4 supplemented by ϵ_o results in the next element of R_6.

(15) If Z_7 is negative, then add Z_6 to the list Z_2

(16) If Z_8 is positive then increase ϵ_1 by 1

(17) It is a necessary condition for R_o, that Z_8 is negative

(17a) If Z_8 is negative, then add Z_6 to the list Z_3.

(18) Add Z_4 to the list Z_1
Go back to (7)

(19) The pair of values ϵ_o, ϵ_1 results in the next element of R_2

(20) Supplement the elements of the list Z_1 by ϵ_o and add them to the list Z_{10}

(21) The remaining list of Z_o, without the elements contained in Z_1, is the new list Z_o

(22) Increase ϵ_o by 1
Go back to 4

(23) It is a necessary condition for R_o, that ϵ_1 is equal to 1.

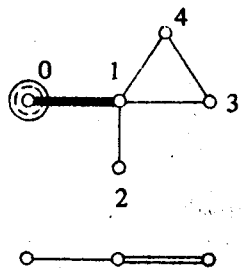
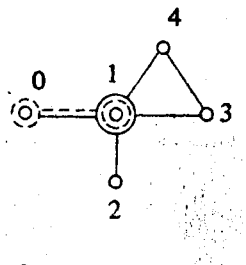
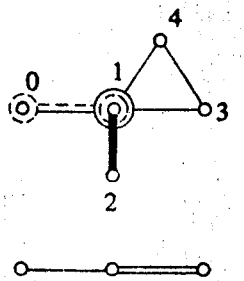
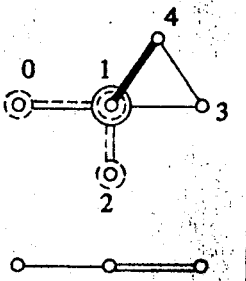
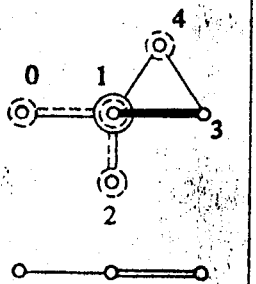
(24) Z_{10} results in R_1

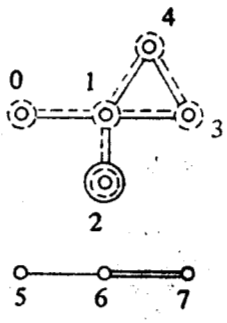
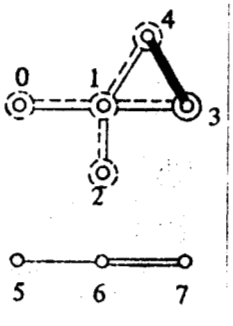
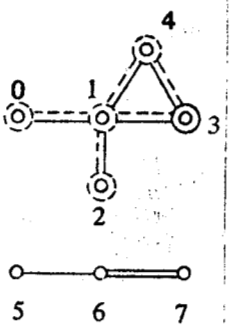
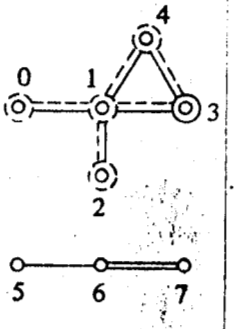
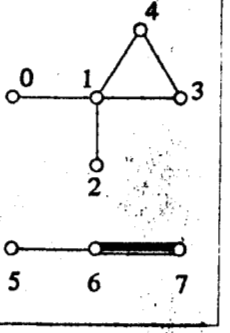
	Op.		Z_o	Z_1	Z_{10}	Z_2	Z_3				
1	10–	1	0–1	6 – 2 –	8 –	4	0	R_o		Z4	0–1
2	11		6–7					R_2		5	7 0
3	12		3–3							6	11 1
4	13		1–2					R_3		7	12 –
5	14–		5–6					R_4		8	13 –
6			7–6								
7			3–4					R_5			
8			1–4					R_6		€o	3 0
9			1–3							1	5 1
15			0–1	18 0–1	15	1	17a 0	R_o	17+	Z4	
16–			6–7				1	2		5	7 1
17	17a		3–3							6	
18			1–2					3		7	
9–			5–6					4		8	
7			7–6								
			3–4					5			
			1–4					6		€0	
			1–3							1	
8			0–1	0–1	8–		0	R_o	+	Z4	9 1–2
9			6–7				1	2		5	1
10–			3–3							6	11 2
11			1–2					3		7	12 –
12			5–6					4		8	13 –
13			7–6								
14–			3–4					5			
			1–4					6		€0	0
			1–3							1	1
15			0–1	0–1	15	2	0	Ro	17+	Z4	1–4
16			6–7	18 1–2			1	2		5	1
17	17a		3–3				17a 2			6	11 4
18			1–2					3		7	12 –
9			5–6					4		8	13 –
10–			7–6								
11			3–4					5			
14–			1–4					6		€0	0
			1–3							1	1
15	14–		0–1	0–1		2	0	Ro	17+	Z4	1–3
16–			6–7	1–2	15	4	1	2		5	1
17	17a		3–3	18 1–4			2			6	11 3
18			1–2				17a 4	3		7	12 –
9			5–6					4		8	13 –
10			7–6								
11			3–4					5			
12			1–4					6		€0	0
13			1–3							1	1

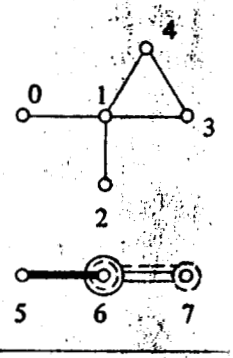
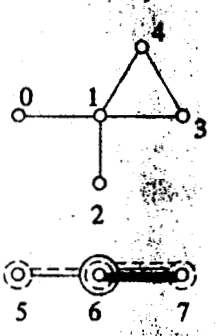
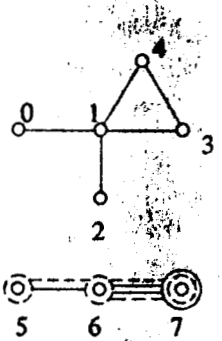
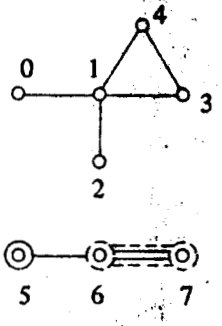
Op.		Z_o		Z_1		Z_{10}		Z_2		Z_3								
15		0–1		0–1				2		0	Ro	17	+	Z4				
16–		6–7		1–2				4		1	2			5		7	2	
17	17a	3–3		1–4			15	3		2				6				
18		1–2	18	1–3						4	3			7				
9–		5–6							17a	3	4			8				
7		7–6																
		3–4									5							
		1–4									6			ϵ0			0	
		1–3												1			1	
8	14	0–1		0–1			8	–		0	Ro	17	–	Z4		9	3–4	
9–		6–7		1–2						1	2			5		7	4	
7		3–3		1–4						2				6		11	3	
8		1–2		1–3						4	3			7		12	–	
9		5–6								3	4			8		13	+	
10–		7–6																
11		3–4									5							
12		1–4									6			ϵ0			0	
13		1–3												1		16	2	
15		0–1		0–1			15	3		0	Ro		–	Z4		9	3–3	
16		6–7		1–2						1	2			5				
17		3–3		1–4						2				6				
17a		1–2		1–3						4	3	10	+	7				
18		5–6	18	3–4						3	4	10	3–3	8				
9		7–6																
10		3–4									5							
9–		1–4									6			ϵ0			0	
		1–3												1			2	
7		21 6–7		0–1	20	0–1,0	8	–		0	Ro		–	Z4				
8		5–6		1–2		1–2,0				1	2	19	0,2	5		7	3	
9–		7–6		1–4		1–4,0				2				6				
7–				1–3		1–3,0				4	3		+	7				
19				3–4		1–4,0				3	4		3–3	8				
20																		
21										5								
22										6				ϵ0		22	1	
														1				
4		6–7	6	–		0–1,0	8	–	4	6	Ro		–	Z4		9	6–7	
5		5–6				1–2,0					2		0,2	5		7	6	
7		7–6				1–4,0								6		11	7	
8						1–3,0					3		+	7		12	–	
9						3–4,0					4		3–3	8		13	–	
10–																		
11											5							
14–											6			ϵ0			1	
														1		5	1	

	Op.	Z_o		Z_1	Z_{10}		Z_2		Z_3						
15	16	6–7	18	6–7	0–1,0	15	7				Ro	–	Z4	9	–
16	14–	5–6			1–2,0				17a	7	2	0,2	5		6
17–		7–6			1–4,0								6	11	5
17a					1–3,0						3	+	7	12	–
18					3–4,0						4	3–3	8	13	–
8															
10–											5				
11											6		ϵ0		1
12													1		1
15	13	6–7		6–7	0–1,0		7			6	Ro	–	Z4	9	7–
16–	14	5–6	18	5–6	1–2,0	15	5			7	2	0,2	5		6
17–	15–	7–6			1–4,0				17a	5			6	11	7
17a	16				1–3,0						3	+	7	12	+
18	17–				3–4,0						4	3–3	8	13	+
9	17a														
10–											5	14 +			
11											6	14 7–6,1	ϵ0		1
12													1	16	2
18		6–7		6–7	0–1,0	8	–			6	Ro	–	Z4		
9–		5–6		5–6	1–2,0					7	2	0,2		7	7
7		7–6	18	7–6	1–4,0					5	6				
8					1–3,0						3	+	7		
9–					3–4,0						4	3–3	8		
											5	+			
											6	7–6,1	ϵ0		1
													1		2
7		6–7		6–7	0–1,0	8	–			6	Ro	–			
8		5–6		5–6	1–2,0					7	R2	0,2		7	5
9–		7–6		7–6	1–4,0					5	9	1,2			
7–					1–3,0										
19					3–4,0						R3	+			
20				20	6–7,1						R4	3–3			
21–				20	5–6,1										
4–				20	7–6,1						R5	+	ϵ0	22	2
23–											R6	7–6,1	1		2
24					24=R_1										

die restlichen Seiten von "Chapter 2" des Plan Kalküls

Op.	Z_0	Z_1	Z_{10}	Z_2	Z_3					
① ⑩–	1 0-1	⑥ –	② –	⑧ –	④ 0	R_0		ZA	0–1	
② ⑪	6–7					R_2		⑤	⑦ 0	
③ ⑫	3–3							6	⑪ 1	
④ ⑬	1–2					R_3		7	⑫ –	
⑤ ⑭–	5–6					R_4		8	⑬ –	
⑥	7–6									
⑦	3–4					R_5				
⑧	1–4					R_6		€0	③ 0	
⑨	1–3							1	⑤ 1	
⑮	0–1	⑱ 0–1		⑮ 1	⑰a 0	R_0	⑰+	ZA		
⑯–	6–7				1	2		⑤	⑦ 1	
⑰ ⑰a	3–3							6		
⑱	1–2					3		7		
⑨–	5–6					4		8		
	7–6									
	3–4					5				
	1–4					6		€0		
	1–3							1		
⑧	0–1	0–1		⑧–	0	R_0	+	ZA	⑨ 1–2	
⑨	6–7				1	2		⑤	1	
⑩–	3–3							6	⑪ 2	
⑪	1–2					3		7	⑫ –	
⑫	5–6					4		8	⑬ –	
⑬	7–6									
⑭–	3–4					5				
	1–4					6		€0	0	
	1–3							1	1	
⑮	0–1	0–1		⑮ 2	0	Ro	⑰+	ZA	1–4	
⑯	6–7	⑱ 1–2			1	2		⑤	1	
⑰ ⑰a	3–3				⑰a 2			6	⑪ 4	
⑱	1–2					3		7	⑫ –	
⑨	5–6					4		8	⑬ –	
⑩–	7–6									
⑪	3–4					5				
⑭–	1–4					6		€0	0	
	1–3							1	1	
⑮ ⑭–	0–1	0–1		2	0	Ro	⑰+	ZA	1–3	
⑯–	6–7	1–2		⑮ 4	1	2		⑤	1	
⑰ ⑰a	3–3	⑱ 1–4			2			6	⑪ 3	
⑱	1–2				⑰a 4	3		7	⑫ –	
⑨	5–6					4		8	⑬ –	
⑩–	7–6									
⑪	3–4					5				
⑫	1–4					6		€0	0	
⑬	1–3							1	1	

Op.	Z_o	Z_1	Z_{10}	Z_2	Z_3					
(15)	0–1	0–1		2	0	Ro	(17) +	(Z4)		
(16–)	6–7	1–2		4	1	2		5	7 2	
(17) (17a)	3–3	1–4		(15) 3	2			6		
(18)	1–2	(18) 1–3			4	3		7		
(9–)	5–6				(17a) 3	4		8		
(7)	7–6									
	3–4					5				
	1–4					6		ϵ0	0	
	1–3							1	1	
(8) (14)	0–1	0–1		(8) –	0	Ro	(17) –	(Z4)	(9) 3–4	
(9–)	6–7	1–2			1	2		5	(7) 4	
(7)	3–3	1–4			2			6	(11) 3	
(8)	1–2	1–3			4	3		7	(12) –	
(9)	5–6				3	4		8	(13) +	
(10)	7–6									
(11)	3–4					5				
(12)	1–4					6		ϵ0	0	
(13)	1–3							1	(16) 2	
(15)	0–1	0–1		(15) 3	0	Ro	–	(Z4)	(9) 3–3	
(16)	6–7	1–2			1	2		5		
(17)	3–3	1–4			2			6		
(17a)	1–2	1–3			4	3	(10) +	7		
(18)	5–6	(18) 3–4			3	4	(10) 3–3	8		
(9)	7–6									
(10)	3–4					5				
(9–)	1–4					6		ϵ0	0	
	1–3							1	2	
(7)	(21) 6–7	0–1	(20) 0–1,0	(8) –	0	Ro	–	(Z4)		
(11)	5–6	1–2	1–2,0		1	2	(19) 0,2	5	(7) 3	
(9–)	7–6	1–4	1–4,0		2			6		
(7–)		1–3	1–3,0		4	3	+	7		
(19)		3–4	1–4,0		3	4	3–3	8		
(20)										
(21)					5					
(22)					6			ϵ0	(22) 1	
								1		
(4)	6–7	(6) –	0–1,0	(8) –	(4) 6	Ro	–	(Z4)	(9) 6–7	
(5)	5–6		1–2,0			2	0,2	5	(7) 6	
(7)	7–6		1–4,0					6	(11) 7	
(8)			1–3,0			3	+	7	(12) –	
(9)			3–4,0			4	3–3	8	(13) –	
(10–)										
(11)						5				
(14–)						6		ϵ0	1	
								1	(5) 1	

Op.	Z_o	Z_1	Z_{10}	Z_2	Z_3					
⑮ ⑯	6–7	⑱ 6–7	0–1,0	⑮ 7		Ro	–	Z4	⑨ 5–6	
⑯– ⑭–	5–6		1–2,0		⑰a 7	2	0,2	5	6	
⑰–	7–6		1–4,0					6	⑪ 5	
⑰a			1–3,0			3	+	7	⑫ –	
⑱			3–4,0			4	3–3	8	⑬ –	
⑧										
⑩–						5				
⑪						6		ϵ0	1	
⑫								1	1	
⑮ ⑬	6–7	6–7	0–1,0	7	6	Ro	–	Z4	⑨ 7–6	
⑯– ⑭	5–6	⑱ 5–6	1–2,0	⑮ 5	7	2	0,2	5	6	
⑰– ⑪–	7–6		1–4,0		⑰a 5			6	⑪ 7	
⑰a ⑯			1–3,0			3	+	7	⑫ +	
⑱ ⑰–			3–4,0			4	3–3	8	⑬ +	
⑨ ⑰a–										
⑩–						5	⑭ +			
⑪						6	⑭ 7–6,1	ϵ0	1	
⑫								1	⑯ 2	
⑱	6–7	6–7	0–1,0	⑧ –	6	Ro	–	Z4		
⑨–	5–6	5–6	1–2,0		7	2	0,2	5	⑦ 7	
⑦	7–6	⑱ 7–6	1–4,0		5	6				
⑧			1–3,0			3	+	7		
⑨–			3–4,0			4	3–3	8		
						5	+			
						6	7–6,1	ϵ0	1	
								1	2	
⑦	6–7	6–7	0–1,0	⑧ –	6	Ro	–			
⑧	5–6	5–6	1–2,0		7	R2	0,2		⑦ 5	
⑨	7–6	7–6	1–4,0		5	⑨	1,2			
⑦–			1–3,0							
⑲			3–4,0			R3	+			
⑳		⑳	6–7,1			R4	3–3			
㉑–		⑳	5–6,1							
④–		⑳	7–6,1			R5	+	ϵ0	㉒ 2	
㉓–						R6	7–6,1	1	2	
㉔			24=R_1							

Lists of input – and result –values :

V_o = [0–1, 6–7, 3–3, 1–2, 5–6, 7–6, 3–4, 1–4, 1–3]

R_1 = [0–1,0, 1–2,0, 1–4,0, 1–3,0, 3–4,0, 6–7,1, 5–6,1, 7–6,1]

R_o = –

R_2 = [0,2 ; 1,2]

R_3 = +

R_4 = 3–3

R_5 = +

R_6 = 7–6,1

Explanation of the results :

The list of pairs is not univocally coherent (R_o)

The list consists of the following coherent groups (R_1)

Group 0 : [0–1, 1–2, 1–4, 1–3, 3–4]

Group 1 : [6–7, 5–6, 7–6]

Group 0 and Group 1 are of the degree of determination 2 (R_2) .

A symmetrical pair (R_3) exists there, namely 3–3 .

A duplicate pair (R_4) exists there, namely 7–6 (R_6) .

Appendix to chapters 1 and 2

1) Legend of the letters

		explained
A	Type of data	44
B	Limitation –Symbol	46
C	Constant	50
E	Existance Operator	68
F, G	Function–Symbols, General	50/65
I,i	Index	65
j	Index	47
K	Component	67
L	List–Symbol	46
m, n	Number of Components of Data	46, 106 (107)
N	Number of Elements of a List	50
P	Program –Symbol	57
R	Result – Value	46
S	Structure	47
T	Type of Data	53
U	Subprogram	50
V,v	Variable	62
W	Repetitive Program	66
x, y	Bound Variables, also General Variables	50
Z	Intermediate Value	

α	Symbol for Variable Types of Data	56
ϵ	Currently Varying Auxiliary Value of a Program	64
κ	Variable Index for Components	59
λ	" The last "	75
μ	" The Next "	73

Π	Variable Program Symbol	
σ, τ	Variable Structure Symbol	
Π	Product	
Σ	Sum	

2) Legend of the Symbols

$\square$	General Symbol for Vacant Position	
$\vee$	Disjunction	In the sense of the calculus of propositions
$\wedge$	Conjunction	
$\bar{x}$	Negation	
$\rightarrow$	Implication	
$\curlyvee$	Disvalence	
$\sim$	Equivalence	
$\dot{\rightarrow}$	Conditional Program–Symbol	

$\Rightarrow$, $\approx$, $\approx$, $\approx$ multiple operations for chains of propositions.

$+$	Addition	In the sense of arithmetics
$-$	Subtraction	
$\times$	Multiplication	
$:$	Division	
$\sqrt{}$	Squareroot	
$=$	Equal–Symbol	
$\equiv$	Identity–Symbol	
$\Rightarrow$	" Results in " Symbol	
Θ	Same composition	
$=_{Df}$	Definition Symbol	
$<$	Less than	
$>$	Greater than	
$\leqslant$	Less or equal	
$\geqslant$	Greater or Equal	

o	Indifference	48 —
Δ	Symbol for a Set of Programs	45
∅	Empty List	41 —
∂	Variable Operation Symbol	54 —
∈	" Is Element of "	68 —
∪	Union of sets	115 + 1
∩	Section of sets	
V	Disjunction Element of	79
Λ	Conjunction Element of	79
( ) [ ]	Brackets	2 2
|	Symbol for Separation of Expressions	
,	Symbol for Separation of Components	48 —
⌈⌉ ⌊	Shifting of Lines	59, 80 —
x̂	Those Which (without repetition)	40 —
x̂	Those Which (with repetition)	41 —
x́	That one Which	42 —
⊢	Statement Symbol	81 + 1
⊖	General Negation	86 + 1
⋀	General Conjunction	84 + 1
⋁	General Disjunction	83 + 1

3) Legend of Generally Valid Function Symbols

Ca	Field of a Relation (List of Pairs)	– 123 + 1
Fin	End–Symbol	51 –
Fpos	Positive Value of a Function	153 —
Ger	Even Number	42, 151 —
Gz	Integer Number	151 —
Lz	Concatenation of Lists etc.	104, 110 —
Maj	The Greater One	58, 93
Max	The Greatest	100
Min	The Lesser One, the Smallest	92, 100
Nb	Back Area of a Relation	120
Nr	Numbering of the Elements of a List	110
Ord 0	Ordering of a Pair	92
Ord 1	Ordering of a List	100

Ord 2 to Ord 6	Ordering of a List of Pairs		
Pos	Positive		
Pz	Subprogram		
Qz	Horizontal Composition		
Rz	Result of a Subprogram		
Sign	Sign of a Number		
Sp	Column of a List of Pairs		
T11 T12	Partial List		
Ub	Transfer		
Vb	Front Area of a Relation		

4) Legend of General Structure Symbols

So	Yes–No–Value
S1.n=nXSo	Series of n Yes –No–Values
S2=2Xσ	Pair of Data
S3=mXσ	List
S4=mX2σ	List of Pairs

A8	General Notation for Numbers	Chapter 3,
A9	Positive Integer Number	
A10	Integer Number	
A12	Rational Number	
A13	Complex Number	

5) Order of Priority of the Symbols in Compliance with their Range

⟶ ⇒	≈	→	∧ ∨	=	+ X
				< – :	
				>	
				∈	

⟶ stronger binding

Chapter 3

Programs for Arithmetic Operations

Contents

I. Structures and Types of Numbers

The structures and types of numbers in arithmetic operations are of a great variety. Without considering their representation in detail , the following types of numbers can be distinguished form the start :

1) Scalars

   a) Integer positive Number (natural number)

   b) integer positive or negative number

   c) rational positive number

   d) rational positive or negative number

   The investigation of irrational numbers is superfluous, since no structure allows them to be exactly. They must always be approximated by rational numbers.

2) Composite Values :

   a) Complex numbers

   b) Vectors

All these types of numbers can be represented in very different ways. For instance, the following number systems can be distinguished :

1) Homogenuous number systems , e.g. :

   a) Binary system

   b) Decimal system

2) Non–homogeneous systems , e.g.

   a) Division of the circle into grades, minutes , and seconds

   b) Division of the time into hours, minutes , and seconds

   c) British measurement of distances (mile, yard, inch)

   d) Monetary systems (standards) , e.g. the British system.

Further, after the type of numbers and the number system are established, different ways of representation can be applied.

1) The identification of positive and negative numbers can be accomplished

   a) by signs

   b) by complement representation

2) To specify the order of magnitude , power factors can be added to numbers (half–logarithmic representation)

3) Representation by logarithms

4) Representation by fractions ($\frac{a}{b}$)

Finally, numbers may be supplemented by special data which specify those cases, which cannot be represented by the normal representation of the number e.g. :

1) Specification of " ∞ " or " very great "

2) Specification of " exactly zero " (in half – logarithmic and logarithmic representation)

3) Specification of " indeterminate "

(see chapter IV)

Out of this multitude of possibilities types of numbers of general and of special meaning can be developed.

Of general meaning are e.g. the basic types of numbers (positive integer numbers etc.)

Of special importance are , e.g. the numbers for the division of the circle, or monetary systems. Amounts in Mark–currency for instance are represented by decimal numbers, with two digits after the point. Of further special importance are the number–systems in a special computer (e.g. in the Z4) .

First, only programs for number–systems of general importance are developed.

The following types of data are defined :

A8	Number in the general meaning
A9	Positive integer number
A10	Positive or negative integer number
A11	Rational positive number
A12	Rational positive or negative number
A13	Complex number

These general symbols may be substituted in different representations by various structure symbols, e.g. those for decimal numbers, or for binary numbers.

It is not possible , however, to mention all possible strucutres, because of their multitude (the set of possible number–systems is infinite) .

Consequently, at any time , only the immediately required strucutres are specially defined. The following special forms of the types of numbers listed above are established :

A9.2	Positive integer binary number
A9.10	Positive integer decimal number
A10.2	Integer positive or negative binary number (kind of specification of negative numbers not fixed)
A10.2.0	As A10.2 , but by complement in representation
A10.2.1	As A10.2. , but in representation with sign
A10.10	Integer positive or negative decimal number
A10.10.1	According to A10.2.0 , A10.2.1
A10.10.2	

Definition of structures:

A9.2	=	S1.n	
A9.10	=	nXS1.4 (the single decimal digits are represented by S1.4)	
A10.2.0	=	S1.n	
A10.2.1	=	(So, S1.n)	Ko = sign K1 = number
A10.10.0	=	nXS1.4	
A10.10.1	=	(So, A8.10)	
A11.2	=	(S1.m, S1.n)	Ko = series of digits before the point K1 = series of digits after the point
A11.10	=	(mXS1.4, nXS1.4)	Ko , K1 , according to A10.2
A12.2.0	=	(S1.m, S1.n)	
A12.2.1	=	(So, A10.2)	
A12.10.0	=	(mXS1.4, nXS1.4)	
A12.10.1	=	(So, A10.10)	
A13	=	2XA12	

Other structures are defined ,when required.

II. Introduction

As manifold as the types of numbers and their representation, are the programs with numbers.

Programs with numbers mostly correspond to arithmetic operations. With nearly all types of numbers analog operations are possible , e.g. addition. Therfore, the well–known operation–symbols of arithmetics will be used here, too. They represent the symbols of a set of analog programs. The program for addition for instance is different, depeding on the structure of the operands. The special program is a function of this structure. So generally, the operation–symbol suffices.

But a symbol for the program–set is assigned to each operation–symbol. If necessary, the specification of a special program of the set can be achieved by supplementary data or indices, which normally correspond to the structure–symbol.

Nevertheless, various programs are feasible for the same structure, (e.g. those with or without indication of an overflow) .

Various programs may be equivalent or quasi–equivalent. Even if the structure of the operands is the same, in various cases still, different methods of multiplication are possible (e.g. first or second factor can be used as multiplicator) . If the results are exactly the same with all these variations of the input data, then the programs are equivalent. If they differ slightly in accuracy

then they are quasi–equivalent. In the latter case it can be assumed that these differences approach zero when the number of digits increase .

The complete program for an operation sometimes contains supplementary information, e.g. the identication of an overflow, besides the computation of the proper result (e.g. the sum) In such cases the specification of the values must be performed by program–symbols or result –symbols (for instance R8.10) .

The real arithmetic operations only represent in this case a reduced part of the total program. Such reductions are also possible with regard to the number of the digits of the result, for instance .

In the following section programs of homogeneous structure are developed. In those programs only data or numbers of the same structure appear. Nevertheless , the results can be propositions (for instance : $a > b$).

The following groups are distinguished :

1) Propositions on single numbers

2) Operations with single numbers

3) Propositions on pairs of numbers

4) Operations with pairs of numbers

5) Operations with sets of numbers

This grouping does not correspond to a systematic logical development of one operation into another. In some programs other groupings are used , which are defined only later.

First, we refrain from an axiomatic representation here. The well–known systems of axioms like those of Peano represent implicit solutions of numbers and their operations, which may be realized by different structures and programs. In order to develop the rules of special number–systems form those general axioms. the introduction of additional axioms would be necessary.

The numbering of the programs is generally related to the group 8, (e.g. P8.3) . The figure 8 can be replaced by the numbers 8, 10, 11, 12 , 13 on the types of the numbers used.

Survey of the programs with numbers

1) Propositions on numbers

Program identification	Operation symbol	Marginal values	Remarks	Page
P8.0	Pos (V) o 8	R (V) ⇒ R o 8 o	V_o is positive or equal to zero	
P8.1	Gz (V) o 8	"	V_o is an integer number	
P8.2	Ger (V) o 8	"	V_o is an even number	155
P8.3	V = 0 o 8	"		155
P8.4		"	V_o is an integer power of 2	155

In stead of the program–symbol P8.□ we have beginning with page 155 the special program–symbol P9.□

2) Operations with One Operand

Program identification	Operation symbol	Marginal values	Remarks	Page
P8.8	V + 1 ⇒ R	R (V)⇒ (R, R)	R = signal	155
V	o o	o o 1	1 "end–carry–over"	
A	8 8	8 8 o		
P8.9	V − 1 ⇒ R	"	"	156
V	o o			
A	8 8			
P8.10	V × 2 ⇒ R			156
	o o	"	"	
	8 8			
P8.11	V × $\frac{1}{2}$ ⇒ R			156
	o o	"	"	
	8 8			
P8.12	V × 10⇒ R			157
	o o	"	"	
	8 8			
P8.13	V × $\frac{1}{10}$ ⇒ R			—
	o o	"	"	
	8 8			
P8.16	V^2 ⇒ R			157
	o o	"	"	
	8 8			
P8.17	1 : V ⇒ R			—
	o o	"	"	
	8 8			
P8.18	$\sqrt[2]{V}$ ⇒ R			157
	o o	"	"	
	8 8			
P8.19	$\sqrt[3]{V}$ ⇒ R			—
	o o	"	"	
	8 8			

Program identification	Operation symbol	Marginal values	Remarks	Page
P8.22	V × (– 1) ⇒ R	R (V) ⇒ R	Inversion of sign	
V	o o	o o		
A	8 8	8 8		
P8.23	| V | ⇒ R	"	Absolute value	
V	o o			
	8 8			
P8.24	sign (V) ⇒ R		V ⩾ 0 → R = +1	
V	o o		o o	
A	8 10		V < 0 → R = –1	
			o o	
P8.25	Fpos (V) ⇒ R	"	V ⩾ 0 → R = V	
V	o o		o o o	
A	8 8		V < 0 → R = 0	
			o o	
P8.26		"	Extension ~~Increase~~ of number of digits	154
P8.27		"	Conversion to even number of digits	
P8.28		"	Formation of the next lower integer number	
P8.29		"	Formation of the next greater integer number	
P8.30		"	Formation of the next even number	

3) Propositions on pairs of numbers

Program identification	Operation symbol	Marginal values	Remarks	Page
P8.48	V = V	R (V, V) ⇒ R		153
V	o 1	o 1 o		
A	8 8	8 8 o		
P8.49	V < V	"		158
V	o 1			
A	8 8			
P8.50	V ⩽ V			158
V	o 1	"		
	8 8			

4) Operations with two operands

P8.64	V + V ⇒ R	R (V, V) ⇒ (R, R)	R_1 signal :	158
V	o 1 o	o 1 o 1	increase of digit position	
A	8 8 8	8 8 8 o		
P8.65	V − V ⇒ R		R_1 " result less than	158
V	o 1 o	"	zero	
A	8 8 8			
P8.66	V − V ⇒ R			159
V	1 o o	"	"	
A	8 8 8			
P8.67	V × V ⇒ R		R_i = signal	159
V	o 1 o	"	increase of digit	
A	8 8 8		positions	
P8.68	V : V ⇒ R			160
V	o 1 o	"	"	
A	8 8 8			
P8.69	Maj (V, V) ⇒ R	R (V, V) ⇒ R	The greater of two	161
V	o 1 o	o 1 o	values	
A	8 8 8	8 8 8		
P8.70	Min (V, V) ⇒ R	"	The lesser of two	161
V	o 1 o		values	
A	8 8 8			
P8.72	$V_o \times B_1^V \Rightarrow R$	R (V, V) ⇒ R	B = Base of the number–	161
		o 1	system	
		8 8		

III. Programs with Positive Integer Binary Numbers (A9.2)

1) Propositions on simple numbers

	R (V) ⇒ R
V	o o
S	1.n o

P9.2 even number

A9.2

	$\overline{V} \Rightarrow R$
V	o o
K	o
S	o o

P9.3 V = 0

A9.2 o

	~~A~~ ~~O~~ (V) ⇒ R
V	o o
S	1.n o

P9.4 integer power of 2

	R1.9 (V) ⇒ R
V	o o o
S	o 1.n o

2) Operations with one Operand

P9.8 Counting forward

A9.2 R = signal „ increase of digit positions "
1

	R (V) ⇒ (R , R)
V	o o 1
S	1.n 1.n+1 o

	+ ⇒ Z	W1 (n)	V ⇒ Z	Z ∧ Z ⇒ R	Z ∧ Z ⇒ Z
V	o		o 1	o 1 o	o 1 o
K			i	i	
S	o		o o	o o o	o o o

	Z ⇒ R	Z ⇒ R
V	o o	o 1
K	n+1	
S	o o	o o

P9.9. Counting backward

A9.2 R = signal „ change of sign " .
1

	R (V) ⇒ (R , R)
V	o o 1
S	1.n 1.n o

	– ⇒ Z	W1 (n)	V ⇒ Z	Z ~ Z ⇒ R	Z ∨ Z ⇒ Z
V	o		o 1	o 1 o	o 1 o
K			i	i	
S	o		o o	o o o	o o o

	Z̄ ⇒ R
V	o 1
S	o o

P9.10 Dublication

A9.10 (Marginal values and R as for P9.8)
1

	– ⇒ R	W1 (n)	V ⇒ R	V ⇒ R
V	o		o o	o 1
K	o		i i+1	n–1
S	o		o o	o o

P9.11 Bisection ,

A9.2 (Mariginal values as for P9.9)
R = signal „ remainder "
1

	– ⇒ R	W4 (n,1)	V ⇒ (R	V ⇒ R
V	o		o o	o 1
K	n–1		i i–1	o
S	o		o o	o o

P9.12 X 10 .

A9.2

	R (V) ⇒ R , R)
V	o o 1
S	1.n 1.n+4 o

Implicit representation

(V X 2 X 2 + V) X 2 ⇒ R
o o o

R = sign „ increase of digit position "
1

	R9.10	(	V	)	⇒	(	Z	,	Z	)	Z	⇒	V	R
V	o		o				o		1		1		1	
S			1.n				1.n+1		o		o		o	

	R9.10	(	Z	)	⇒	(	Z	,	Z	)	Z	⇒	V	R
V	o		o				o		1		1		1	
S			1.n+1				1.n+2		o		o		o	

	R9.64	(	Z	,	(	V	,	–	)	)	⇒	(	Z	,	Z	)
V			o			o							o		1	
S			1.n+2			1.n+2							1.n+3		o	

	Z	⇒	V	R	R9.10	(	Z	)	⇒	(	R	,	Z	)	Z	⇒	V	R
V	1		1				o				o		o		1		1	
S	o		o				1.n+3				1.n+4		o		o		o	

P9.16 $V^2 \Rightarrow R$

A9.2

	R	(	V	)	⇒	R	,	R
V			o			1		1
S			1.n			1.m		o

R = signal „ increase of digit position ” .
1

	R9.67	(	V	,	V	)	⇒	(	R	,	R
V			o		o				o		1
A			9.2		9.2				9.2		o

P9.18 Squareroot

The development of this program is omitted in the English version of the Plankalkuel. See German version .

P9.26 Extension of the number of digits

A9.2

	R	(	V	)	⇒	R		
V			o			o		
S			1.n			1.n+1		

	–	⇒	R	W1	(	n	)	[	V	⇒	R	]
V			o						o		o	
K			n+1						i		i	
S			o						o		o	

Example : V_o = LL

R_o = 0LL

The value of the number is not changed.

3) Propositions on pairs of numbers

In all programs with two or more numbers as input – values it is presupposed that the number of digits is the same with all numbers. If this is not the case, then the number with the smaller number of digits must be transformed to that of the greater one by repeated application of the rule P9.26.

P9.48 These programs have been anticipated by the programs P1.68,

P9.49 P1.72, and P1.74

P9.50

4) Operations with two operands

With regard to the number of digits see page

P9.64

A9.2

	V	+	V	⇒	R
V	o		1		o
S	1.n		1.n		1.n+1

R_1 = signal " increase number of digits "

	−⇒	z	W1(n)	V ↛ V ⇒ z			(V ∧ V) ∨ (z ∧ z) ⇒ z			z ↛ z ⇒ R		
V		o		o	1	1	o	1	1 o o	1	o	o
K				i	i	i	i	i	i i i+1	i	i	i
S				o	o	o	o	o	o o o	o	o	o

	z⇒	R	z	⇒	R
V	o	o	o		1
K	n	n	n		
S	o	o	o		o

P9.65

A9.2

	V	−	V	⇒	R
V	o		1		o
S	1.n		1.n		1.n

R_1 = signal " result negative '
In this case R is representing a supplement
o

	+⇒	z	W1(n)	V ~ V ⇒ z			(V ∧ $\overline{V}$) ∨ (z ∧ z) ⇒ z			z ↛ z ⇒ R		
V		o		o	1	1	o	1	1 o o	1	o	o
K				i	i	i	i	i	i i i+1	i	i	i

	z⇒	R
V	o	1
K	n	o

P9.66

A9.2

	V	– V	⇒ R	R as	R9.65
V	1	o	o	1	1
K	1.n	1.n	1.n		

As P9.65 , however, V_1 exchanged with V_o

P9.67 Multiplication

A9.2 In the general case of the structures , V_o as well as V_1 have to be transformed to the same number of digits

	V	X V	⇒ R
V	o	1	o
S	1.ℓ	1.m	1.n

The normal case then reads :

	V	X V	⇒ R	R = signal " overflow "
V	o	1	o	1
S	1.n	1.n	1.n	o

Implicit representation of R_o

	ΣUb (V, V) · 2^i	⇒ R	Ub ( ) see chapter 2, page 89
V	i 1 o	o	
K	i		

Explicit form

	0 ⇒ z	V ⇒ z
V	o	1 1

	W1 (n) [z + Ub (V, V) ⇒ z	z X 2 ⇒ z	z ⇒ R]
V	o 1 o o	1 1	o o
K	i		

P9.67 with signal

```
                    |  R ( V   , V   ) ⇒ ( R   , R )
                  V |      o     1         o     1
                  S |      1.n   1.n       1.n   o

    | 0 ⇒ z | V   ⇒ z
  V |     o | 1     1
  S |   1.n | 1.n   1.n

    | W1 (n) | R9.64 ( z   ,  Ub ( z , V ) ⇒ ( z   , V z )) | R8.10 ( z ) ⇒ ( z ,  Vz )
  V |        |         o        o   o         o     2      |       1       1      2
  K |        |                      i                     |
  S |        |       1.n        1.n o         1.n   o      |       1.n     1.n

    | z   ⇒ R   z ⇒ R
  V | o     o   2   1
  S | 1.n   1.n o   o
```

P9.68 Division

```
    | V   : V ⇒ R                R = Remainder
  V | o     1   o                  1
  S | 1.n   1.n 1.n                o
```

Implicit representation

```
Max | x̂ ( V X x ≤ V ) ⇒ R | ( V X R ≠ 0 ) ⇒ R
    |   1       o     o  |   1   o         1
```

Explicit form

```
V ⇒ z | V ⇒ z , | −1 ⇒ ϵ | 0 ⇒ z
o   o | 1   1  |         |     3
```

```
W [ z > z ⇒ [ z X 2 ⇒ z | ϵ + 1 ⇒ ϵ ] ]
  [ o   1   [ 1       1 |           ]
```

```
    | W [ z − z ⇒ z | z ≥ 0 ⇒ [ z ⇒ z | + ⇒ z ] ]
  V |   [ o   1   2 | 2       [ 2   o |   3   ]
  K |   [                             |   ϵ   ]
    |   [                             |       ]
    |   [ ( z = 0 ) ∨ ( ϵ = 0 ) ⇒ Fin²        ]
```

```
z ⇒ R | z ≠ 0 ⇒ R
3   o | 2       1
```

Legend of the values :

V_o = Dividend

V_1 = Divisor

R_o = V_o : V_1

R_1 = " Remainder "

Z_o = Current remainder of the dividend

Z_1 = Divisor $\times 2^k$

Z_2 = $Z_o - Z_1$

Z_3 = Compiling value of the result

P9.69

A9.2 Maj (V , V) ⇒ R
o 1 o
1.n 1.n 1.n

V ⩾ V ⇒ z | z ↛ (V ⇒ R) | $\bar{Z}$ ↛ (V ⇒ R)
o 1 o | o o o | o 1 o

P9.70

A9.2 Min (V , V) ⇒ R
o 1 o
1.n 1.n 1.n

V ⩾ V ⇒ z | z ↛ (V ⇒ R) | $\bar{Z}$ ↛ (V ⇒ R)
1 o o | o 1 o | o o o

P9.72

A9.2

	V × 2^{V_1} ⇒ R
V	o o
A	9.2 9.2

Correct representation

	V × 2 ⌐V ⇒ R	
V	o 1 o	
S	9.2 9.2 9.2	
	R (V , V) ⇒ R ⌐1.(n + V)	
V	o 1 o ⌊ 1	
S	1.n 1.m ⌊ 1.m	

V ⇒ z | V ⇒ ϵ | W [ϵ ≠ 0 ↛ [R9.10 (z) ⇒ z | ϵ −1 ⇒ ϵ]]
o o | 1 | o o |

z ⇒ R
o o

IV. Programs with Positive and Negative Integer Binary Numbers

Representation by complement A10.2.0

Only examples are mentioned.

P10.0

	Pos (V) ⇒ R
V	o o
S	1.n .o

	$\overline{V}$ ⇒ R
V	o o
K	n–1
S	o o

P10.2 Ger (V) | P9.2

as

P10.3 $V_o = 0$ | P9.3

P10.4 " V_o is an integer power of 2

	[V ⇒ R1.9 (V)] ∧ [V ⇒ R1.17 (V))] ⇒ R10.4
V	[o o] ∧ [o o o] o
K	[n–1] ∧ [n–1]
S	[o 1.n] ∧ [o 1.n] o

Examples n = 3

ooo = o
oLo = +2
LLo = –2
Loo = –4

P10.8

V + 1 ⇒ R10.8 , R10.8 as R9.8
o o o o

	R ∧ $\overline{R}$ ⇒ R
V	o o o
K	n–1 n

P10.9

V_o – 1 ⇒ R10.9 (= R9.9)
o

	R ∧ R ⇒ R
V	o o 1
K	n–1 n 0

P10.18

	R (V) ⇒ (R , R)	R9.18 (| V |) ⇒ R	V ⇒ R
V	o o 1	o o o	o 1
A	10.2.0 9.2 o	9.2 1.n 9.2	n−1 o

$$R \underset{o}{\Rightarrow} \sqrt[2]{|V|}_{o} \quad , \quad R = \text{“ imiginary ”}_{1}$$

P10.22

	V × (−1) R10.8 (θ V) ⇒ (R , R)
V	o o o 1
S	1.n 1.n o

P10.23

$$|V|_{o}$$

	V → (V ⇒ R)	V → R10.8 (∉ (V)) ⇒ (R , R)
V	o o o	o o o o 1
K	n−1	n−1
S	o 1.n 1.n	o 1.n 1.n o

P10.24

	V ⇒ R	V ⇒ R
V	o o	o o
K	n−1	n−1

P10.25

$$Fpos(V)_{o} \quad . \quad Pos(V)_{o} \rightarrow (V \Rightarrow R)_{o\ o}$$

$$\overline{Pos}(V)_{o} \rightarrow (0 \Rightarrow R)_{o\ o}$$

P10.26 Increase of the number of digits by 1

	R (V) ⇒ R
V	o o
S	1.n 1.n+1

	W1 (n) [V ⇒ R]	V ⇒ R
V	o o	o o
K	i i	n−1 n
S	o o	o o

P10.72

$V \overset{V_1}{\Rightarrow} R$

	$R(V, V) \Rightarrow (R, R)$			
V	o	1	o	1
A	10.2.0	10.2.0	10.2.0	
S	1.n	1.m	1.n	o

The number of the digits did not increae. R_1 is the signal specifying that the given range of digits is not sufficient.

The ecpression is generally valid :

	$V \Rightarrow R \overset{(i+V)}{\ulcorner\!\!\lrcorner}$		
V	o	o	1
K	i		
S	o		

But i may only be varied to the extent, that the following is true :

$$0 \leqslant i + V_1 < n$$

This can be achieved by the following program :

	$W1(n)$	$\left[0 \leqslant i + V_1 < n \rightarrow \left[V \Rightarrow R \overset{(i+V)}{\ulcorner\!\!\lrcorner} \right]\right]$			
V			o	o	1
K			i		
S		1.m	o	o	1.m

The digits of R_o not yet specified by this program can be derived as follows :

a) If V_1 is greater then zero then the digits with the indices 0 to $V-1_1$ are set equal to zero.

	$V_1 > 0 \rightarrow (W1(V-1_1))\left[- \Rightarrow R \right]$		
V	1	1	o
K			i

Interpretation : " W-program 1 with the limit $V-1_1$ applied to [ ] ".

Since V_1 can be greater than n, then W1 must be limited by n−1 in this case. The general expression for the limitation of W1 is :

$$\text{Min}\ (V\text{-}1_1\ ,\ n-1)$$

Then the following expression results :

$$V_1 > 0 \rightarrow \left[W1(\text{Min}(V, n))\left[- \Rightarrow R \right]\right]$$

b) If V_1 is negative , then the digits with the indices R_{n-1} to R_{n-1+V_1} must be set equal to the highest digit of V_o . Because if this digit position is occupied by a one , this means that V_o is negative and that all higher unwritten digits must then also be equal to one. The corresponding expression reads :

$$V_1 < 0 \rightarrow (W1(|V_1|)_1) \begin{bmatrix} V_o \Rightarrow R_o \\ n-1 \quad n-1-i \end{bmatrix}$$

Here too , $|V_1|$ may be greater than n . Consequently, the expression must be supplement as follows :

$$V_1 < 0 \rightarrow \left| (W1(\mathrm{Min}(|V_1|+1, n))_1) \begin{bmatrix} V_o \Rightarrow R_o \\ n-1 \quad n-1-i \end{bmatrix} \right.$$

Finally , a criterion for R_1 has to be established :

R_1 can only become positive if $V_1 > 0$, and if one of the digits of V_o with the indices $n-1$ to $n-1-V_1$ is different form $V_{o\,n-1}$

(These are the digits which get lost by an upwards shift).

Then , the following expression becomes true :

$$V_1 > 0 \rightarrow \left[W1(\mathrm{Min}(V_1, n))_1 \begin{bmatrix} V_o \not\to V_o \Rightarrow R_o \\ n-1-i \quad n-1 \quad 1 \end{bmatrix} \right]$$

This expression can be combined with the expression on page 164 . Thus we obtain the following program :

```
P10.72
A10.2.0   |  R ( V   , V   ) ⇒ ( R   , R )
        V |    o     1        o     1
        A |    10.2.0 10.2.0   10.2.0
        S |    1.n    1.m      1.n   o

        V |  W1 ( n ) [ 0 ≤ i + V < n ⇒ [V ⇒ R [(i + V)]
        K |                  1          [o   o    1  ]
                                        [i   ]

        V |  V > 0 ⇒ [ ( W1 ( Min ( V, n ))) [ -⇒ R |  V   ↛ V   ⇒ R ]
        1 |                  1                [   o |  o     o     1 ]
        K |                                   [   i |  n-1-i n-1     ]

        V |  V < 0 ⇒ [ ( W1 ( Min ( |V| + 1 , n ) ) ) [ V   ⇒ R ] ]
        1 |                  1                       [ o     o ]
        K |                                          [ n-1   n-1-i ]
```

V. Operations with Positive Integer Decimal Numbers

1) Structure of the numbers

A9.10 = nXS1.4

The single components cprrespond to the decimal digits. These are represented by binary numbers with four digits.

0	oooo
1	oooL
2	ooLo
3	ooLL
4	oLoo
5	oLoL
6	oLLo
7	oLLL
8	Looo
9	LooL

Examples for decimal numbers : n = 4

	digit 3	digit 2	digit 1	digit 0
305 =	oooo	ooLL	oooo	oLol
1972 =	oooL	oLLL	LooL	ooLo

2) Operations with decimal numbers

P9.64 A9.10

		V + V ⇒ R	
	V	o 1 o	R_1 = signal " overflow "
	A	9.10 9.10 9.10	

		R (V , V) ⇒ (R , R)	
	V	o 1 o 1	
	S	nX1.4 nX1.4 (n+1)X1.4 o	

This sum is produced digit by digit. First for each position the sum of the digits $Z_{o.i}$ is evaluated form V_o and V_1 .

This sum is a binary number of 5 digits. If there is a carryover form the position i−1 to the position i, then $Z_{o.i}$ must be increased by one.

If the value $Z_{o.i}$ thus compacted is greater then oLooL (9) , then LoLo (10) must be subtracted and a carry−over $Z_{1.i+1}$ to the position Z_{i+1} must be effected.

There is no carry−over to the first position (− ⇒ $Z_{1.o}$)

We then obtain the following program

```
      - ⇒ z

  |  W ( n ) ⎡ V   + V   ⇒ z   | z  →⎡ z    + 1 ⇒ z   ⎤ ⎤
V |          ⎢ o     1     o   | 1   ⎢ o            o   ⎥ ⎥
K |          ⎢ i     i         |     ⎢                  ⎥ ⎥
S |          ⎢ 1.4   1.4   1.5 | o   ⎣ 1.5          1.5 ⎦ ⎥
             ⎢                                            ⎥
  |          ⎢ z  ≥ oLoLo  ⇒ z                            ⎥
V |          ⎢ o               1                          ⎥
S |          ⎢ 1.5             o                          ⎥
             ⎢                                            ⎥
  |          ⎢ z  →⎡ z   - LoLo  ⇒ z   ⎤ | z    ⇒   R     ⎥
V |          ⎢ 1   ⎢ o             o   ⎥ | o        o     ⎥
K |          ⎢     ⎢                   ⎥ |          i     ⎥
S |          ⎣ o   ⎣ 1.5           1.5 ⎦ | 1.5      1.4   ⎦

  | z  →⎡ 0 ⇒ R   ⎤ | z  →⎡ oooL ⇒ R   | + ⇒ R ⎤
V | 1   ⎢     o   ⎥ |     ⎢        1   |     1 ⎥
K |     ⎢     n   ⎥ |     ⎢        n   |       ⎥
S | o   ⎣     1.4 ⎦ | o   ⎣        1.4 |     o ⎦
```

In this formula the sub-indices of Z_o and Z_1 can be omitted because of the rule of the " results in " symbol. This formula can be varied under different aspects :

1) Instead of the subtraction of LoLo (10) an addition of the complement can be performed. The complement of LoLo is :

```
   .ooLoLo
   -------
   .LLoLoL
 +       L
   -------
   .LLoLLo
```

Since the difference Z_o – LoLo must have a value between 0 and 9 , only the lower four digits of the supplement are relevant.
Thus we can write

$$Z_o + \text{oLLo} \Rightarrow Z_o$$

2) Instead of the general operation–symbols the special program identification for the used type of numbers and number of digits can be substituted. In the first case ($V_{o,i} + V_{1,i}$) represents the addition of two binary numbers of four digits, resulting in a binary number of five digits. The used program is P9.64 for n = 4 .

We will call it P9.100

$$\text{P9.100} = \underset{\text{A9.2}}{\text{P9.64}} \quad , \; n = 4$$

The operation $Z_o + 1$ can be represented by $R_o\,9.8\,(Z_o)$.

The case Z_o + oLLo represents an addition of two binary numbers of four digits each, resulting in a number also of four digits. This is the reduced form of P9;100 without R_{o4}.

This program will be indentified by P9.101. .

P9.101= P1.100 with ,

	(R9.100,	R9.100,	R9.100,	R9.100) ⇒	R9.101
V	o	o	o	o	o
K	o	1	2	3	

From Z_o which was five digits , the value Z_2 with four digits is formed whereby the digit Z_4 is ommited . We then obtain the following program :

	− ⇒ z										
V	1										
S	o										
	W1 (n)	R9.100 (V	, V	) ⇒ z	z ⇒	R9.8 (z	) ⇒ z				
V		o	o	1	o	1	o	o	o		
K			i	i							
S		1.5	1.4	1.4	1.5	o	1.6	1.5	1.5		
		z	⩾ oLoLo ⇒	z	(z, z, z, z)	⇒ z					
V		o		1	o o o o	2					
K					o 1 2 3						
		1.5		o	o o o o	1.4					
		z ⇒	R9.101 (z	, oLLo)	⇒ z	z	⇒ R				
V		1	2		2	2	o				
K							i				
S		o	1.4		1.4	1.4	1.4				
	W3 (1,4)	− ⇒ R		z ⇒ R	z ⇒ R						
V		o		1 o	1 1						
K		n.i		n.o							
S		o		o	o o						

In this , another representation for R_{on} was chosen .

3) Now the operation R9.8 (Z_o) will be eliminated.

$(z_o \geqslant \text{LoLo}) \rightarrow (z_o + 1 > \text{LoLo})$

Which means, that in the case $V_{oi} + V_{1i} \geqslant$ LoLo, there is no influence

of $Z_{1.i}$ on the evaluation of $Z_{1.i+1}$

(In any case, there ia s carry–over $Z_{1.i+1}$ to the next higher position i+1) .

Further, the following applies :

$$z_o < LooL \rightarrow \overline{(z_o + 1) \geqslant LoLo}$$

Consequently in the case of $V_{oi} + V_{1i} <$ LooL there will never be a carry–over $Z_{1.i+1}$ to the next position.

To consider the influence of $Z_{1.i}$ on $Z_{1.i+1}$ therefore, only the case Z_o = LooL is relevant.

Then the following expression for Z_1 applies :

$$R9.100 (V_{oi}, V_{1i}) \Rightarrow z_o$$

$$(z_o \geqslant oLoLo) \vee (z_o = oLooL \wedge z_{1.i}) \Rightarrow z_{1.i+1}$$

R9.8 (Z_o) therefore, is not necessary for the evaluation of $Z_{1.i+1}$ anf for the evaluation of Z_2 it can be combined with the addition of LLo.

First we write :

$$z_{1.i} \rightarrow R9.101 (z_2 , oooL) \Rightarrow z_2$$

$$z_{1.i+1} \rightarrow R9.101 (z_2 , oLLo) \Rightarrow z_2$$

In this , Z_2 is altered not at all , once or twice depending on the values of $Z_{1.i}$ and $Z_{1.i+1}$.

Since the two values oooL and oLLo do not have a one in the same position the two operations in R9.101 can be combined.

$$R9.101 (z_2 , (z_{1.i} , z_{1.i+1} , z_{1.i+1} , -)) \Rightarrow z_2$$

In this representation the digits of the second value have to be written in reverse sequence.

Since Z_1 is a control value, an intermediate value $Z_3 = Z_{1.i}$ has to be introduced.

Thus , the following program results :

```
P9.64      | – ⇒ z
A9.10   V  |     1
        S  |     o

        | W1 (n) [ R9.100 ( V   , V ) ⇒ z | z ⇒ z | ( z, z, z, z ) ⇒ z
     V  |        | o       o     1     o  | 1   3 |   o o o o      2
     K  |        |             i     i    |       |   o 1 2 3
     S  |        |             1.4   1.4  1.5| o   o|   o o o o      1.4
        |        | (z  ⩾ oLoLo )    ( z =  oLooL ∧ z ) ⇒ z
     V  |        | o                  o          1     1
     K  |        |
     S  |        | 1.5                1.5              1.5
        |        | R9.101 ( z   , ( z, z, z, – )) ⇒ R
     V  |        |          2       3 1 1          o
     K  |        |                                 i
     S  |        |          1.4     o o o          1.4 ]

        | W3 ( 1,4 ) [ – ⇒ R ] | z ⇒ R | z ⇒ R
     V  |            |   o   | | o   o | 1   1
     K  |            | n.i   | |   n.o |
     S  |            |   o ] | o   o | o   o
```

VI. The Semi–logarithmic Representation

(as used with computer V_4)

1) Structure of the Number

In the computer V_4 the numbers are represented in the form

$$y = 2^a \times b$$

wherein a is an integer and b is satisfying the condition

$$L,o \leqslant b < Lo,o$$

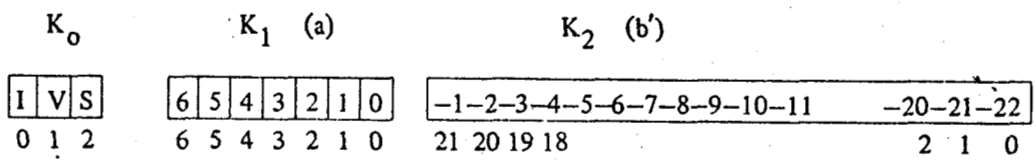
This expression is supplemented by the sign as well as by the symbol for " imaginary " and some special symbols.

The value a can be positive or negative and of the structure A10.2.0 with n = 7. a is an integer binary number of 7 digits, the negative values being represented as supplements.

The value b is represented by 1 + b′ . b′ represents the digits of b after the point. The digit before the point is always L . b′ has 22 binary digits.

The sign, the " Im" –symbol and the special symbols are each Yes–No–Values. They are combined to form a group of 3 Yes–No–Values.

Thus we obtain the following structure of the number :



$S\Delta 1 = (S1.3, S1.7, S1.22)$

In order to avoid negative indices, the digits of b' are numbered sucessively, beginning with the digit with the lowest value. This binary number with the structure S1.21 is called b''.

$$b' = b'' \cdot 2^{-22}$$

The notation $A\Delta 1$ is represented by the following expression :

$A\Delta 1 = (S1.3 , A10.2.0 , A9.2)$.

The meaning of the components 1 and 2 according to page 171, is only true for normal values . This is indicated by the expression

Ko.2

In the case of a special value, the component 1 has another meaning while the component 2 is insignificant. The following values are represented as special values :

1) The value y is " exactly zero " (Kl.2)
2) The value y is " very small " (Kl.5)
$|y| \geq 2^{64}$
3) The value y is very great (Kl.4)
$|y| \geq 2^{64}$

} The limits are only approximate values

In this , the sign may be known or unknown (Kl.3) .

4) The value y is indeterminate. It may also be complex (Kl.1) .

In the following section some programs for arithmetic operations with such complicated structures are developed. As they are only of relevant in case of special interest, this selection is omitted in the English version .

Chapter 4

Operations with Algebraic Expressions

(In particular Calculus of Propositions)

Contents Page

Teil 4 u. 5

2. Korrektur

I. Introduction

1) Problem Definition

The processing of any algebraic expression , e.g. arithmetic or propositional , is to be automated.

In detail, the problems are of the following types :

a) Investigation if given expressions are formed in compliance with the rules.

b) Simplification of Expressions

c) Investigation of Predicates of Expressions

d) Ordering of Elements of Expressions

e) Transformation of Expressions

f) Substitution of Variables by Expressions

g) Development of Programs

h) Investigation of Expressions for Identity

i) Derivation of Explicit Expressions form Implicit Expressions

j) Development of the Derivative of Arithmetic Expressions

k) Integration of Arithmetic Expressions

l) Transformation of Different Forms of Representation into each other

This list can be enlarged , arbitrarily, up to an automation of complete algebras .

2) Representation of Expressions

Algebraic expressions are most advantageously represented by a series of symbols (strings). The representation may follow already introduced forms. But new forms can also be introduced, which are specially suited for computer processing.

The commonly used forms of representation must be varied to some extent, since they are not mere series of symbols (E.g. negation dash in the representation of Hilbert 's calculus of propositions, faction stroke, and representation of powers in arithmetic expressions).

For the various symbols a uniform structure σ is advantageous, e.g. S1.n.

An expression has then the form of a list $S3.m = m \times \sigma$.
This principle allows different forms of representation.

3) Representation of Programs and Functions

The entire field is very extensive, therefore , a systematical numbering of all programs would be too complex.

Within various sections many analog programs occur, such as those for investigations of expressions, simplifications, predicates etc. For the notation of such programs series of letters are introduced first which, if necessary , can be supplemented by indices , e.g. Sal , Sa2 etc. for meaningful expressions (syntactic correct) .

II. The Calculus of Propositions

1) Definition of the Form of Representation

A form based on the formalism of Hilbert is chosen here, but with the following variations:

a) Instead of the " & " , the symbol " Λ " is introduced.

b) Operation symbols are never omitted.

c) Negation is represented by a negation dash " – " before the expression which is to be negated. If this expression is composite it must be put in brackets.

d) Only one type of brackets is allowed.

This form of representation results in mere series of symbols, e.g. :

$$a \wedge b$$
$$\neg a \vee (b \wedge c)$$
$$(\neg (a \wedge b) \vee (c \wedge d) \vee e) \sim g.$$

The following types of symbols are avaiable :

a) Variable Symbol

b) Negation Symbol (monadic)

c) Operation Symbol (dyadic)

d) Bracket Symbol

e) Blank Space Symbol

The number of the variable symbols is in principle unlimited, but a practical limit is set by the structure σ of the symbols.

The following operation symbols are used :

$\vee$	Disjunction
$\wedge$	Conjunction
$\rightarrow$	Implication
γ	Disvalence
$\sim$	Equivalence

The brackets are round ( ) .

As an example for the representation of the symbols we take the following notation:

$$\sigma = S1.8$$

01234567

+	Variable	V_i	Va (x)
oooo–––––	Blank space		Zr (x)
oooo––+–	Negation	–	Neg (x)
oooo+–+–	Brackets	(	Kal (x)
oooo++––		)	Klz (x)
+––o+–––	Operationssymbols	$\wedge$	Op (x)
–+–o+–––		$\vee$	
++–o+–––		$\rightarrow$	
––+o+–––		$\not\rightarrow$	
+–+o+–––		$\sim$	

" o " means, that the component of σ in question is different. In the case of variable symbols the positions o to 6 of σ are used to identify the variable (index of V).

Predicates are introduced for the different types of symbols, for instance :

	$V \Rightarrow Va(V)$		$V \wedge \overline{V} \wedge \overline{V} \wedge \overline{V} \Rightarrow Op(V)$				
V	o	o	o	o	o	o	o
K	7		4	5	6	7	

2) Rules for the formation of series of Symbols and Testing these for Compliance with the Rules

The following rules are established :

a) A single variable represents a meaningful expression

b) Insertion of a negation symbol in front of a meaningful expression results in another meaningful expression.

c) Insertion of an operation symbol between two meaningful expressions results in another meaningful expression.

d) Bracketing of a meaningful expression results in another meaningful expression.

Expressions developed according to these rules are in any case propositions. But they do not always correspond to the operational linkages of their development.

The expression $-a \vee b$ for instance can be developed form the expression $a \vee b$, but the former is not the negation of the proposition $a \vee b$. In common representation it does not mean $\overline{a \vee b}$ but $\overline{a} \vee b$.
Likewise the expression $a_1 \wedge b_1$ and $a_2 \wedge b_2$ can be combined :

$$a_1 \wedge b_1 \quad \vee \quad a_2 \vee b_2$$

According to the rule, that the symbol $\vee$ binds more strongly than the symbol $\wedge$, this expression does not correspond to the expression :

$$(a_1 \wedge b_1) \vee (a_2 \wedge b_2)$$

but to the expression

$$a_1 \wedge (b_1 \vee a_2) \wedge b_2$$

But this result is not relevant with regard to the criterion " meaningful expression " . Expressions formed according to the rules given above are in any case meaningful.

Since any meaningful expression can be produced with the aid of brackets, redundant brackets may occur.

The exact formulation of the rules is the following :

$Sa(x)$ means " x is a meaningful expression "

$Va'(x)$ means " the expression x consists of a single variable " .

$LZ(y_o , y_1)$ means concatenation of the series of symbols y_o and y_1

$$(x)\left[Sa(x) \sim \left[\begin{array}{l} Va'(x) \\ \vee \left[x = Lz(y_o, y_1) \wedge Neg(y_o) \wedge Sa(y_1) \right] \\ \vee \left[x = Lz(y_o, y_1, y_2) \wedge Sa(y_o) \wedge Op(y_1) \wedge Sa(y_2) \right] \\ \vee \left[x = Lz(y_o, y_1, y_2) \wedge Kla(y_o) \wedge Sa(y_1) \wedge Klz(y_2) \right] \end{array} \right] \right]$$

In order to transform this implicit expression for $Sa(x)$ into an explicit one the following procedure is applied :

Expressions of the form $Sa(x)$ are developed step by step from single variables according to the rules b, c, d (see page 145) . In any step , symbols or series of symbols are concatenated. Therefore the criterion has first to be established that two symbols may follows each other in a meaningful expression .

According to the rules a negation symbol must be situated in front of a meaningful expression.

An operation symbol must be situated between two meaningful expressions

The left bracket must be situated in front of the right bracket behind a meaningful expression .

The following predicates are defined :

$Az(x)$ " The symbol x may be situated in front of a meaningful expression "

$Sz(x)$ " The symbol x may be situated behind a meaningful expression "

The exact formulas read:

```
    | (Ex) (Sa ( x ) ∧ x = V) ⇒ Az ( V ) .
V   |                    o         o
K   |                 o
S   | □Xσ             σ  σ      o   σ
```

```
    | (Ex) (Sa ( x ) ∧ x =  V) ⇒ Sz ( V )
V   |                      o         o
K   |                m−1
S   | mXσ            σ   σ      o   σ
```

(Meaningful expressions exist, the first and the last element of which are equal to $\overset{V}{o}$).

From the formulas on page 176 the following recursive definition for Az results:

$Va(x) \vee Neg(x) \vee Kla(x) \vee Az(x) \sim Az(x)$

By an transformation, not shown in detail here, results:

$Va(x) \vee Neg(x) \vee Kla(x) \sim Az(x)$.

Similary, an expression for Sz can be developed:

$Va(x) \vee Klz(x) \sim Sz(x)$.

The predicate 'Az (x) is defined as follows:

" In an meaningful expression the symbol x may stand in front of a symbol of the property Az ".

From the results:

$Neg(x) \vee Kla(x) \vee Op(x) \sim {}'Az(x)$.

In the same way Sz' (x) is defined :

"The symbol x is a meaningful expression may follow a symbol of the property Sz ".

$Klz(x) \vee Op(x) \sim Sz'(x)$.

Supposing that each symbol must have just one of the properties

$Va(x), Zr(x), Neg(x), Kla(x), Klz(x), Op(x)$

then must be true:

$\overline{Zr(x)} \rightarrow ('Az(x) \sim Sz(x)) \wedge (Sz'(x) \sim \overline{Az(x)})$.

Now the auxiliary function is developed :

```
    | Sp0 ( V, V, V )
V   |      o 1 2
S   |      σ σ □Xσ
```

"In a meaningful expression the symbol x may follow [illegible]

" In the list V_2 the element V_1 immediately follows the element V_o " .

$$(Ex) \left[\begin{matrix} V_2 = V_o \wedge V_{2} & = V_1 \\ x & x+1 \end{matrix} \right] \Rightarrow Sq0 \; (V_o, V_1, V_2)$$

Now Sq1 can be defined implicitely :

" Meaningful series of symbols are possible in which the symbol V_1 follows the symbol V_o " .

$$(Ex) \left[(Sa(x) \wedge Sq0 \; (V_o, V_1, x)) \Rightarrow Sq1 \; (V_o, V_1) \right.$$

The following expression can be established :

$$(\text{`}Az \; (V_o) \wedge Az \; (V_1)) \vee (Sz(V_o) \wedge Sz' \; (V_1)) \rightarrow Sq1 \; (V_o, V_1)$$

Its derivation has already been performed. This has to be additionally investigated only if other cases of succeeding symbols are possible which this formula does not identify. Recursive conclusion proves that the formula rules all cases. It representes all cases in which symbols for the limits become succeeding symbols by concatenation .

Single symbols and series of symbols of the property Sa (x) exist as elements of this concatenation. These can be developed step by step form single symbols, without generation of new adjacent symbol combinations.

So we may write :

$$(\text{`}Az \; (V_o) \wedge Az \; (V_1)) \vee (Sz(V_o) \wedge Sz' \; (V_1)) \Rightarrow Sq1 \; (V_o, V_1)$$

This can be transformed as follows :

$$\overline{Zr(V_o)} \wedge \overline{Zr(V_1)} \wedge (Sz(V_o) \not\rightarrow Az (V_1)) \Rightarrow Sq1 (V_o, V_1)$$

This is only true for isolated expressions which do not include symbols for " blank space " .

Now one necessary condition for Sa (V_o) has been formed :

$$Sa(V_o) \rightarrow \left[(x) (y) (Sq0 \; (x, y, V_o) \rightarrow Sq1 \; (x, y)) \right]$$

But this condition is not sufficient in the following cases :

$$\vee a$$

$$(a \wedge b \wedge$$

It becomes clear, that the first symbol must be of the property Az and the last one of the

property Sz :

```
  | Sa( V    ) →  ⌈ Az    ( V ) ∧  Sz( V    )⌉
V |     o          |        o          o      |
K |                |        o          m–1    |
S |     mXσ        ⌊        σ          σ      ⌋
```

The two formulas still do not cover all cases ; e.g. not the following ones :

(a ∧ b ; a ∧ (b ∨ c))

These expressions contain too many, respectively too few, brackets. According to the rules the numberof the (symbols must be equal to the number of the) symbols :

```
Sa( V ) → ⌈ N (x̂ ( x ∈ V ∧ Kla ( x ))) = N ( x̂ ( x ∈ V ∧ Klz ( x ))) ⌉
    o     ⌊               o                             o             ⌋
```

Even the three formulas are still not sufficient , since the following series of symbols satisfies them

a) ∨ (b

Additionally, the following conditions are required :

If for " growing list " any list is understood which can be extended to the list $\underset{o}{V}$ by only adding further symbols, then the covering condition reads : " For any growing list the number of (–symbols must be greater or equal to the number of) – symbols " . This condition is not fulfilled by the growing list a) which belongs to the series of symbols a) ∨ (b .

It can be defined :

```
  | ( Ex  ) ( Lz( V     , x     ) = V  )    ⇒ A1  ( V V )
V |             o                   1              o 1
S |   □Xσ         □Xσ     □Xσ       □Xσ        o
```

$A1\ (\underset{o}{V}, \underset{1}{V}) \equiv$ " V is a growing list of $\underset{1}{V}$ " .

The conditions mentioned above then look as follows:

```
  | Sa ( V  ) →
V |      o
S |      □Xσ

  |  ( x ) ⌈ Aℓ ( x,   V  ) →  N ( ŷ ( y ∈ x ∧ Kla ( y ))) ≥  N ( ŷ ( y ∈ x ∧ Klz ( y )))
V |        |           o
S |   □Xσ  ⌊ o  □Xσ   □Xσ        σ   σ  □Xσ      σ             σ   σ  □Xσ      σ
```

Now we will set up the complete program for $Sa(\underset{o}{V})$:

$$
\begin{array}{l|l}
V & \Big[(x)(y)\Big[Sq0(x,y,\underset{o}{V}) \rightarrow Sq1(x,y)\Big]\Big] Sa(\underset{o}{V}) \wedge Sz(\underset{m-1}{V}) \\
K & \\
 & \wedge \Big[N(\hat{\hat{x}}(x \in \underset{o}{V} \wedge Kla(x))) = N(\hat{\hat{x}}(x \in \underset{o}{V} \wedge Klz(x)))\Big] \\
 & \wedge (x) Al(x, \underset{o}{V}) \rightarrow N(\hat{y}(y \in x \wedge Kla(y))) \geqslant N(\hat{y}(y \in x \wedge Klz(y)))
\end{array}
$$

$$
\begin{array}{l|l}
 & \Rightarrow Sa(\underset{o}{V}) \\
V & \\
S & m \times \sigma
\end{array}
$$

This formula was derived hypothetically. Its proof must be performed by recursive conclusions.

Since a composite expression is developed step by step by combining symbols and sub – expressions, the following must be true :

"If $Sa(x)$ is true for the generation of used partial expressions then it is also true for composite expressions." The exact proof is not reproduced here.

The explicit formula for $Sa(\underset{o}{V})$ can be transformed to achieve easy computability. To each growing list of $\underset{o}{V}$ a value ϵ is assigned, which is equal to the difference of (– ~~symbol and~~) – symbol.

the (-symbols and the)-symbols

$$
\begin{array}{l|l|l}
 & R(\underset{o}{V}) \Rightarrow \underset{o}{R} & R = Sa0(\underset{o}{V}) \\
V & & \\
S & \square \times \sigma \quad o & o \quad o
\end{array}
$$

$$
\begin{array}{l|l|l|l}
 & \textcircled{1} & \textcircled{2} & \textcircled{3} \\
 & Az(\underset{o}{V}) \Rightarrow \wedge R & - \Rightarrow Z & 0 \Rightarrow \epsilon \\
V & o \quad o & o & \\
K & o & & \\
S & \sigma \quad o & \sigma \quad \sigma & 1.n
\end{array}
$$

$$
\begin{array}{l|l|l}
 & W\Big[\mu x(\textcircled{4}\ x \in \underset{o}{V}) \Rightarrow z & \textcircled{5}\ Sq1(z,z) \Rightarrow \wedge R \\
V & o \quad 1 & o\ 1 \quad o \\
K & & \\
S & \sigma \quad \square \times \sigma \quad \sigma & \\
 & \textcircled{6}\ Kla(z) :\rightarrow (\epsilon+1 \Rightarrow \epsilon) & \textcircled{7}\ Klz(z) :\rightarrow (\epsilon-1 \Rightarrow \epsilon) \\
V & 1 & 1 \\
S & \sigma & \sigma \\
 & \textcircled{8}\ \epsilon \geqslant 0 \Rightarrow \wedge R & \textcircled{9}\ z \Rightarrow z \\
V & o & 1 \quad o \\
S & 1.n \quad o & \sigma \quad \sigma \\
 & \textcircled{10}\ Sz(z) \Rightarrow \wedge R & \textcircled{11}\ \epsilon = 0 \Rightarrow \wedge R \\
V & o \quad o & o \\
S & \sigma \quad o & o
\end{array}
$$

Explanation :

① The first symbol must be a start symbol

② Z_o is the last symbol of the just investigated growing list.
Z_1 is the following symbol.
The first Z_o is supposed to be a negation symbol, in order to comply with Sq (Z, Z)
o 1

③ The " balance of brackets " ϵ is zero at the start,

④ The next element of the list V_o results in the new Z_1.
If this does not exist then go to ⑩ .

⑤ For Z_o and Z_1 the " sequence–condition " Sq1 must be true .

⑥ If Z_1 is a (–symbol then ϵ is increased by one .

⑦ If Z_1 is a) –symbol then ϵ is decreased by one.

⑧ In any step ϵ must be greater then or equal to zero .

⑨ Z_1 is substituted for the new Z_o . Go back to ④

⑩ The last Z_o must be an end–symbol.

⑪ ϵ has to be zero at the end of the computation .

It was already been mentioned that this program also results in the predicate " meaningful expression " if there are redundant brackets.

The following cases can be distinguished :

a) Single variable in brackets :

$$(a) \wedge b$$

b) The entire expression in brackets :

$$(a \vee b)$$

c) Redundant brackets :

$$((a \wedge b)) \vee c$$

d) Brackets which are redundant according to the associative rule :

$$(a \vee b) \vee b$$

e) Brackets which are redundant according to the rule of stronger binding .

$$(a \vee b) \wedge c$$

The cases a, b, c are especially simple and will therefore be discussed first.

In order to exclude these cases, new rules for the development of meaningful expressions have to be established. For this purpose some new predicates are defined : :

Aa (x) = " x is a " linkage expression " (it can be linked to operation symbols)

Ba (x) = " The series of symbols x can be put in brackets " .

Ca (x) = " The series of symbols x is put in brackets once " .

The rules for the development of meaningful expressions then assume the following form :

a) A single variable is a linkage expression .

b) By inserting a negation symbol in front of a linkage expression an other expression , which can be put in brackets , is produced .

c) By inserting an operation symbol between two linkage expressions an other expression, which can be put in brackets , is produced.

d) By bracketing an expression for which this is allowed a " bracketed expression " is produced.

e) An expression which can be put in brackets or a bracketed expression is also a linkage expression.

f) A linkage expression is also a meaningful expression. A bracketed expression is not a meaningful expression.

The formulas for these rules are the following :

a) $Va' \; (\underset{o}{V}) \rightarrow Aa \; (\underset{o}{V})$

b) $\left[\underset{o}{V} = Lz(x,y) \wedge Neg \; (x) \wedge Aa \; (y) \right] \rightarrow Ba(\underset{o}{V})$

c) $\left[\underset{o}{V} = Lz(x,y,z) \wedge Aa \; (x) \wedge Op \; (y) \quad Aa (z) \right] \rightarrow Ba (\underset{o}{V})$

d) $\left[\underset{o}{V} = Lz(x,y,z) \wedge Kla (x) \wedge Ba (y) \wedge Klz (z) \right] \rightarrow Ca (\underset{o}{V})$

e) $Ba(\underset{o}{V}) \vee Ca(\underset{o}{V}) \rightarrow Aa (\underset{o}{V})$

f) $Aa \; (\underset{o}{V}) \wedge \overline{Ca(\underset{o}{V})} \rightarrow Sal \; (\underset{o}{V})$

In order to specify that these are the only formulas for Aa, Ba, Ca, Sal they have to be formulated as follows :

$$\vdash \quad \left[Va'_{o}(V) \vee Ba_{o}(V) \vee Ca_{o}(V) \sim Aa_{o}(V) \right]$$

$$\wedge \left[V_{o} = Lz(x,y) \wedge Neg(x) \wedge Aa(y)) \vee (V_{o} = Lz(x,y,z) \wedge Aa(x) \wedge Op(y) \wedge Aa(z)) \sim Ba_{o}(V) \right]$$

$$\wedge \left[V_{o} = Lz(x,y,z) \wedge Kla(x) \wedge Ba(y) \wedge Klz(z)) \sim Ca_{o}(V) \right]$$

$$\wedge \left[Aa_{o}(V) \wedge \overline{Ca}_{o}(V) \sim Sa1_{o}(V) \right]$$

To transform this implicit formula Sal (V) into an explicit formula, the condition for Sq1 (x,y) is established first. It can shown that the formula derived before for Sq1 is also true in this case . Generally , the following is true :

$$Sal(x) \rightarrow Sa0(x)$$

which means that if an expression is meaningful according to stronger rule Sal, then it is also correct according to Sa0.

To obtain Sal the formula for Sa0 must be supplemented by additional rules for brackets. Only expressions of the form Ba (x) may be bracketed. But these again can only be developed by the linking of expressions by means of operation symbols or negation symbols. Therefore , at least one operation symbol or one negation symbol must be situated between two coordinated brackets.
Examples :

$$(a \wedge b)$$

$$(a \vee b \wedge c)$$

$$(-a)$$

$$(-a \vee b)$$

But this symbol must not be bracketed by additional subbrackets such as, e.g. :

$$((-a))$$

This is a case of duplicate brackets. The formulation of this condition is possible with the aid of the balance of brackets ϵ, which specifies how many brackets have to be eliminated for each Va – or Neg – or Op– symbol .

Example :

	(a ∧ b) ∨ (a ∧ (–b ∨ c))
ϵ	1 1 1 0 1 1 22 2 2

Within each pair of brackets there must at least be one operation present or one negation symbol of the corresponding level of ϵ . This is expressed by the auxiliary values Z^{2}_{ϵ}

The following program for Sal results :

In order to exclude bracketing of the entire expression, $Z_{2\,o}$ has to be positive in the case where V_o is not a single variable .

```
   | R ( V  ) = R | R = Sal  ( V )
V  |     o     o |     o        o
S  |   □Xσ       |
```

```
   | Az  ( V ) ⇒ ∧ R | ⊖ ⇒ z | 0 ⇒ ϵ | − ⇒ z | ⊖ ⇒ z
V  |       o       o |     o |       |     o |     o
K  |       o         |       |       |     o |
```

```
   | W [ μ x ( x ∈ V  ) ⇒ z | Sql ( z, z ) ⇒ ∧ R ]
V  |           o       1  |        o 1       o
K  |                        |
   |
   |     Kla ( z ) → [ ϵ + 1 ⇒ ϵ | − ⇒ z ]
V  |          1     [           |     2 ]
K  |                [           |     ϵ ]
   |
   |     Op ( z ) ∨ Neg (  z ) → [ + ⇒ z ]
V  |          1          1     [     2 ]
K  |                             [     ϵ ]
   |
   |     Klz ( z ) → [ z ⇒ R | ϵ − 1 ⇒ ϵ ]
V  |          1     [ 2   o |           ]
K  |                [ ϵ     |           ]
   |
   |     ϵ ⩾ 0 ⇒ ∧ R | z ⇒ z ]
V  |               o | 1   o ]
K  |                 |       ]
```

```
   | Sz ( z ) ⇒ ∧R | ϵ = 0 ⇒ ∧ R | ( V̄ā (V) → z ) ⇒ ∧ R
V  |      o     o |             o |      o     2       o
K  |                |               |                 o
```

Now the case d) and e) (see page 182) are dealt with. It is also idented to exclude brackets which can be omitted according to the binding rank of the operation symbols.

The ranks will be ordered in such a way that the stronger binding symbol has the lower rank. The symbol with the higher rank, therefore reaches farther. Thus the propositional operations are then ordered as follows :

```
− ∨ ∧  → ↛ ~
0 1 2  3 4 5
```

If the rank of a symbol is designated Rg (x) , then it is possible to assign a rank Rg (y)

to each expression. In this single Va – symbols are assigned the rank zero.

Since composite expressions can only be developed with the aid of operation symbols, each composite expression has an operation symbol, which corresponds to the last composition. The rank of this symbol is equal to the rank of the composite expression.

Now new rules for meaningful expression can be established :

a) A single variable symbol is a meaningful expression of the rank zero.

b) By insertion of a negation symbol in front of a meaningful expression, the rank of which is either zero or is not zero but bracketed, another meaningful expression is produced.

c) By insertion of an operation symbol. for which the associative rule applies ($\vee$, $\wedge$) between two meaningful expressions, another meaningful expression is produced, if the expressions to be connected satisfy the following conditions : either their rank is lower than or equal to that of the operation symbol, or it is higher and the expressions to be connected are bracketed singly.

d) By insertion of an operation symbol, for which the associative rule does not apply ($\rightarrow$, $+$, $\sim$), between two meaningful expressions, another meaningful expression is produced , if the expressions to be connected satisfy the following conditions : either their rank is lower than that of the operation symbol or it is equal to or higher than this and the expressions to be connected are bracketed singly.

e) An expression between single brackets is developed by bracketing a meaningful expression not previously bracketed.

f) The rank of composite expressions according to b), c) and d) is equal to that of the operation symbol used in the last linkage procedure.

g) The rank of an expression is not changed by its bracketing.

The exact formulation of the rules a) to g) reads:

a) $Va'(V) \rightarrow Sa2(V) \wedge Rg(V) = 0$ (o, o, o)

b)f) $\left[V = Lz(x,y) \wedge Neg(x) \wedge Sa2(y) \wedge ((Rg(y) = 0) \not\rightarrow Kl'(y)) \right]$ (o)

$\rightarrow Sa2(y) \wedge Rg(y) = 0$

c)f) $\left[V = Lz(x,y,z) \wedge Sa2(x) \wedge Opa(y) \wedge Sa2(z) \right.$ (o)

$\left. \wedge (Rg(x) \leqslant Rg(y) \not\rightarrow Kl'(x)) \wedge (Rg(z) \leqslant Rg(y) \not\rightarrow Kl'(z)) \right]$

$\rightarrow Sa2(V) \wedge Rg(V) = Rg(y)$ (o, o)

d)f) $\left[V = Lz(x,y,z) \wedge Sa2(x) \wedge Op(y) \wedge \overline{Opa}(y) \wedge Sa2(z) \right.$ (o)

$\left. \wedge (Rg(x) < Rg(y) \not\rightarrow Kl'(x)) \wedge (Rg(z) < Rg(y) \not\rightarrow Kl'(z)) \right]$

$\rightarrow Sa2(V) \wedge Rg(V) = Rg(y)$ (o, o)

e)g) $\left[V = Lz(x,y,z) \wedge Kla(x) \wedge Sa2(y) \wedge \overline{Kl'}(y) \wedge Klz(z) \right]$ (o)

$\rightarrow Kl'(V) \wedge Rg(V) = Rg(y)$ (o, o)

Meaning of the new predicates:

Kl'(x) "The expression x is a singly bracketed meaningful expression".

Opa(x) "The symbol x is an operation symbol for which the associative rule applies".

The corresponding program for Sa2 can be derived as follows:

$Sa2(V) \rightarrow Sa1(V)$ (o, o)

The conditions of Sa2 remain the same, but the conditions for the justification of brackets are made more stringent.
First, for each bracketed expression the rank assigned to it must be investigated; further, the rank of that operation symbol which is linkage with the pairs of brackets. For according to the rules, each bracketed expression must be fitted with an "inside" and an "outside" operation symbol. If x is the inside and y the outside operation symbol, then the bracketing is justified, if the following condition is satisfied:

$(Rg(x) > Rg(y)) \vee (x = y \wedge Opa(x))$

The inside operation symbol is the one between the brackets, but not between sub–brackets, which has the higher rank. The outside operation symbol is that one of the two adjacent operation symbols (if they exist) which has the lower rank.

Example :

$$a \wedge (b \rightarrow (c \sim d) \wedge e) \vee g$$

Outer brackets

inside op.–symbol :	$\rightarrow$	Rank 3
outside op.–symbol :	$\vee$	Rank 1

Inner brackets

inside op.–symbol	$\sim$	Rank 5
outside op.–symbol	$\wedge$	Rank 2

The rank of the two symbols must be investigated for each bracketed expression. To perform this investigation currently, i.e. by a step by step inspection of succeeding symbols of an expression the relevant values must be stored up to the bracket level ϵ . To each bracket level 0 to ϵ the following values are assigned :

$Z_{3\epsilon}$ = highest rank of all so far inspected operation symbols of the just investigated bracketed expression of the level ϵ . (inside operation symbol) .

$Z_{4\epsilon}$ = lowest rank of all so far inspected operation symbols of the just investigated bracketed expression of the level ϵ . (outside operation symbol of the level $\epsilon + 1$) .

$Z_{5\epsilon}$ = " The operation symbol belonging to $Z_{4\epsilon}$ has the property Opa. "

Z_{6} = indicates that the preceeding symbol was a Klz–symbol.

The computation is performed in the following way : as soon as the preceding symbol was a Klz–symbol (Z_6) , i.e. after the investigation of a bracketed expression was completed and the succeeding symbol became known, the justification of the bracketing is tested

After that the values $Z_{3\epsilon}, Z_{4\epsilon}, Z_{5\epsilon}$ are erased.

At the start of each bracketing investigation the input values must be set as follows :

$Z = 0$	$Z = 5$	$Z = -$
3	4	5
ϵ	ϵ	ϵ

Z_4^{ϵ} must be equal to the highest possible rank of the operation symbols, so that in the first computation of the formula

$$\text{Min}\left(Z_4^{\epsilon}, \text{Rg}\left(Z_1\right)\right) \Rightarrow Z_4^{\epsilon}$$

$\text{Rg}\left(Z_1\right)$ results form Z_4^{ϵ}

First we state :

	R (V) ⇒ R	R = Sa2′ (V)
V	o o	o o
S	□Xσ	

Erroneous
see page 191

	Az (V) ⇒ ∧ R	0 ⇒ z	0 ⇒ ϵ	– ⇒ z	0 ⇒ z	5 ⇒ z	– ⇒ z	– ⇒ z
V	o o	o		2	3	4	5	6
K	o			o	o	o	o	
S	σ o	σ σ	1	o	1,3	1,3	o	

	W [μ x (x ∈ V) ⇒ z	Sq1 (z, z) ⇒ ∧R
V	o 1	o 1 o
K		
S	σ □Xσ σ	σ σ o

	Op (z) ∨ Neg (z) → [+ ⇒ z	(Rg (z) ⇒ z	Maj (z , Z) ⇒ z
V	1 1	2 1 7	3 7 3
K		ϵ	ϵ
S	σ σ	o σ 1.3	1.3 1.3 1.3

	z < z → [z ⇒ z	Opa (z) ⇒ z
V	7 4	7 4	1 5
K	ϵ	ϵ	ϵ
S	1.3 1.3	1.3 1.3	σ o

	z → [(z > z ∨) ∨ (z = z ∧ $\overline{z}$) ⇒ ∧R]
V	6	3 4	3 4 5 o
K		ϵ+1 ϵ	ϵ+1 ϵ ϵ	
S	o	1.3 1.3	1.3 1.3 o o

	z ⇒ z	Opa (z) ⇒ z]
V	7 4	1 5
K	ϵ	ϵ
S	1.3 1.3	σ o

	Kla (z) → [ϵ+1 ⇒ ϵ	– ⇒ z	0 ⇒ z	5 ⇒ z	– ⇒ z]
V	1	2	3	4	5
K		ϵ	ϵ	ϵ	ϵ
S	1 1	o	1.3	1.3	o

	Klz (z) → [z ⇒ ∧ R	ϵ–1 ⇒ ϵ]
V	1	2 o	
K	ϵ		
S	o o	1 1

	ϵ ⩾ 0 ⇒ ∧ R	Klz (z) ⇒ z	z ⇒ z]
V	o	1 6	1 o
S	1	σ o	σ σ

	Sz (z) ⇒ ∧ R	ϵ = 0 ⇒ ∧R	$\overline{(\text{Va}'\ (\text{V}) \rightarrow \text{z})}$ ⇒ ∧ R
V	o o	o	o 2 o
S	σ o	1 o	□Xσ o o

The program for Sa2 can be demonstrated by an example :

$V_o \equiv -(a \vee b) \rightarrow ((c \wedge d) \vee -b \rightarrow e) \sim f$

	V	ϵ	z	z	z z z	z	z	z	z	z	z	z z z	z	z
V	o		o	1	2 2 2	3	3	3	4	4	4	5 5 5	6	7
K					o 1 2	o	1	2	o	1	2	o 1 2		
		o			—	o			5			—	–	
1	–	o	–		+	o			o			—	–	5
2	(	1	–	(	+–	o	o		o	5		—–	–	
3	a	1	(	a	+ .	o	o		o	5		—–	–	
4	∨	1	a	∨	++	o	1		o	1		–+	–	1
5	b	1	∨	b	++	o	1		o	1		–+	–	
6	)	o	b	)	++	o	1		o	1		–+	+	
7	→		)	→	++	3	1		3	1		–+	–	3
8	(	o 1	→	(	+–	3	o		3	5		—–	–	
9	(	2	(	(	+––	3	o	o	3	5	5	——+	–	
10	c	2	(	c	+––	3	o	o	3	5	5	——+	–	
11	∧	2	c	∧	+–+	3	o	2	3	5	2	—–+	–	2
12	d	2	∧	d	+–+	3	o	2	3	5	2	—–+	–	
13	)	1	d	)	+–+	3	o	2	3	5	2	—–+	+	
14	∨	1	)	∨	+++	3	1	2	3	1	2	–++	–	1
15	–	1	∨	–	+++	3	1	2	3	o	2	–++	–	0
16	b	1	–	b	+++	3	1	2	3	o	2	—–+	–	
17	→	1	b	→	+++	3	3	2	3	o	2	—–+	–	3
18	e	1	→	e	+++	3	3	2	3	o	2	—–+	–	
19	)	o	e	)	+++	3	3	2	3	o	2	—–+	+	
20	~	o	)	~	+++	5	3	2	3	o	2	—–+	–	5
21	f	o	~	f	+++	5	3	2	3	o	2	—–+	–	

The current intermediate values have to be distinguished with regard to the difference between old and new values to the left and to the right of the symbol " ⇒ " . Therefore, the symbols

are sometimes positioned between the lines.

The program of page 184 is not yet complete. For this reason it was designated $Sa2'(V_o)$ according to $Sa'(V_o)$ all brackets which are redundant according to the rules, are eliminated, but no investigation is performed to find out whether all necessary brackets are set.

The program allows the expression :

$$a \rightarrow b \rightarrow c$$

but according to the rule d) page 185 such an expression is not allowed. Permitted are, either the expression :

$$(a \rightarrow b) \rightarrow c$$

or the expression :

$$a \rightarrow (b \rightarrow c)$$

for the associative rule does not apply to the implication. Therefore, if the expression which is to be linked with another expression by the symbol $\rightarrow$ is an implication itself, then this expression has to be bracketed , since both expressions are of equal rank.

The same applies to the symbols $\nrightarrow$ and $\sim$. We get the following demand :

" On the same bracket level only one of the symbols $\rightarrow$, $\nrightarrow$, $\sim$ at most is allowed " .

Thus we need for every level three Yes–no–Values which indicate if these symbols already appeared within the just investigated pair of brackets. The three values are combined to the value Z 8 ϵ

Definition :

Imp (x) = " x is an implication symbol "

Disv (x) = " x is a disvalence symbol "

Aeq (x) = " x is an equivalence symbol "

The program on page 189 must be supplemented by the following :

	Op (z) →	Imp (z) ⇒ z		Disv (z) ⇒ z		Aeq (z) ⇒ z	
V	1	1	10	1	11	1	12
S	σ	σ	o	σ	o	σ	o

	$\overline{z \wedge z} \wedge$		$\overline{z \wedge z} \wedge \overline{z \wedge z} \Rightarrow \wedge$ R			
V	8	10	8	11	8	12	o
K	ϵ.o	o	ϵ.1	o	ϵ.2		
S	o	o	o	o	o	o	o

	z ⇒ z		z ⇒ z		z ⇒ z	
V	10	8	11	8	12	8
K		ϵ.o		ϵ.1		ϵ.2
S	o	o	o	o	o	o

At the start the following has to be set :

	(– – –) ⇒ z	
V		8
K		o
S		1.3

and for each Kla – symbol :

	Kla (z) →	[ϵ+1	⇒ ϵ	(– – –)	⇒ z	]	...]
V					8		
K					ϵ		
S		1	1		1.3		

The program on page 188 contains an error.

If the last symbol is a) – symbol then the coordinated bracketed expression is not investigated. For instance, the following expression is allowed according to the program :

$$a \wedge (b \vee c)$$

Here the brackets are redundant. This fact cannot be tested, however before a symbol suceeding the Kla – symbol is investigated. Since this symbol does not exist, the test cannot be performed. This failure can be avoided in the following way :

a) An i – expression is substituted for the μ – expression

b) If the last symbol is a (–symbol then the investigation of the bracketed expression is performed at once.

	R (V) ⇒ R	R = Sa2 (V)
V	o o	o o
S	mXσ o	

	Az (V) ⇒ Λ R	⊖ ⇒ z	0 ⇒ ϵ	– ⇒ z	0 ⇒ z	5 ⇒ z
V	o o	o		2	3	4
K	o			o	o	o
S	σ o	σ σ	1.3	o	1.3	1.3

	– ⇒ z	– ⇒ z	(– – –) ⇒ z
V	5	6	8
K	o		o
S	o	o	1.3

	W1 (N(V))	V ⇒ z	Sq1 (z, z) ⇒ Λ R
V	o	o 1	o 1 o
K		i	
S		σ σ	σ σ o

	Op (z) ∨ Neg (z) →	+ ⇒ z	Rg (z) ⇒ z	Maj (z , z) ⇒ z
V	1 1	2	1 7	3 7 3
K		ϵ		ϵ ϵ
S		o	σ 1.3	1.3 1.3 1.3

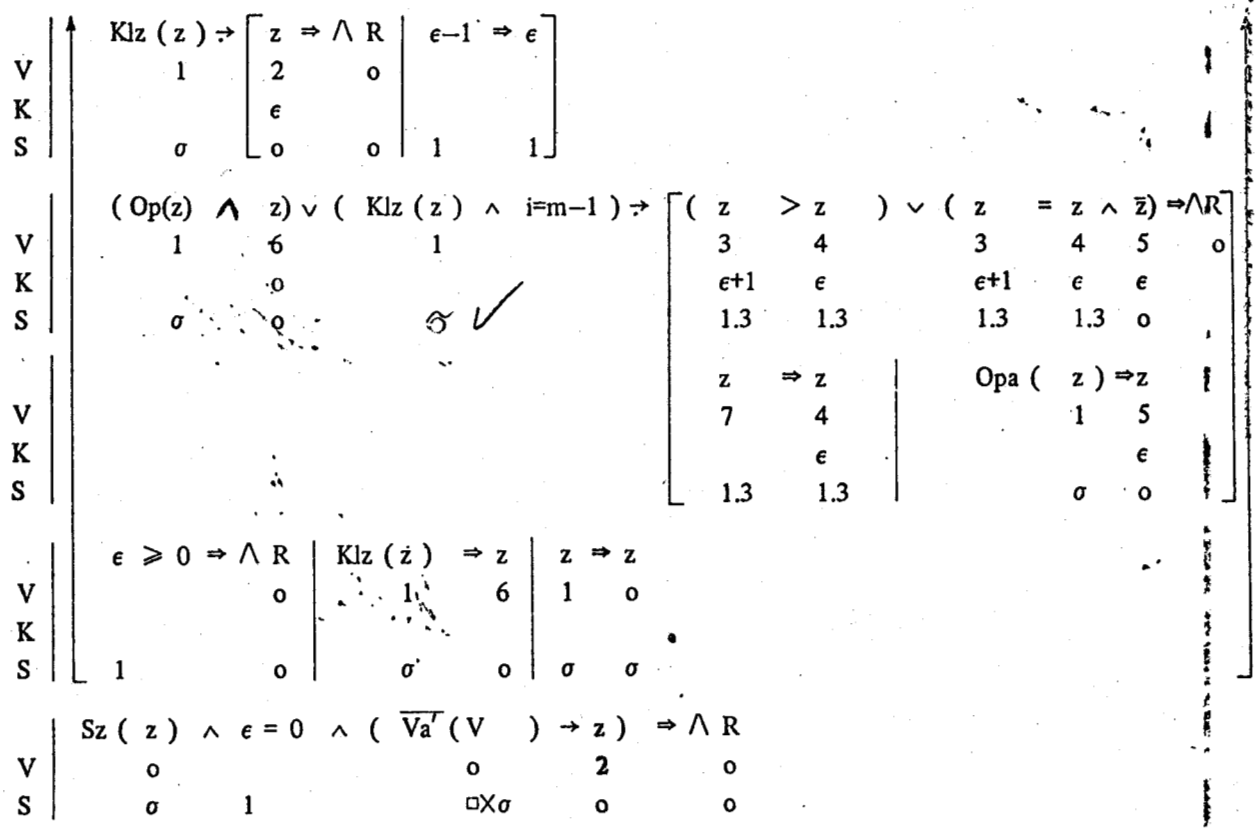
	z < z →	z ⇒ z	Opa (z) ⇒ z
V	7 4	7 4	1 5
K	ϵ	ϵ	ϵ
S	1.3 1.3	1.3 1.3	σ σ

	Op (z) →	Imp (z) ⇒ z	Disv (z) ⇒ z	Aeq (z) ⇒ z
V	1	1 10	1 11	1 12
K				
S	σ	σ o	σ o	σ o

	$\overline{z \wedge z} \vee \overline{z \wedge z} \wedge \overline{z \wedge z}$ ⇒ Λ R
V	8 10 8 11 8 12 o
K	ϵ.o ϵ.1 ϵ.2
S	o o o o o o o

	z ⇒ z	z ⇒ z	z ⇒ z
V	10 8	11 8	12 8
K	ϵ.o	ϵ.1	ϵ.2
S	o o	o o	o o

	Kla(z) →	ϵ+1 ⇒ ϵ	– ⇒ z	0 ⇒ z	5 ⇒ z	– ⇒ z	(– – –) ⇒ z
V	1		2	3	4	5	8
K			ϵ	ϵ	ϵ	ϵ	ϵ
S	σ	1 1	o	1.3	1.3	o	1.3



3) Simplification of Expressions

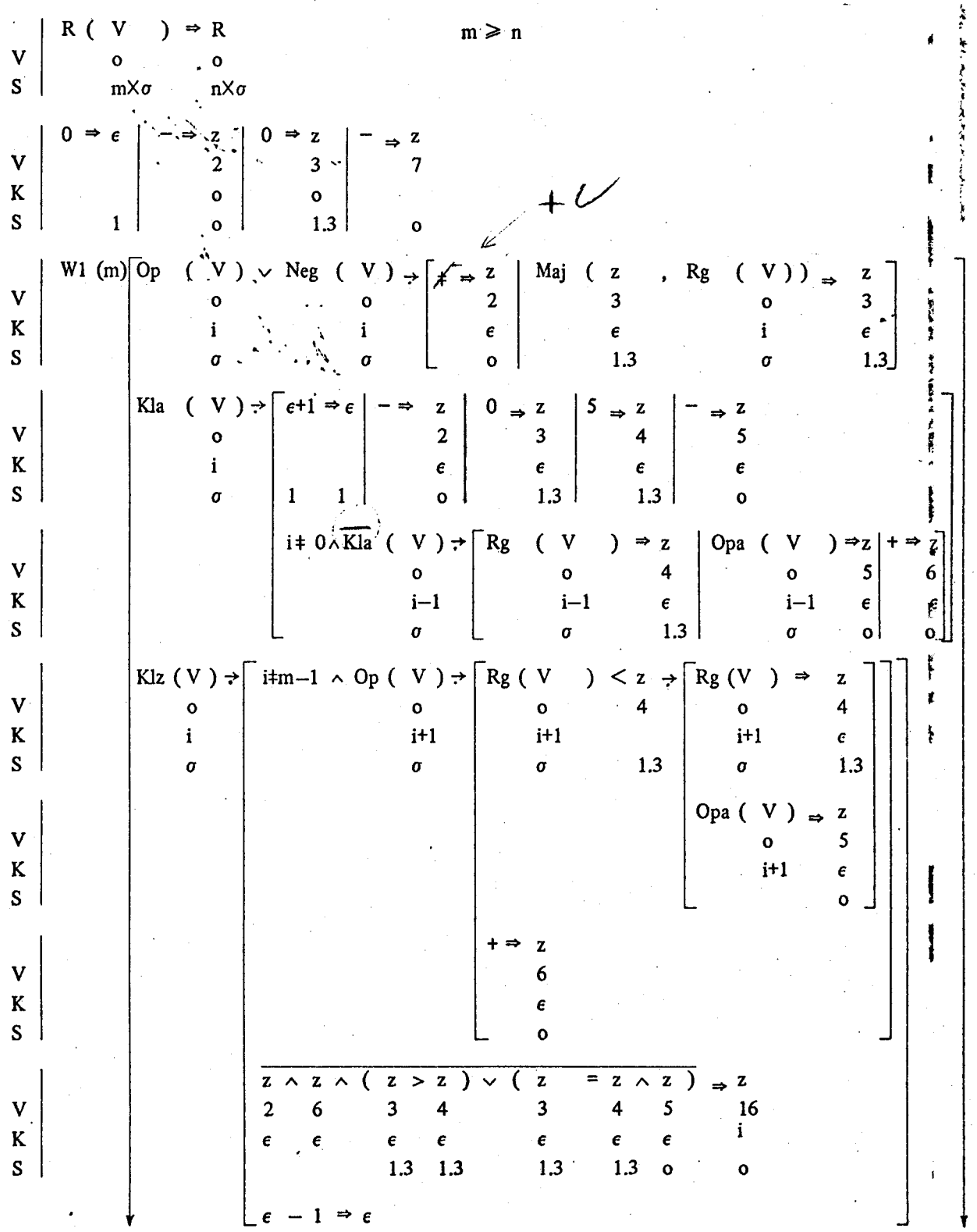
An expression for which the predicate Sa0 is true is to be simplified by elimination of duplicate negations and redundant brackets. This problem cannot be solved in one run since it is only after completion of the investigation of a bracketed expression and of the following symbol that it can be stated whether the brackets are necessary. In the first run a supplementary value Z_{16} is formed, by which a Yes – No – Value is assigned to each element of V_o which indicates whether the relevant symbol is redundant or not. First, the investigation is only possible with redundant negation symbols and Klz symbols. The corresponding Kla – symbols result from a subsequent backward run of V_o.

The program is based on that for Sa2. But here only those expressions are necessary, which serve the investigation as to whether brackets are necessary. A little variation is introduced, relative to Sa2. The outside operation symbol is formed as soon as a Kla – or a Klz – symbol occurs.

The Yes – No – Value Z_7 serves the investigation for duplicate negations. It changes its value whenever a negation symbol occurs. At the start of each new period it has to be negative. If Z_7 is positive then Z_{16} turns positive. If several negation symbols follow each other in a sequence, then every second gets the mark Z_{16}^{i}.

In the backward run of V_o from m – 1 to 0 the bracket level is investigated again.

Z_{17}^{ϵ} indicates, whether the brackets of the level ϵ are redundant. Then the respective (and) symbols can be omitted. Similarily in the case Z_{16}^{i} is positive the respective negation symbol and the preceding one (i −1) can be omitted.



```
        Neg ( V ) ∧ z ⇒ z  |  Neg ( V ) ∧ z̄ ⇒ z
V       o     7   16       |  o     7   7
K       i         i        |  i
S       σ     o   o        |  σ     o   o

W2 (m) [ Klz ( V ) → [ϵ+1 ⇒ ϵ | z ⇒ z ]
V               o    |        | 16  17
K               i    |        | i   ϵ
S               σ    |        | o   o ]

       [ Kla ( V ) ∧ z ⇒ z
V              o     17  16
K              i     ϵ   i
S              σ     o   o

         Neg ( V ) ∧ z ⇒ z
V        o         16  16
K        i         i   i−1
S        σ         o   o ]

W1 (m) [ z → V ⇒ μ R
V        16  o     o
K        i   i
S        o   σ     □Xσ ]
```

Further simplifications of expressions will not be discussed here.

4) Introduction of the Computer Oriented Representation of Propositional Expressions

Before entering a discussion on additional automation of the Plankalkuel it is of advantage to introduce a form of representation which is specially suited for this purpose. Since this form will be processed by computers, it will be called " Computer representation " .

The operation symbols between two neighbouring variables are replaced by function symbols which are placed in front of the variables.

So	is replaced by
a ∧ b	∧ (a,b)
a ∨ b	∨ (a,b)
a → b	→ (a,b)

etc.

For operations for which the associative rule applies, the function symbol can be associated with several variables :

a ∨ b ∨ c ∨ d	∨ (a, b, c, d)

Composite expressions then assume the following form :

a ∧ b ∨ c ~ d	~ (∧ (a, ∨ (b, c)))
(a ∧ b) ∨ (c → d)	∨ (∧ (a, b) , → (c, d))

Negations are represented as follows :

Single variables are negated, as with Hilbert's representation. The negation of composite expressions is indicated by a negation of the respective operation symbol.

$\overline{\bar{a} \vee b} \wedge c$	$\wedge (\overline{\vee} (, \bar{a}, b))$
$\overline{a \vee \bar{b} \sim \overline{a \vee b}}$	$\overline{\sim} (\vee (a, \bar{b}), \overline{\vee} (\bar{a}, b))$

Assuming that the variables are represented by single symbols the comma–symbol can first be omitted. Further, the brackets can be eliminated by the following method :

To each operation symbol a rank is assigned which represents the bracket level of the expression succeeding the operation symbol. In the following expressions the ranks of the operation symbols are demonstrated :

	∨ (a, b)	~ (∧ (a, ∨ (b, c)))
Rg	1	3 2 1

	$\overline{\sim} (\vee (a, \bar{b}), \overline{\vee} (\bar{a}, b))$
Rg	2 1 1

These expressions can now be represented univocally by omission of brackets and commas :

	∨ ab	~ ∧ a ∨ b c	$\overline{\sim} \vee a \bar{b} \overline{\vee} \bar{a} b$
Rg	1	3 2 1	2 1 1

Now the " range " of an operation symbol always reaches to the next operation symbol of the same or a higher rank.

The order of procedure of the operation symbols according to page 184 could be applied; the highest rank of a formula could then be reduced. But the resulting simplification must be paid for by more complete programs.

184

However, this representation allows ambiguities, as the following expressions with identical computer representation show :

a ∨ b ∧ c ∧ d	aeq	∧ ∨ a b c d 2 1
a ∨ b ∨ c ∧ d	aeq	∧ ∨ a b c d 2 1

This can happen only if a conjunction and a disjunction symbol are adjacent to each other. The same ambiguity occurs through an exchange of the symbols ∧ and ∨ in the above expressions ;

(a ∧ b) ∨ c ∨ d	aeq	∨ ∧ a b c d 2 1	
(a∧b ∧ c) ∨ d	aeq	∨ ∧ a b c d 2 1	

Such ambiguity is not possible, however , with operation √ which must always deal exactly with two operands.

a ∨ b ∨ c → d	aeq	→ ∨ a b c d 2 1	
(c ~ d) ∧ a ∧ b	aeq	∧ ~ c d a b 2 1	

In the first case the last of the variables a, b, c, d, must be coordinated to the symbol → , since only then does the symbol combine two operands. The same applies to the second expression in which the variables c and d must be coordinated to the symbol ~ and the variables a and b to the symbol ∧ .

These ambiguities can be eliminated in various ways :

a) Variation of the sequence according to the commutative rule :

a ∨ b ∧ c ∧ d	aeq	c ∧ d ∧ a ∨ b	∧ c d ∨ a b 2 1
a ∨ b ∨ c ∧ d	aeq	d ∧ a ∨ b ∨ c	∧ d ∨ a b c 2 1

This method does not need any additional symbols, but only formal transformation .

b) Assignment of ranks to the variables :

a ∨ b ∧ c ∧ d	aeq	∧ ∨ a b c d 2 1 o o 1 1	
a ∨ b ∨ c ∧ d	aeq	∧ ∧ a b c d 2 1 o o o 1	

c) Auxiliary operation of the identify I (x)

I (x) = x

Now a rank can be assigned to I (x)

a ∨ b ∧ c ∧ d	aeq	∧ ∨ a b I c I d 2 1 1 o	
a ∨ b ∨ c ∧ d	aeq	∧ ∨ a b c I d 2 1 1	

d) Use of separation symbols .

a ∨ b ∧ c ∧ d	aeq ∧ ∨ a b | c d 2 1
a ∨ b ∨ c ∧ d	∧ ∨ a b c | d 2 1

e) Combination of the separating symbols with the variable symbols :

A Yes–No-Value is assigned to the variables which indicates whether the variable is situated at the end of a sub–expression:

a ∨ b ∧ c ∧ d	aeq ∧ ∨ a b c d 2 1 ·
a ∨ b ∨ c ∧ d	aeq ∧ ∨ a b c d 2 1 ·

Of all methods, except a) , the methods d) and e) require the smallest investment.

In the following method , d) separation of symbols is applied. These can then be easily eliminated by method a) .

Once again some formulas are confronted with each other in S–representation and in computer representation :

S–representation	Computer–representation
a	$\bar{a}$
a ∧ b	∧ a b
a → b	→ a b
a ∨ b ∨ c ∨ d	∨ a b c d
a ∧ b ∨ c ~ d	~ ∧ a ∨ b c 3 2 1
(a ∧ b) ∨ (c → d)	∨ ∧ a b → c d 2 1 1
$\overline{a \vee \bar{b}} \wedge c$	∧ $\bar{\vee}$ a $\bar{b}$ 2 1
$\overline{a \vee \bar{b}} \sim \bar{\bar{a}} \vee b$	$\bar{\sim}$ ∨ a $\bar{b}$ $\bar{\vee}$ $\bar{a}$ b 2 1 1
a ∨ b ∧ c ⊼ d	⊼ ∨ a b | c d 2 1
a ∨ b ∨ c ∧ d	∧ ∨ a b c | d 2 1
(a ∧ b ∧ c) ∨ d	∨ ∧ a b c | d 2 1
(a ∨ b ∧ c) ∨ d ∧ e	∧ ∨ ∧ ∨ a b | c | d | e 4 3 2 1

Representation of the symbols :
Each single symbol is now composed of several components :

1) The type of the symbol
Variable or operation symbol, or blank space, or separating symbol

2) The information whether a value has to be considered as positive or as negative.

3a) In case of a variable the index

3b) In case of an operation symbol, the type of the symbol and the rank.

An example for such a code which the structure

σ = S1.8 (see page 145)

is given below :

0	1	2	3	4	5	6	7			
						+	–	Neg	Variable	$K_0 \div K_5$ = Index
						+	+	Pos		
			+	–	–	–	–	Neg	$\vee$	op-symbol; K_0, K_1, K_2 = Rank; K_7 = Negation–information
			+	–	–	–	+	Pos		
			–	+	–	–	–	Neg	$\wedge$	
			–	+	–	–	+	Pos		
			+	+	–	–	–	Neg	$\rightarrow$	
			+	+	–	–	+	Pos		
			–	–	+	–	–	Neg	γ	
			–	–	+	–	+	Pos		
			+	–	+	–	–	Neg	$\sim$	
			+	–	+	–	+	Pos		
o	o	o	–	–	–	–	–		blank space	
o	o	o	+	+	+	–	–		separation symbol	

Remark : Method d) is not sufficient , it is better to use method e)

With the above used code the following predicates can be defined :

Neg (x)	the symbol x is negated
Va (x)	the symbol x is a variable
Op (x)	the symbol x is a operation symbol
Opa (x)	the symbol x is a conjunction – or disjunction –symbol
Zr (x)	the symbol x is a blank space symbol
Tr (x)	the symbol x is a separation symbol
Rg (x)	Rank of x (in the case Op (x))

Chapter 5

Chess–Programs

Contents

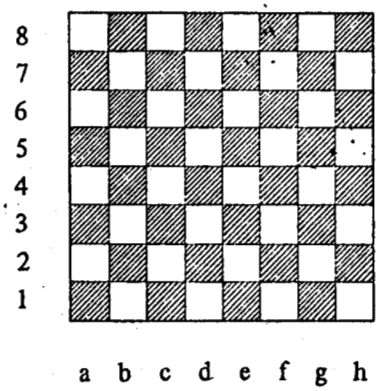
I) Geometry of the Chess–Board

1) Given System

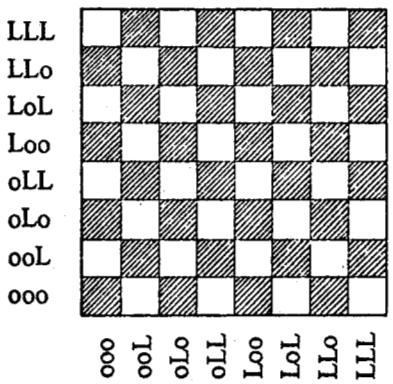
The chess board contains 64 squares. From now on they will be called " points " . The whole area of the board will be called " field " .

The location of any point in the field is determined by two coordinates, each of which is a 2^3 = 8–fold variable. Therefore, the location of any point can be represented by 6 yes-no-values, or by 2 binary numbers with 3 digits each . Then the following notations are equivalent to each other :

Common Notation



Coordinates Notation



In the coordinates notation the horizontal coordinate is written first, the vertical coordinates second. Then the following notations for instance are equivalent:

c2 = Loo, ooL

g6 = LLo, LoL

Two new structure symbols are introduced :

AΔ1 = S1.3. Coordinate of a point
TΔ1

AΔ2 = 2XAΔ1 Location of a point

Several operations with coordniates are required. These correspond to the arithmetic rules for binary numbers.

(V + V , V – V , \| V – V \|)
o 1 o 1 o 1

2) Propositions on the location of a point

Marginal Values for PΔ1 to PΔ3

	R (V) ⇒ R
V	o o
A	Δ.2 o

PΔ.1 " V_o is a white point "

	V	~ V	⇒ RΔ.1
V	o	o	o
K	o.o	1.o	
S	o	o	o

PΔ.2 " Diagonal point "

	V	= V	∨	(	0	V	)	= V	⇒ RΔ.2
V	o	o				o		o	o
K	o	1				o		1	
S	1.3	1.3				1.3		1.3	o

PΔ.3 " Corner point "

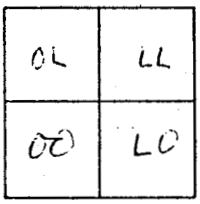
	V	= ooo	∨	V	=. LLL	∨	V	= ooo	∨	V	= LLL	⇒ RΔ.3
V	o			o			o			o		o
K	o			o			1			1		
S	1.3			1.3			1.3			1.3		o

3) Division of the field into sub – areas

	R	(V)	⇒ R
V		o	o
A		Δ.2	1.2

PΔ.4 Quadrant of a point

	(V	, V)	⇒ R Δ.4
V	o	o	o
K	o.2	1.2	
S	o	o	1.2

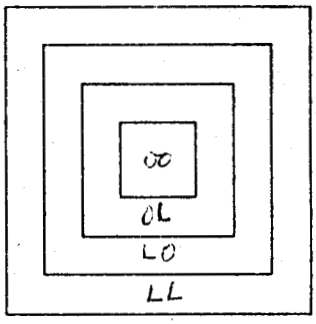


PΔ.5 Zone of a point

Auxiliary program :
ordinate ' s distance from the center

	V	⩾ 4 → V–4	⇒ R Δ.5
V	o	o	o
S	1.3		1.2

	V	< 4 → (3–V)	⇒ R Δ.5
V	o	o	o
S	1.3		1.2



PΔ.6

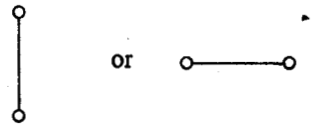
	Min (R Δ.5 (V) , R Δ.5 (V)) ⇒ R Δ.6
V	o o o
K	o 1
S	1.3 1.3 1.2

4) Propositions in the location of two points relative to each other

	R (V , V) ⇒ R
V	o 1 o
A	Δ.2 Δ.2 o

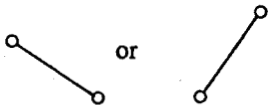
PΔ.8 Orthogonal relation

	V ≠ V ∧ V = V ∨ V = V ⇒ RΔ8
V	o 1 o 1 o 1 o
K	o o 1 1
A	Δ.2 Δ.2 Δ.1 Δ.1 Δ.1 Δ.1 o



PΔ.9 Diagonal Relation

	V ≠ V ∧ |V – V | = |V – V | ⇒ RΔ9
V	o 1 o 1 o 1 o
K	o o 1 1
A	Δ.2 Δ.2 Δ.1 Δ.1 Δ.1 Δ.1 o

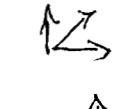


PΔ.10 Knight relation

	(|V – V| = L ∧ |V – V| = Lo) ∨ (|V – V| = Lo ∧ |V – V| = L) ⇒ RΔ.10
V	o 1 o 1 o 1 o 1 o
K	o o 1 1 o o 1 1

PΔ.11 Queen relation

	R Δ.8 (V , V) ∨ R Δ.9 (V , V) ⇒ R Δ.11
	o 1 o 1 o



PΔ.12 White pawn can move

	V – V = 0 ∧ [V – V = L ∨ (V = ooL ∧ V = oLL)] ⇒ RΔ.12
V	o 1 1 o o 1
K	o o 1 1 1 1

PΔ.13 Black pawn can move

	V – V = 0 ∧ [V – V = L ∨ (V = LLo ∧ V = Loo)] ⇒ RΔ.13
V	o 1 o 1 o 1
K	o o 1 1 1 1

PΔ.14 White pawn can capture

$$\begin{array}{c|cccccc} & |V-V| & = L \wedge & (V-V) & = L & \Rightarrow R\Delta.14 \\ V & o \quad 1 & & 1 \quad o & & \\ K & o \quad o & & 1 \quad 1 & & \end{array}$$

PΔ.15 Black pawn can capture

$$\begin{array}{c|cccccc} & |V-V| & = L \wedge & (V-V) & = L & \Rightarrow R\Delta.15 \\ V & o \quad 1 & & 1 \quad 1 & & \\ K & o \quad o & & 1 \quad o & & \end{array}$$

PΔ.16 No move or capture relation

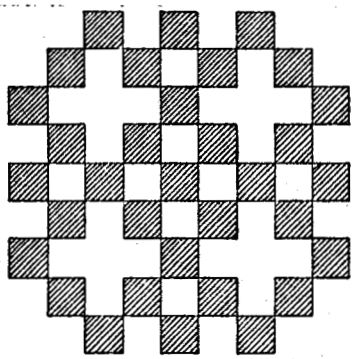
$$\overline{R\Delta.10}\,(V, V) \wedge \overline{R\Delta.11}\,(V, V) \Rightarrow R\Delta.16 \qquad (o\ 1 \quad o\ 1)$$

PΔ.17 The points are adjacent

$$\begin{array}{c|cccccc} & V \neq V \wedge |V-V| & \leqslant L \wedge & |V-V| & \leqslant L & \Rightarrow R\Delta.17 \\ V & o \quad 1 \quad o \quad 1 & & o \quad 1 & & \\ K & o \quad o & & 1 \quad 1 & & \end{array}$$

PΔ.18 There is a point in knight relation to the two points given

signalization of threat to two pieces attacked by a knight



Implicit expression

$$(Ex)\,[\,R\Delta.10\,(V, x) \wedge R\Delta.10\,(V, x) \wedge V \neq V\,] \qquad (o \quad 1 \quad o \quad 1)$$

Explicit expression

$$\begin{array}{c|cccccc} & (V \neq V) \wedge [\,R\Delta.1\,(V) & \sim & R\Delta.1\,(V)\,] & \wedge & |V-V| \leqslant 4 \wedge |V-V| \leqslant 4 \\ V & o \quad 1 & & o \quad 1 & & 1 \quad o \quad 1 \quad o \\ K & & & & & o \quad o \quad 1 \quad 1 \end{array}$$

$$\wedge\,[R\Delta.9\,(V,V) \rightarrow \overline{R8.2}\,(|V-V|)\,] \Rightarrow R\Delta.18 \qquad (o\ 1 \quad 1\ o)$$

V_o and V_1 are different points of the same color (RΔ.1) and both the horizontal and the vertical differences of their coordinates, taken absolutely are less then 4, and if the points are located in diagonal relation to each other, the difference may not be even (R8.2) .

PΔ.19 There are points located between the two given points

$$R_{\Delta}.11\,(V,V)_{o\,1} \wedge \overline{R_{\Delta}.17\,(V,V)_{o\,1}} \Rightarrow R_{\Delta}.19$$

The given points are located in orthogonal or diagonal relation and are not adjacent.

5) Propositions on the location of three points relative to each other

	$R(V_o, V_1, V_2) \Rightarrow R_o$
V	
A	$\Delta.2 \quad \Delta.2 \quad \Delta.2 \quad o$

Restriction :

$$V_o \neq V_1 \wedge V_o \neq V_2 \wedge V_2 \neq V_3$$

(All point locations are different form each other) .

PΔ.24 All three points lie on a horizontal line

	$V_o = V_1 \wedge V_o = V_2 \Rightarrow R_{\Delta}.24$
V	o 1 o 2
K	1 1 1 1

PΔ.25 All three points lie on a vertical line

	$V_o = V_1 \wedge V_o = V_2 \Rightarrow R_{\Delta}.25$
V	o 1 o 2
K	o o o o

PΔ.26 They lie on the same orthogonal line

$$R_{\Delta}.24\,(V,V,V)_{o\,1\,2} \vee R_{\Delta}.25\,(V,V,V)_{o\,1\,2} \Rightarrow R_{\Delta}.26$$

PΔ.27 They lie on the same diagonal line

$$R\Delta.9\,(V,V)_{o\,1} \wedge R\Delta.9\,(V,V)_{2\,2}$$

	$\wedge[(\mathrm{Pos}(V-V) \sim \mathrm{Pos}(V-V)) \sim (\mathrm{Pos}(V-V) \sim \mathrm{Pos}(V-V))] \Rightarrow R_{\Delta}.27$							
V	1	o	1	o	2	o	2	o
K	o	o	1	1	o	o	1	1

PΔ.28 They lie on a straight line

R Δ.26 (V, V, V) $\vee$ R Δ.27 (V, V, V) $\Rightarrow$ Ger (V, V, V)
o 1 2 o 1 2 o 1 2

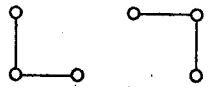
PΔ.29 V lies between V and V
o 1 2

V	RΔ.28 (V, V, V) $\wedge$ [Pos (V − V) $\not\sim$ Pos (V − V)] $\vee$ [Pos (V − V) $\not\sim$ Pos (V − V)]
	o 1 2 1 o 2 1 1 o 2 1 o
K	o o o o 1 1 1

$\Rightarrow$ R Δ.29

PΔ.30 V, V, V are located on the edges of an rectangular triangle
o 1 2

V on top
o



V	[(V = V) $\wedge$ (V = V)] $\vee$ [(V = V) $\wedge$ (V = V)] $\Rightarrow$ R Δ.30
	o 1 o 2 o 2 o 1
K	o o 1 1 o o 1 1

6) Formation of sets of points

PΔ.32 List of the points located between V and V
o 1

V	R (V , V) $\Rightarrow$ R
	o 1 o
A	Δ.2 Δ.2 $\square\times\Delta$.2

Implicit expression :

V	$\hat{x}$ [R Δ.29 (x , V , V)] $\Rightarrow$ R Δ.32
	o 1
A	Δ.2 Δ.2 Δ.2 Δ.2 $\square\times\Delta$.2

PΔ.32.1 Subprogram to PΔ.32

List of coordinates located between two given coordinates :

V	R (V , V) $\Rightarrow$ R Δ.32.1
	o 1 o
S	1.3 1.3 $\square\times$1.3

V	V $\geqslant$ V $\Rightarrow$ z	z $\rightarrow$ (+ $\Rightarrow$ δ)	$\bar{z}$ $\rightarrow$ (− $\Rightarrow$ δ)		
	1 o o	o	o		
S	1.3 1.3 o				

V	V δ 1 $\Rightarrow$ z	W [z	$\neq$ V	$\rightarrow$ [z	$\Rightarrow \mu$ R	z	δ 1 $\Rightarrow$ z]]					
	o 1	1	1	1	o	o	o					
S	1.3 1.3	1.3	1.3	1.3		1.3	1.3					

P△.32 Explicit program

	$\overline{V} = \overline{V} \Rightarrow z$			$\overline{V} = \overline{V} \Rightarrow z$		
V	o	1	o	o	1	1
K	o	o		1	1	
S	1.3	1.3		1.3	1.3	o

	$z \to$	$R\triangle.32.1\ (V, V) \Rightarrow z$				$z \to$	$R\triangle.32.1\ (V, V) \Rightarrow z$			
V	o	o	o	1	2	1	o	o	1	3
K			o	o				1	1	
S	o		1.3	1.3	□X1.3	o		1.3	1.3	□X1.3

	$z \wedge z \to$	$Qz\ (z, z) \Rightarrow R$		
V	o 1	2	3	o
S	o o	□X1.3	□X1.3	□X△.2

	$z \wedge \overline{z} \to$	$Qz\ (V, z) \Rightarrow R$			$\overline{z} \wedge z \to$	$Qz\ (z, V) \Rightarrow R$		
V	o 1	o	3	o	o 1	2	o	o
K		o					1	
S	o o	1.3	□X1.3	□X△.2	o o	□X1.3	1.3	□X□.2

P△.34 List of points in knight relation to a given point.

Implicit expression :

	$\hat{x}$	$R\triangle.10\ (V, x)$		
V		o		
A	△.2	△.2	△.2	

Explicit program

(Constructive method with variation of parameters)

```
V | 0 ⇒ z
  |   1
S |   1.3

W[ | z → [ Lo ⇒ z | oL ⇒ z ] | z̄ → [ oL ⇒ z | Lo ⇒ z ]
V | 1       2       3        | 1       2       3
K | o                        | o
S | o       1.2     1.2      | o       1.2     1.2

  | z → ( + ⇒ δ ) | z̄ → ( − ⇒ δ )
V | 1       1     | 1       1
K | 1             | 1
S | o             | o

  | z → ( + ⇒ δ ) | z̄ → ( − ⇒ δ )
V | 1       2     | 1       2
K | 2             | 2
S | o             | o

  | V  δ z  ⇒ z  | V  δ z  ⇒ z
V | o  1 2    4  | o  2 3    5
K | o             | 1
S | 1.3  1.2  1.4 | 1.3  1.2  1.4

  | (z  ⩾ 0 ∧ z  < Looo ∧ z  ⩾ 0 ∧ z  < Looo ) ⇒ z
V | 4       4       5       5                   6
S | 1.4     1.4     1.4     1.4                 o

  | z → [ (z, z, z ) ⇒ R ] [ z, z, z ) ⇒ R ]
V | 6    [ 4 4 4     o   ]  5 5 5      o   ]
K |      [ o 1 2     o   ]  o 1 2      1   ]
S | o    [ o o o     1.3 ]  o o o      1.3 ]

  | z = LLL ⇒ Fin2 | z + 1 ⇒ z
  | 1              | 1       1
```

Meaning of the intermediate values :

Z_1 = auxiliary value for the variation of points

Z_2 = horizontal distance between V_o and the next point.

Z_3 = vertical distance between V_o and the next point.

Z_4, Z_5 = coordinates of the point.

Z_6 = " The next point is located within the field " .

The corresponding sets of points for the relations of other pieces can be similarily developed. For the following the implicit expressions are at first sufficient.

II. The Point – Occupation

The points (squares) of the chess field can be occupied by pieces. In order to specify the state of occupation the point notation is supplemented by an occupation notation. There are six types of pieces (P, Kt, B, R, Q, K) of white and black color respectively. Any point may not be occupied. These 13 variations of the state of occupation of a point can be specified by 4 Yes – No – Values .

1) Introduction of the " occupation notation "

$A\Delta 3 = S1.4$
$B\Delta 3$

The code is represented by a list :

0	1	2	3	Meaning
–	–	–	–	not occupied
+	–	–	–	W.P.
–	+	–	–	W.Kt
[+	+	–	–	–]
–	–	+	–	W.K.g ← Wh. V
+	–	+	–	W.R
–	+	+	–	W.B
+	+	+	–	W.Q
[–	–	–	+	–]
+	–	–	+	Bl.S.P
–	+	–	+	Bl.S.Kt ← Bl V
[+	+	–	+	–]
–	–	+	+	Bl.S.K.g
+	–	+	+	Bl.S.R
–	+	+	+	Bl.S.B
+	+	+	+	Bl.S.Q

The undefined cases are excluded. Thus we have the restriction formula :

$B\Delta 3 =$

0	1	2	3
–	–	–	–
+	–	–	–
–	+	–	–
–	–	+	–
+	–	+	–
–	+	+	–
+	+	+	–
+	–	–	+
–	+	–	+
–	–	+	+
+	–	+	+
–	+	+	+
+	+	+	+

2) Operations with Occupation Notations $A\Delta 3$:

A number of propositions on the components of the occupation notation V_o can be derived :

$V \neq 0$ " occupied "

	$V \neq 0 \wedge \overline{V}$	
V	o	o
K		3

" occupied by white "

	V
V	o
K	3

" occupied by black "

V	=	+	–	–	o	occupied by pawn
V	=	–	+	–	o	" " knight
V	=	+	–	+	o	" " rook
V	=	–	+	+	o	" " bishop
V	=	+	+	+	o	" " queen
V	=	–	–	+	o	" " king

P∆.48 The adjacent points are under threat

$V \wedge (V \sim V) \Rightarrow R\,\Delta.48$

V	o		o	o
K	2		o	1
S	o		o	o o

P∆.49 The orthogonal lines are under threat

$V \wedge V \Rightarrow R\,\Delta.49$

V	o	o
K	o	2

P∆.50 The diagonals are under threat

$V \wedge V \Rightarrow R\,\Delta.50$

V	o	o
K	o	2

P∆.51 Occupied by a minor piece (bishop, knight)

$\overline{V} \wedge V \Rightarrow R\,\Delta.51$

V	o	o
K	o	1

P∆.52 Occupied by major piece (queen, rook)

$V \wedge V \Rightarrow R\,\Delta.52$

V	o	o
K	o	2

P∆.53 Occupied by a minor piece a major piece or a king

$V \Rightarrow R\,\Delta.53$

V	o
K	2

P△.54 Occupied by a minor piece or a major piece, king excluded.

	V ∧ V ∨ V ⇒ R △.54
V	o o o
K	2 o 1

Relations between two occupation notations .

	R (V , V) ⇒ R
V	o 1 o
A	△3 △3 o

P△.60 " Occupied by pieces of same color " .

	V ≠ 0 ∧ V ≠ 0 ∧ (V ~ V) ⇒ R △.60
V	o 1 o 1
K	3 3

P△.61 " Occupied by pieces of different color " .

	V ≠ 0 ∧ V ≠ 0 ∧ (V ≁ V) ⇒ R △.61
V	o 1 o 1
K	3 3

P△.62 " The occupation is equivalent (if for instance bishop and knight are rated equivalent)

(0, P, (Kt, B) , R, Q, Kg)

	[(V, V, V) = (V, V, V)] ∨ [R △.51 (V) ∧ R △.51 (V)] ⇒ R △.62
V	o o o 1 1 1 o 1
K	o 1 2 o 1 2
A	o o o o o o △3 △3 o

Alternate form of P△.62

Introduction of a valuation table in the form of a constant C△0.1.

Comp. of A△3	0, 1, 2	Piece	Level of valuation	C△0.1
	– – –	o	o	ooo
	+ – –	P	1	ooL
	– + –	Kt	2	oLo
	+ + –	–		ooo
	– – +	Kg	5	LoL
	+ – +	R	3	oLL
	– + +	B	2	oLo
	+ + +	Q	4	Loo

PΔ.62

	(	C		(	V, V, V))	=	(	C		(	V, V, V))	⇒	R	Δ.62		
V		Δ0.1			o o o			Δ0.1			1 1 1					
K					o 1 2						o 1 2					
S		1.3			o o o			1.3			o o o					

PΔ.63 V_1 is univocally occupied superior to V_o

	K	Δ.1	(	V, V, V)	<	K	Δ.1	(	V, V, V)	⇒	R	Δ.63			
V				o o o					1 1 1		o				
K				o 1 2					o 1 2						
S	1.3			o o o					o o o		o				

3) The " Point Occupied Notation "

AΔ4 = (AΔ2, AΔ3)

Operations with the " point occupied " notation :

	R	(	V	)	⇒ R
V			o		o
A			Δ4		o

PΔ.64 " Occupation is possible "

	[	(	V	= +–––) →	$\overline{V}$	= O]	∧	[	(V	= +––+) →	$\overline{V = LLL}$	⇒	R	Δ.64				
V			o		o				o		o								
K			o.1		1.o				o.1		1.o								
A			Δ3		Δ1				Δ3		Δ1							o	

(A white pawn must not be located on points with the vertical coordinate zero and a black pawn must not be located on points with the vertical coordinate seven).

Relations between two " point occupied " notations.

	R	(	V	,	V	)	⇒ R
V			o		1		o
A			Δ4		Δ4		o

PΔ.72 Occupation condition for the move $V_o - V_1$

(V_o is occupied and V_1 is not occupied or occupied by a piece of the other color).

	V	≠	0	∧	(	V	=	0	∨	(	V	≠	V	))	⇒	R	Δ.72		
V	o					1					o		1			o			
K	1					1					1.3		1.3						
A	Δ3					Δ3					o		o			o			

PΔ.73 Move condition, without consideration of the points located between V_o and V_1.

	V	= 0 $\wedge$	[	[V	= +–––	RΔ12 (V ,	V)	]	]	$\Rightarrow$ RΔ.73	Wh.P
V	1			o			o 1				
K	1			1			o o				
A	Δ3			1.3			Δ2 Δ2				
				[V	= +––+	RΔ13 (V ,	V)	]			Bl.P
V				o			o 1				
K				1			o o				
A				1.3			Δ2 Δ2				
				[V	= –+–o	RΔ10 (V ,	V)	]			Kt
V				o			o 1				
K				1			o o				
A				1.3			Δ2 Δ2				
				[V	= ––+o	RΔ17 (V ,	V)	]			Kg
V				o			o 1				
K				1			o o				
A				1.3			Δ2 Δ2				
				[RΔ49 (V	)	$\wedge$ RΔ8 (	V , V)	]			R,Q
V				o			o 1				
K				1			o o				
A				1.3			Δ2 Δ2				
				[RΔ50 (V	)	$\wedge$ RΔ9 (	V , V)	]			B,Q
V				o			o 1				
K				1			o o				
A				1.3			Δ2 Δ2				
			]						]		

PΔ.74 Capture – and guard condition respectively, without consideration of the points located between V_o and V_1.

```
V |[[ V   = +---  RΔ14 ( V , V )]]   ⇒ RΔ.74   Wh.P
K |[[ o                    o   1 ]]
A |[[ 1                    o   o ]]
  |[[ Δ3                   Δ2  Δ2]]

V |[[ V   = +--+  RΔ15 ( V , V )]]              Bl.P
K |[[ o                    o   1 ]]
A |[[ 1                    o   o ]]
  |[[ Δ3                   Δ2  Δ2]]

V |[[ V   = -+-o  RΔ10 ( V , V )]]              Kt
K |[[ o                    o   1 ]]
A |[[ 1                    o   o ]]
  |[[ Δ3                   Δ2  Δ2]]

V |[[ V   = --+o  RΔ17 ( V , V )]]              Kg
K |[[ o                    o   1 ]]
A |[[ 1                    o   o ]]
  |[[ Δ3                   Δ2  Δ2]]

V |[[ RΔ49 ( V  )  RΔ8 ( V , V )]]              R,Q
K |[[      o          o   1 ]]
A |[[      1          o   o ]]
  |[[      1.3        Δ2  Δ2]]

V |[[ RΔ50 ( V  )  RΔ9 ( V , V )]]              B,Q
K |[[      o          o   1 ]]
A |[[      1          o   o ]]
  |[[      1.3        Δ2  Δ2]]
```

PΔ.75 " En passant " capture condition

V_o = point of the pawn

V_1 = point to which the pawn is moved

V_2 = point on which the beaten piece is located

```
    | R ( V , V , V ) ⇒ R Δ.75
V   |     o   1   2     o
K   |     Δ4  Δ4  Δ4    o
```

```
    | [ V   = +–––∧V  = ooL∧V  = oLL∧R Δ14 ( V , V ) ∧ V ]
V   | [ o       o        1                  o   2     2  ]
K   | [ 1       o.1      o.1                o   o     1.3]
A   | [ Δ3      Δ1       Δ1                 Δ2  Δ2    o  ]
```

```
    | [ V   = +–––1∧V = oLL∧V  = Loo∧RΔ15 ( V   , V )∧V ≠ 0 ∧ V̄ ]
V   | [ o       o        1                  o     2  2       2  ]
K   | [ 1       o.1      o.1                o     o  1       1.3]
A   | [ Δ3      Δ1       Δ1                 Δ2    Δ2 Δ3      o  ]
```

```
    | ∧( V  = V  ) ∧ ( V  = 0 ) ⇒ R Δ.75
V   |   o    1      1
K   |   o.o  o.o    1
A   |   Δ1   Δ1     Δ2          o
```

III. The Field Occupation

1) Introduction of New Structures

The list of the occupation notations AΔ4 assigned to the 64 points of the chess–field is the " field occupation " .

$$A\Delta 5 = 64 \times A\Delta 3$$

It is advantageous to introduce another notation which consists of the field occupation notation, supplemented by the coordinate notation of the points.

The list of pairs of this point occupied notation is of a restricted variability. (The front elements of the pairs are constant)

$$A\,\Delta\,6 = 64 \times A\,\Delta\,4$$
$$B\,\Delta\,6$$

```
    | V  = i   ⇒ BΔ6
K   | i.o
A   | Δ    1.6
```

Alternative formulation :

$$A\,\Delta\,6 = Qz\ (\ A\,\Delta\,2, A\,\Delta\,5\ )$$

The start occupation is specified by the constants.

C Δ 5 ; (A Δ 5) and

C Δ 6 ; (A Δ 6)

C Δ 6			Meaning	
A Δ 2		C Δ 5	point	piece
– – –	– – –	+ – + –	a1	Wh.R
+ – –	– – –	– + – –	b1	Wh.Kt
– + –	– – –	– + + –	c1	Wh.B
+ + –	– – –	+ + + –	d1	Wh.Q
– – +	– – –	– – + –	e1	Wh.Kg
+ – +	– – –	– + + –	f1	Wh.B
– + +	– – –	– + – –	g1	Wh.Kt
+ + +	– – –	+ – + –	h1	Wh.R
– – –	+ – –	+ – – –	a2	Wh.P
:	:	:	:	:
+ + +	+ – –	+ – – –	h2	Wh.P
– – –	– + –	– – – –	a3	–
:	:	:	:	:
+ + +	+ + +	+ – + +	h8	Bl.R

In certain problems, the entire field occupation is not of interest, but only the number of pieces of each type available.

First a constant CΔ0.2, which consists of the list of the different types of pieces, is required.

				Meaning
	C	=	+ – – –	Wh.P
V	Δ0.2		– + – –	Wh.Kt.
A	12XA Δ 3		– + + –	Wh.B.
			+ – + –	Wh.R.
			+ + + –	Wh.Q.
			– – + –	Wh.Kg.
			+ – – +	Bl.P.
			– + – +	Bl.Kt.
			– + + +	Bl.B.
			+ – + +	Bl.R.
			+ + + +	Bl.Q.
			– – + +	Bl.Kg.

To the list AΔ7 can be assigned by horizontal composition ~~c Δ4 AA~~

AΔ7 = 12XS1.4

The composition with CΔ0.2 results in the list of pairs AΔ8.

AΔ8 = Qz (C Δ0.2, AΔ7)

The state at the start is represented by the constants CΔ7 and CΔ8 :

C Δ 8		Meaning	
C Δ 0.2	C Δ 7		
0 1 2 3	3210	piece	number
+ – – –	Looo	Wh.P	8
– + – –	ooLo	Wh.Kt.	2
– + + –	ooLo	Wh.B.	2
+ – + –	ooLo	Wh.R.	2
+ + + –	oooL	Wh.Q.	1
– – + –	oooL	Wh.Kg.	1
+ – – +	Looo	Bl.P.	8
– + – +	ooLo	Bl.Kt.	2
– + + +	ooLo	Bl.B.	2
+ – + +	ooLo	Bl.R.	2
+ + + +	oooL	Bl.Q.	1
– – + +	oooL	Bl.Kg.	1

2) Operations with AΔ7

PΔ.96 Development of AΔ7 from AΔ5

	R (V) ⇒ R
V	o o
A	Δ5 Δ7

	W1 [N [x̂ (x ∈ V ∧ x = C) ⇒ R]
V	o Δ0.2 o
K	i 1
A	Δ3 Δ5 Δ3 1.4

Propositions on AΔ7

	R (V) ⇒ R
V	o o
A	Δ7 o

PΔ.97 The field is completely occupied

(V = C) ⇒ R Δ.97
o Δ7 o

PΔ.98 The occupation is partialy as at the start or equal to it.

	W1 [V ⩽ C] ⇒ R Δ. 98
V	o Δ7 o
K	i i
S	1.4 1.4 o

PΔ.99 Number of white and black pieces :

	R (V) ⇒ (R , R)
V	o o 1
A	Δ7 1.5 1.5

R_o = number of white pieces

R_1 = number of black pieces

	V + V + V + V + V ⇒ R Δ.99						V + V + V + V + V ⇒ R Δ.99					
V	o	o	o	o	o	o	o	o	o	o	o	1
K	o	1	2	3	4		6	7	8	9	10	

PΔ.100 Number of redundant minor and major pieces relative to start occupation.

	R (V) ⇒ (R , R)
V	o o 1
A	Δ7 1.4 1.4

R_o = redundant white minor and major pieces

R_1 = redundant black minor and major pieces

Fpos (V–Lo)	+ Fpos (V–Lo)	+ Fpos (V–Lo)	+ Fpos (V-L)	⇒ RΔ.100	
o	o	o	o	o	
1	2	3	4		
Fpos (V–Lo)	+ Fpos (V–Lo)	+ Fpos (V–Lo)	+ Fpos (V–L)	⇒ RΔ.100	
o	o	o	o	1	
7	8	8	4		

PΔ.101 " Occupation is possible with regard to the exchange of pawns " .

Detailed conditions

1) Both kings are present
2) Number of the white pieces $\leqslant 16$
3) Number of the black pieces $\leqslant 16$
4) Number of white pawns $\leqslant 8$ minus the number of redundant white minor and major pieces
5) Number of black pawns $\leqslant 8$ minus the number of redundant black minor and major pieces
6) Number of pieces at least eqal to 3.

```
   | R ( V   ) ⇒ R Δ.101
V  |     o       o
A  |     Δ7      o

   | ( V = L ) ∧ ( V = L ) ∧ ( RΔ.99 ( V ) ⩽ Loooo ) ∧ ( RΔ.99 ( V ) ≤ Loooo )
V  |   o         o         o     o                    1      o
K  |   5         11

   | ∧( V ⩽ Looo – RΔ.100 ( V )) ∧ ( V ⩽ Looo – RΔ.100 ( V ))
V  |   o          o       o          o              1        o
K  |   o                                6

   | ∧ RΔ.99 ( V ) + RΔ.99 ( V ) ⩾ LL ⇒ R Δ.101
V  |   o       o     1       o           o
K  |
```

PΔ.102 White is equal or superior to black in any type of piece.

```
   | R ( V   ) ⇒ RΔ.102
V  |     o       o
A  |     Δ7      o
```

```
( V⩾V ) ∧ ( V⩾V ) ∧ ( V⩾V ) ∧ ( V⩾V ) ∧ ( V⩾V ) ∧ ( V⩾V ) ⇒ PΔ.102
  o  o      o  o      o  o      o  o      o  o      o  o
  o  6      1  7      2  8      3  9      4  10     5  11
```

PΔ.103 Corresponding specification for black.

PΔ.104 Inivocal superiority of white according to the valuation of pieces in table PΔ.62

Since there is always only one king of each color present, the other 5 types of pieces only are relevant :

P, Kt, B, R, Q ,

These are divided into four value classes :

P	a
Kt,B	b
R	c
Q	d

For the values a, b, c and d only the following is fixed :

$$a > 0$$
$$b-a > 0$$
$$c-b > 0$$
$$d-c > 0$$

According to this determination superiority can be identified in some cases.

If α, β, γ, and δ are surplus, respectively the types of pieces a, b, c, and d in white are deficient then the valuation of white produces the following result.

$$x = \alpha \times a + \beta \times b + \gamma \times c + \delta \times d$$

$$\begin{aligned} x = {} & \alpha \times a \\ & + \beta \times a + \beta \times (b-a) \\ & + \gamma \times a + \gamma \times (b-a) + \gamma \times (c-b) \\ & + \delta \times a + \delta \times (b-a) + \delta \times (c-b) + \delta \times (d-c) \end{aligned}$$

$$\begin{aligned} x = {} & (\alpha + \beta + \gamma + \delta) \times a \\ & + (\beta + \gamma + \delta) \times (b-a) \\ & \quad (\gamma + \delta) \times (c-b) \\ & \quad \delta \times (d-c) \end{aligned}$$

If $x > o$, then at least one of the following expressions must be positive and none of them is allowed to be negative.

$$\begin{aligned} & (\alpha + \beta + \gamma + \delta) \\ & \quad (\beta + \gamma + \delta) \\ & \quad\quad (\gamma + \delta) \\ & \quad\quad\quad \delta \end{aligned}$$

This results in the following program :

PΔ.104

	R	(	V	)	⇒	R	Δ.104							
V			o			o								
A			Δ7			o								
	V	–	V	⇒	z			z	<	0	⇒	Fin	z	> 0 ⇒ V R
V	o		o		o			o					o	o
K	4		10											
	z	+	V	–	V	⇒	z	z	<	0	⇒	Fin	z	> 0 ⇒ V R
V	o		o		o		o	o					o	o
K			3		9									
	z	+	V	+	V	–	V	–	V	⇒	z	z	<	0 ⇒ Fin	z	> 0 ⇒ V R
V	o		o		o		o		o		o	o			o	o
K			1		2		7		8							
	z	+	V	–	V	⇒	z	z	<	0	⇒	Fin	z	> 0 ⇒ V R
V	o		o		o		o	o					o	o
K			o		6									

PΔ.105 Corresponding program for black exchange of :

V , V		V , V		V , V		V , V		V , V		V , V	
o	o	o	o	o	o	o	o	o	o	o	o
o	6	1	7	2	8	3	9	4	10	5	11

3) Operations with the Field Occupation (AΔ5, AΔ6)

The following programs sometimes contain repetitions and implicit partial expressions. Consequently, they do not always show the most convenient solution for computation.

PΔ.128 " The move form V_1 to V_2 is allowed " .

	R (V	,	V	,	V	)	⇒	R Δ.128							
V	o		1		2			o							
A	Δ5		Δ2		Δ2			o							
	(V	,	V	⌐	V	)	⇒	z	(V	,	V	⌐	V	)	⇒	z
V	1		o	1				o	2		o	2				1
K				⌙								⌙				
A	Δ2		Δ3	Δ2				Δ4	Δ2		Δ3	Δ2				Δ4
	R Δ.73 (z	,	z	)	∨	(R Δ.74 (z	,	z	)	∧	(z	≠	z	))		
V		o		1					o		1		o			
K											1.3		1.3			
A		Δ4		Δ4					Δ4		Δ4		o		o	

$$\bigwedge [\, R\Delta 19 (z_o , z_1) \rightarrow (x) \; (x \in R\Delta 32 (z_o , z_1) \rightarrow V_x = 0)] \Rightarrow R\Delta 128$$

V			o		1						o		1		o
K			o		o						o		o		x
A			Δ2		Δ2						Δ2		Δ2		Δ3

V_o = field occupation (AΔ5)

V_1 = point from which the move starts (AΔ2)

V_2 = point to which the move is made (AΔ2)

Z_o = the point occupied notation (AΔ4) assigned to V_1

Z_1 = the point occupied notation (AΔ4) assigned to V_2

x = point located between V_1 and V_2 (AΔ2)

Between V_1 and V_2 either the move condition RΔ.73 is true or, if V_1 and V_2 are occupied by pieces of a different color ($Z_{1.3} \neq Z_{1.3}$) the capture condition RΔ.74 is true and, if there are points located between V_1 and V_2 (RΔ.19), then these (RΔ.32) must be unoccupied ($V_x = o$).

PΔ.128 does not take castling into consideration since this cannot be supervised simply on the grounds of field occupation.

PΔ.129 " The piece located in point V_1 guards the point V_2 or attacked it. (It can capture the piece of the opposite color, if there is any located in V_2) .

As PΔ.128 , however, instead of the expression

$$R\Delta.73\,(z_0, z_1) \vee (R\Delta.74\,(z_0, z_1) \wedge (z_{1.3} \neq z_{1.3}))$$

is substituted

$$R\Delta.74\,(z_0, z_1)$$

PΔ.130 "A certain piece exists, which can move to V_1 conditionally" (possibly restricted because of check uncovered :

V	$R(V_0, V_1) \Rightarrow R_0$
K	$\Delta 6 \quad \Delta 2 \quad 0$

V	$(Ex_0)[x \in V \wedge x \neq 0 \wedge \bar{x}_{1.3} \wedge R\Delta 128\,(Sp1\,(V_0), x, V_1)] \Rightarrow R\Delta.130_0$
K	$1 \quad 1.3 \quad 0$
A	$\Delta 4 \quad \Delta 6 \quad \Delta 3 \quad 0 \quad \Delta 5 \quad \Delta 2\Delta 2 \quad 0$

PΔ.131 As PΔ.130, however, for black

K	$x_{1.3}$	instead of $x \neq 0 \wedge \bar{x}_{1.3}$

PΔ132 " The point V_1 is conditionally guarded, or attacked by white " .

As PΔ.130, however, RΔ.129 instead of RΔ.128

PΔ.133 As PΔ.132, however, for black

K	$x_{1.3}$	instead of $x \neq 0 \wedge x_{1.3}$

If V_1 is occupied by black, then follows :

$$R\Delta.130 \sim R\Delta.132$$

If V_1 is occupied by white , then follows :

$$R\Delta.131 \sim R\Delta.133$$

The same is true, if V_1 is occupied by a minor or major piece.

PΔ.134 " The white king is in check "

Supposition . The occupation corresponds to PΔ.101

```
   | R ( V   ) ⇒ R Δ.134
V  |     o       o
A  |     Δ6      o
```

```
   | x́ [ x ∈ V ∧ x = ( ––+– )] ⇒ z  |  R Δ133 ( V   , z   ) ⇒ R Δ134
V  |           o                 o  |           o     o
K  |           ·   1               |                 o
A  | Δ4        Δ6                Δ4 |           Δ6    Δ2      o
```

PΔ.135 " The black king is in check "

As PΔ.134, however x = (––++)

and PΔ.132, instead of PΔ.133

PΔ.136 The field is transformed into the field R_o by the move $V_1 - V_2$.

Supposition : Move $V_1 - V_2$ is allowed .

```
   | R ( V   , V   , V   ) ⇒ R Δ.136
V  |     o     1     2       o
A  |     Δ6    Δ2    Δ2      Δ6
```

```
   | V ⇒ z  | z ⌐V.1 ⇒ z ⌐V.1 | 0 ⇒ z ⌐V.1 | z ⇒ R Δ.136
V  | o   o  | o   1    o   2  |     o   1  | o   o
K  |        |  ⌙      ⌙       |     ⌙      |
A  | Δ6  Δ6 | Δ3       Δ3     |     Δ3     | Δ6  Δ6
```

PΔ.137 The field V_o is transformed into the field R_o by en–passant capture

```
   | R ( V   , V   , V   , V   ) ⇒ R
V  |     o     1     2     3       o
K  |     Δ6    Δ2    Δ2    Δ2
```

Supposition :

```
   | R Δ.75 ( V ⌐V , V ⌐V , V ⌐V )
V  |          o   1   o   2   o   3
K  |           ⌙       ⌙       ⌙
A  |          Δ4  Δ2   Δ4  Δ2  Δ4  Δ2
```

```
   | V   ⇒ z  | z ⌐V.1 ⇒ z ⌐V.1 | 0 ⇒ z ⌐V.1 | 0 ⇒ z ⌐V.1 | z ⇒ RΔ.136
V  | o     o  | o   1    o   2  |     o   1  |     o   3  | o   o
K  |          |  ⌙      ⌙       |     ⌙      |     ⌙      |
A  | Δ6    Δ6 | Δ3       Δ3     |     Δ3     |     Δ3     | Δ6  Δ6
```

The program PΔ.140 to PΔ.143 correspond to the programs PΔ.130 to PΔ.133 : but the case of a check being discovered is excluded. A meaningful occupation according to PΔ.101 is pre-supposed.

```
   | R ( V , V ) ⇒ R
V  |     o   1     o
K  |     Δ6  Δ2    o
```

PΔ.140 A white piece exists which can be moved to V_1 without a check resulting (if a black piece is located there, this can be captured).

The program must contain the condition, that a white piece exists which can be moved to V_1 (PΔ.130)

By the condition, that in the game situation after the move

```
     | RΔ.136 ( V , x , V )
   V |          o     1
   K |                o
```

the white king is not in check the case of uncovering check is taken into account.
The formula is set up first without regard of en – passant capture.

```
   | ( Ex ) [ x ∈ V ∧ x ≠ 0 ∧ x̄ ∧ RΔ128 ( Spl ( V ), x , V )
V  |          o                          o       1
K  |              1      1.3                  o
A  |  Δ4        Δ6  Δ3    o          Δ5        Δ2  Δ2

   |                ∧ RΔ.134 ( RΔ.136 ( V , x , V )) ] ⇒ RΔ140.0
V  |                                    o     1
A  |                                    Δ6  Δ2  Δ2
```

PΔ.141 As PΔ.140, however, for black

```
   | x          instead of   x ≠ V ∧ x̄
K  | 1.3                              1.3
```

R_Δ.135 instead of R_Δ.134

PΔ.142 " The point V_1 is guarded by white or attacked " . (The guarded or attacked piece is not arrested because of uncovering check)

As P_Δ.140.0, however, R_Δ.129 instead of R_Δ.128

PΔ.143 As PΔ.142, however, for black

```
   | x          instead of   x = o ∧ x̄
K  | 1.3                              1.3
```

R_Δ.135 instead of R_Δ.134

Regarding the identity of the programs P_Δ.140 and P_Δ.142 , respectively P_Δ.141 and P_Δ.143, the same is true of the programs P_Δ.130 and P_Δ.133 (page 223).

4) Programs on the Freedom of Pieces to Move

Supposition : V_1 is occupied

	R (V , V) ⇒ R
V	o 1 o
A	Δ5 Δ2 o

PΔ.144 The piece located in V_1 can move conditionally (possibly it uncovers check)

	(Ex) [RΔ128 (V , V , x)] ⇒ RΔ.132
V	o 1
K	
A	Δ2 Δ5 Δ2 Δ2

PΔ.145 The piece located in V can move without uncovering check.

	Nr(V) ⇒ z	V [V ⇒ z
V	o o	o 1 1
K		
A	Δ5 Δ6	Δ3 Δ2 Δ3

	z → [$\overline{R\Delta 134}$ (z) ∧ (Ex) [RΔ128 (V , V , x) ∧ $\overline{R\Delta 134}$ (RΔ136 (z , V , x))] ⇒ RΔ145]
V	1 o o 1 o 1
K	3
A	o Δ6 Δ2 Δ5 Δ2 Δ2 Δ6 Δ2 Δ2 o

	z → [$\overline{R\Delta 135}$ (z) (Ex) [RΔ128 (V , V , x) ∧ $\overline{R\Delta 135}$ (RΔ136 (z , V , x))] ⇒ RΔ145]
V	1 o o 1 o 1
K	3
A	Δ6 Δ5 Δ2 Δ2 Δ6 Δ2 Δ2 o

V_o = start field occupation

Z_o = start field occupation, supplemented by point notation

V_1 = point to be investigated

x = point to which the piece located in V_1 may move.

RΔ.136 (Z_o, V_1, x) field occupation after the move

The cases, that V_1 is occupied by white ($\overline{Z}_3$) and that V_1 is occupied by black (Z_3) must be separated.

PΔ.146 The piece located in V_1 can be conditionally moved without being attacked in the new location (possibly it uncovers check).

	Nr (V) ⇒ z		V	V ⇒ z	
V	o	o	o	1	1
K					
A	Δ5	Δ6	Δ3	Δ2	Δ3

	(Ex)	[RΔ128 (V,	V,	x) ∧ [	[z →	$\overline{R\Delta 143}$ (RΔ136 (	z ,	V ,	x))]	]	]	⇒ RΔ146
V			o	1		1		o	1			
K						3						
A			Δ5	Δ2	Δ2			Δ6	Δ2	Δ2		
						v [z →	$\overline{R\Delta 142}$ (RΔ136 (	z ,	V ,	x))]		
V						1		o	1			
K						3						
A						o		Δ6	Δ2	Δ2		
							⇒ RΔ146					

PΔ.147 The piece located in V_1 can move without uncovering check and without being attacked in the position.

PΔ.147 results from a combination of PΔ.145 and PΔ.146

	Nr (V) ⇒ z		V	V ⇒ z	
V	o	o	o	1	1
K					
A	Δ5	Δ6	Δ3	Δ2	Δ3

	($\overline{z}$ →	[$\overline{R\Delta 134}$ (z)	∧	(Ex)	[RΔ128 (V ,	V ,	x) ∧ $\overline{R\Delta 134}$ (RΔ136 (	z ,	V ,	x))]	]	⇒ RΔ147
V	1	o			o	1			o	1			
K	3												
A	o	Δ6		Δ2		Δ5	Δ2	Δ2		Δ6	Δ2	Δ2	
					∧ $\overline{R\Delta 143}$ (RΔ136 (	z ,	V ,	x))					
V						o	1						
K													
A						Δ6	Δ2	Δ2					
	z →	[$\overline{R\Delta 135}$ (z)	∧	(Ex)	[RΔ128 (V ,	V ,	x) ∧ $\overline{R\Delta 135}$ (RΔ136 (	z ,	V ,	x))]	]	⇒ RΔ147
V	1	o			o	1			o	1			
K	3												
A	o	Δ6		Δ2		Δ5	Δ2	Δ2		Δ6	Δ2	Δ2	
					∧ $\overline{R\Delta 143}$ (RΔ136 (	z ,	V ,	x))					
V						o	1						
K													
A						Δ6	Δ2	Δ2					

PΔ.148 The white king can move without getting into check.

Supposition : Occupation corresponds PΔ.101

```
   | R ( V ) ⇒ RΔ148
V  |     o      o
A  |    Δ6      o

   |  x´ [ x ∈ V ∧ x = ––+– ] ⇒ z
V  |        o                    o
K  |
A  | Δ4     Δ6                   Δ4

   | ( Ex ) [ x ∈ V   RΔ17 ( z , x ) ∧ x = 0 ∨ x ∧ R̄Δ133 ( RΔ136 ( V , z , x ) , x ) ]
V  |          o          o                                    o   o
K  |                     o   o   1        1.3                     o   o   o
A  |  Δ4      Δ6         Δ2  Δ2                                Δ6  Δ2  Δ2
```

⇒ RΔ148

V_o = field occupation at the start

Z_o = point occupation notation for the white king.

" There is a point x adjacent to the point of the white king ($\underset{o}{Z}$) , which is not occupied (x = o) or which is occupied by black ($\underset{1}{\underset{3}{x}}$) and in the resulting field occupation (RΔ.136 (...)) the point x is not attacked by black ($\overline{\text{RΔ.135}}$)

PΔ.149 As PΔ.148, however, for the black king.

x = ––++ instead of x = ––+–

RΔ.132 instead of RΔ.133

PΔ.150 " There are no white pieces present apart from the white king or these cannot move ". (Supposition for " Draw ").

```
   | R ( V ) ⇒ RΔ150
V  |     o
A  |    Δ6      o

   | ( x ) [ x ∈ V ∧ x ≠ 0 ∧ x ≠ ––+– ∧ x̄ → R̄Δ145 ( Sp1 ( V ) , x ) ]
V  |          o                                                o
K  |               1        1          1.3                          o
A  |  Δ4      Δ6   Δ3       Δ3         o          Δ5                Δ2
```

PΔ.151 As PΔ.150, however, for black

```
   | x ≠ ––++ ∧ x          instead of  x ≠ o ∧ x ≠ ––+– ∧ x
K  | 1        1.3                      1       1        1.3
```

PΔ.152 A certain piece exists, which may move to a point between V_1 and V_2 without uncovering check."

```
  | R ( V , V , V ) ⇒ RΔ152        ⊢ RΔ8 (V , V ) V RΔ9 (V , V )
V |     o   1   2     o                    o   1          o   1
A |     Δ6  Δ2  Δ2    o
```

Supposition : V_o and V_1 are located on a straight line.

```
  | ( Ex ) ⎡ x ∈ V ∧ x ∈ RΔ32 ( V , V ) ∧ RΔ140 (V , x ) ⎤ ⇒ RΔ152
V |        ⎢     o                o   1           o       ⎥
K |        ⎢          o                                   ⎥
A |   Δ4   ⎣ Δ6  Δ2               Δ2  Δ2          Δ6  Δ2  ⎦
```

PΔ.153 As PΔ.132, however, for black

RΔ.141 instead of RΔ.140

5) The conditions for " Check" and " Draw "

```
PΔ.160    | R ( V ) ⇒ ( R , R )
        V |     o       o   1
        K |     Δ6      o   o
```

V_o = Field occupation

R_o = white king is mated

R_1 = white king is drawn

```
  | x́  ( x ∈ V ∧ x = ––+– ) ⇒ z
V |          o                o
K |              1
A | Δ4       Δ6               Δ4
```

```
  | x̂  ( RΔ129 (V , x , z ) ∧ x ) ⇒ z    | N( Z ) ⇒ z
V |            o       o           1     |    1       2
K |                o   o      1.3        |
A | Δ4         Δ6  Δ2  Δ2     o    □XΔ4  |  □XΔ4     1.2
```

```
  | (z = 0 ) ∨ R Δ 148 ( V ) ⇒ Fin
V |  2                   o
K |
A |  1.2                 Δ6
```

```
  | ( z ⩾ 2 ) ∨ ⎡ ‾RΔ142‾ (V , z ) ∧ (‾RΔ19‾ ( z , z ) ∨ ‾RΔ152‾ (V , z , z ) ⎤ ⇒ z
V |   2         ⎢          o   1               o   1                o   o   1  ⎥   3
K |             ⎢              0.0             o   0.0                  o   0.0⎥
A |   1.2       ⎣          Δ6  Δ2              Δ2  Δ2               Δ6  Δ2  Δ2 ⎦   o
```

```
  | R Δ 150 ( V ) ⇒ z | z ∧ z̄ ⇒ R | z ∧ z ⇒ R
V |           o     4 | 3   4    o | 3   4    1
K |                   |            |
A |           Δ6    o | o   o    o | o   o    o
```

Meaning of the intermediate values :

Z_0 = point occupied notation of the white king

Z_1 = list of the point occupied notations form which the king is attacked.

Z_2 = number of pieces attacking the king

Z_3 = " The white king is attacked by double check ($Z_2 \geqslant 2$) ,
or it can neither capture the attacking piece, nor can it be guarded by the interposition of a piece between the king and the attacking piece ".

Z_4 = no piece other than the king can move.

PΔ.161 As PΔ.160, however, for black

x = ––++	instead of	x = ––+–
RΔ.149	"	"	RΔ.148
RΔ.143	"	"	RΔ.142
RΔ.153	"	"	RΔ.152
RΔ.151	"	"	RΔ.150

V. The Game Situation

1) Introduction of new Types of Data

The field occupation information alone is not sufficient for the specification of the game situation. The following supplements are required :

a) Information as to whether white or black has to move

b) Information about the execution of castlings

c) Information, as to whether it is allowed to capture en–passant

To a) This is specified by a single Yes–No–Value.

To b) The conditions for castling are the following :

α) The points between king and rook must be unoccupied.

β) The king may not pass any point which is under threat by the opponent.

γ) Castling is not allowed if the king is in check.

δ) The pieces involved in castling must not have moved in so far.

The conditiona α, β, γ are functions of the field occupation. For the condition δ the knowledge of the course of the game so far is necessary. In order to avoid inclusion of the entire course of the game in the calculation every time, four Yes – No – Values are evaluated, which indicate whether white or black is allowed to execute king's side or queen' s side castling. These four Yes – No – Values are components of the game situation.

To c) This information results from the previous move. It is sufficient to specify the point to which the piece was moved. This information (AΔ2) must also be a component of the game situation.

Correspondingly the following structure can be assigned to the entire " game situation " :

$$A\Delta 9 = (A\Delta 5,\ So,\ S1.4,\ A\Delta 2)$$

The components have the following meaning :

Ko	($A\Delta 9$) =	$A\Delta 5$,	field occupation
K1	($A\Delta 9$) =	So,	– " white has the move "
			+ " black has the move "
K2	($A\Delta 9$) =	S1.4	Castling information

In detail

K2.0	=	So	" white may execute queen's side castling "
K2.1	=	So	" " " king's " "
K2.2	=	So	" black " " queen's " "
K2.3	=	So	" " " king's " "

K3 ($A\Delta 9$) = $A\Delta 2$ point to which the last piece moved. At the start and after a castling this is zero.

The start situation is specified by the constant $C\Delta 9$.

$$C\Delta 9 = (C\Delta 5, –, ++++, 0)$$

Here again it is advantageous to introduce the data structure AΔ10 in which the component AΔ5 is replaced by $A\Delta 6$ (field occupation with specification of the points).

$$A\Delta 10 = (A\Delta 6, So, S1.4, A\Delta 2)$$

The start situation is specified by the constant

$$C\Delta 10 = (\Delta C 6, –, ++++, 0)$$

For the data $A\Delta 9$ and AΔ 10 various restrictions apply, since not every variation is meaningful see PΔ.192.

A move is specified by the following data :

a) Normal moves :

α) Point from which the piece is moved

β) Point to which the piece is moved

b) Castling : Queen's side or King's side

c) En–passant capture: information as to whether capture is executed

To a) It is usual to mention the moving piece and to mark the case of capture (x). Sometime the captured piece is reported too.
The case of announced check is reported specially.
These supplementary data are redundant since they result form the game situation.

To. b) Castling is normally marked by special symbols. In principle it is sufficient to mention the points between which the king is moving. Since only in case of castling is the king allowed to move over two points, this fact is sufficient specification for castling.

To c) If the captured piece is reported in any case, then a special specification is superfluous.

Two types of move data will be used :

The concentrated move data : AΔ.11

This contains the necessary data only.

$$A\Delta 1 = (A\Delta 2, A\Delta 2, So)$$

Meaning of the components:

Ko (A Δ 11) = A Δ 2 , point from which the move starts

K1 (A Δ 11) = A Δ 2 , point to which the move goes

K2 (A Δ 11) = So, " Capture is executed " .

K2 is only relevant in the case of en–passant capture. The specification of castling is established by the following data :

a) queen's side white castling	ooo,Loo/ooo,oLo
b) king's " " "	ooo,Loo/ooo,LLo
c) queen's " black "	LLL,Loo/LLL,oLo
d) kings's " " "	LLL,Loo/LLL,LLo

As well as the move data AΔ.11 it is possible to introduce extended move data AΔ.12 , but this will not be discussed.

2) Operations with AΔ9 and AΔ10

PΔ.192 " The game situation is meaningful "

The following conditions must be satisfied:

1) The positions of the pawns must satisfy the conditions of PΔ.64 ($\frac{Z}{1}$).

2) The number of the various types of pieces must satisfy the conditions of PΔ.101 ($\frac{Z}{2}$).

3) The two kings must not be in check simultaneously ($\frac{Z}{3}$).

4) A king must not be in check more than twice .

5) If a king is in check then its color must be equal to the color which has the move ($\frac{Z}{5}$).

6) If a king or a rook is not in its starting position then the corresponding components of K2 (AΔ10) must be negative ($\frac{Z}{6}$).

7) If K3 (AΔ10) is not equal to zero, then the corresponding point of the field must be occupied by a piece of the color which does not have the move ($\frac{Z}{7}$).

With PΔ.192 not all impossible cases are excluded. If for example at the start knight and bishop are exchanged, then this fact is not compatible with the rules, but it is compatible with the above program.
The condition for two bishops either white or black is omitted.

PΔ.192

```
   | ( x ) ( x ∈ V → RΔ64 ( x )) ⇒ z | RΔ101[ RΔ96 ( Sp1 ( V ))] ⇒ z
V  |              o            1     |                 o        2
K  |              o                   |                 o
A  |              Δ6      Δ4     o    |              Δ7  Δ5  Δ6   o

   | x'( x ∈ V ∧ x = ——+— ) ⇒ z | x' ( x ∈V ∧ x = ——++ ) ⇒ z
V  |           o            10  |        o   1          11
K  |           o   1            |        o
A  | Δ4        Δ6  Δ3       Δ4  | Δ4     Δ6  Δ3         Δ4

   | N[ x̂ ( x ≠0  x ∧ RΔ129 ( V , x , z ))] ⇒ z
V  |                  o     10              12
K  |            1.3  o  o  o
A  |        Δ4  o     Δ6 Δ2 Δ2              1.n

   | N[ x ( x ≠0 ∧ x̄ ∧ RΔ129 (V , x , z ))] ⇒ z
V  |                  o     11              13
K  |            1.3  o  o  o
A  |        Δ4  o     Δ6 Δ2 Δ2              1.n
```

	(z	= 0 ∨ z	= 0) ⇒ z	(z	⩽ Lo ∧ z	≤ Lo) ⇒ z
V	12	13	3	12	13	4
K						
A	1.n	1.n	o	1.n	1.n	o

	(z	≠ 0 → $\overline{V}$	) ∧ (z	≠ 0 → V	) ⇒ z
V	12	o	13	o	5
K		1		1	
A	1.n	o	1.n	o	o

	(V	= +–+–) ⇒ z	(V	= ––+–) ⇒ z
V	o	14.0	o	14.1
K	o.(ooo,ooo).1		o.(ooo,Loo).1	
A	Δ3	o	Δ3	o

	(V	= +–+–) ⇒ z	(V	= +–++) ⇒ z
V	o	14.2	o	14.3
K	o.(ooo,LLL).1		o.(LLL,ooo).1	
A	Δ3	o	Δ3	o

	(V	= ––++) ⇒ z	(V	= +–++) ⇒ z
V	o	14.4	o	14.5
K	o.(LLL,Loo).1		o.(LLL,LLL).1	
A	Δ3	o	Δ3	o

	(V	→ z	∧ z	) ∧ (V	→ z	∧ z	) ∧ (V	→ z	∧ z	) ∧ (V	→ z	∧ z	) ⇒ z
V	o	14.o	14.1	o	14.1	14.2	o	14.3	14.4	o	14.4	14.5	6
K	2.o			2.1			2.2			2.3			
A	o	o	o	o	o	o	o	o	o	o	o	o	o

	V ≠ 0 →	[(V ⌐	V.1.3) ~ V]	⇒ z	z ∧	z ∧	z ∧	z ∧	z ∧	z ∧	z	⇒ R
V	o	o	o	7	1	2	3	4	5	6	7	o
K	3	o ⌐	1									
A	Δ2	o	o	o	o	o	o	o	o	o	o	o

<u>To PΔ.192</u> :

Meaning of the intermediate values :

Z_1 to Z_7	Conditions according to page 233
Z_{10}	point occupied data of the white king
Z_{11}	" " " " " black "
Z_{12}	number of pieces attacking the white king
Z_{13}	" " " " " black "
$Z_{14.0}$ to $Z_{14.5}$	Information as to whether rooks and kings are in start position.

PΔ.193 Possibility of castling for white

```
R ( V   ) ⇒ ( R, R )        R_o = queen's side castling possible
    o       o 1
    Δ10     o o             R_1 = king's  "   "      "
```

```
  | ( V     = +–+– ) ∧ ( V     = 0 ) ∧ ( V     = 0 )
V |   o                  o              o
K |   o.(o,o).1          o.(o,1).1      o.(o,2).1
A |   Δ3                 Δ3             Δ3

  | ∧ (V     = 0 ) ∧ R̄Δ133 (V , ( o,2 )) ∧ R̄Δ133 (V , ( o,3 )) ⇒ z
V |   o                  o              o                        o
K |   o.(o,3).1          o              o
A |   Δ3                 Δ6             Δ6                       o

  | (V     = ––+– ) ∧ R̄Δ133 (V , ( o,4 )) ⇒ z
V |  o                  o                  1
K |  o.(o,4)            o
A |  Δ3                 Δ6                 o

  | (V     = 0 ) ∧ ( V     = 0 ) ∧ ( V     = +–+– )
V |  o                  o              o
K |  o.(o,5).1          o.(o,6).1      o.(o,7).1
A |  Δ3                 Δ3             Δ3

  | ∧ R̄Δ133 (V , ( o,5 )) ∧ R̄Δ133 (V , ( o,6 )) ⇒ z
V |          o                  o                  2
K |          o                  o
A |          Δ6                 Δ6                 o

  | z ∧ z ∧ V ⇒ R  | z ∧ z ∧ V ⇒R
V | o   1   o   o  | 1   2   o   1
K |         2.o   |         2.1
A | o   o   o   o  | o   o   o   o
```

PΔ.194 Corresponds to PΔ.193, however for black

PΔ.200 Development of the new game situation form the previous game situation and the move notification:
Supposition: meaningful game situation and move allowed.

```
  | R ( V   , V   ) ⇒ R Δ 200
V |     o     1       o
A |     Δ10    Δ11     Δ10
```

The program has several parts:

a) Normal moves, according to PΔ.136

b) En–passant capture:
The criterion for this is :
The moving piece is a pawn. This moves two steps straight ahead and is captured.
As well as the normal variation of the game situation by the moving piece, the captured piece must disappear from the field.

c) Castling
The criterion for this is:
The moving piece is a king and this moves sideways in its rank two points. The color of the king and the direction of its move specify the type of castling. As well as the normal variation of the game situation by the moving king, its variation by the participating rook has to be observed.

2) Notification, whether white or black has the move ($\overset{V}{\underset{1}{o}}$)

3) Development of information as to whether castling is allowed:
The data $\overset{V}{\underset{2}{o}}$ is tranferred, if the moving piece is neither a rook nor a king. Otherwise the corresponding variation has to be performed.

4) The third component of $\overset{R}{o}$ results form the point to which the move was executed.

PΔ.200

```
   | V ⇒ z  | z ┌V ⇒ z  | z ⇒ z ┌V.1  | 0 ⇒ z ┌V.1
V  | o   o  | o │ 1   1  | 1   o │ 1      o │ 1
K  | o      |   └ o      | 1     └ 1        └ o
A  | Δ6  Δ6 | Δ3  Δ2  Δ4 | Δ3  Δ3  Δ2     Δ3  Δ2
```
normal moves

```
   | ┌ z =+––o ∧ ( |V  –  V | = Lo )∧V ┐ ⇒  o ⇒ z ┌ V.1
V  | │ 1            1     1        1   │       o │ o
K  | │ 1            1.1   o.1      2   │         └ 3
A  | └ Δ3           Δ1    Δ1       o   ┘       Δ3
```
En–passant capture

```
   | z = ––+– ∧ ( V  –V  = Lo ) ⇒ ┌ 0 ⇒ z  | +–+–⇒ z ┐
V  | 1           1   1          │     o   |     o  │
K  | 1           o.o 1.o        │     (o,o).1 |  (3,o).1
A  | Δ3          Δ1  Δ1         └     Δ3    |  Δ3    ┘
```
Wh.Q.side Castling

```
   | z = ––+– ∧ ( V  –V  = Lo ) ⇒ ┌ 0 ⇒ z  | +–+–⇒ z ┐
V  | 1           1   1          │     o   |     o  │
K  | 1           1.o o.o        │     (7,o).1 |  (5,o).1
A  | Δ3          Δ1  Δ1         └     Δ3    |  Δ3    ┘
```
Wh.Kg. side Castling

	z = ––++ ∧ (	V	–V	= Lo) →	0 ⇒ z		+–++ ⇒ z		Bl.Q.side
V	1		1	1		o		o	Castling
K	1		o.o	1.o		(o,7).1		(3,7).1	
A	Δ3		Δ1	Δ1		Δ3		Δ3	

	z = ––++ ∧ (	V	–V	= Lo) →	0 ⇒ z		+–++ ⇒ z		Bl.Kg. side
V	1		1	1		o		o	Castling
K	1		1.o	o.o		(7,7).1		(5,7).1	
A	Δ3		Δ1	Δ1		Δ3		Δ3	

	V ∧	$\overline{z = ((o;o), +–+–)}$ ∧	z = ––+– ⇒	R	Wh.Q. side
V	o	1	1	o	Castling
K	2.o		1	2.o	
A	o	Δ4	Δ3	o	

	V ∧	$\overline{z = ((7,o), +–+–)}$ ∧	z = ––+– ⇒	R	Wh.Kg.side
V	o	1	1	o	Castling
K	2.1		1	2.1	
A	o	Δ4	Δ3	o	

	V ∧	$\overline{z = ((o,7), +–++)}$ ∧	z = ––++ ⇒	R	Bl.Q. side
V	o	1	1	o	Castling
K	2.2		1	2.2	
A	o	Δ4	Δ3	o	

	V ∧	$\overline{z = ((7,7), +–++)}$ ∧	z = ––++ ⇒	R	Bl.Kg. side
V	o	1	1	o	Castling
K	2.3		1	2.3	
A	o	Δ4	Δ3	o	

	z	⇒ R	$\overline{V}$	⇒ R	V	⇒ R	Results
V	o	o	o	o	1	o	
K		o	1	1	1	3	
A	Δ6	Δ6	o	o	Δ2	Δ2	

Alternative representation of PΔ.200
Extraction of the " move analyses" as a special program.

PΔ.201 Move analyses

	R (V , V) ⇒ (	R	,	R	,	R	,	R	, R)
V	o	1		o	1	2	3	4	
A	Δ6	Δ11		Δ3	o	o	o	1.2	

R_o = Moving piece

R_1 = " Normal move"

R_2 = "En–passant capture"

R_3 = " Castling "

R_4 = Type of Castling

R_4	
––	Wh.Q.s.C.
–+	Wh.Kg.s.C.
+–	Bl.Q.s.C.
++	Bl.Kg.s.C.

	V	V ⇒ z		z ⇒ R		V	– V ⇒ z		V	– V ⇒ z		
V	o	1	o	o	o	1	1	1	1	1	2	
K		o		1		1.o	o.o		1.1	o.1		
A	Δ6	Δ11	Δ4	Δ3	Δ3	Δ1	Δ1	8.2	Δ1	Δ1	8.2	

	z = +––o ∧ (z = 2) ∧ V ⇒ R				
V	o	2	1	2	
K	1		2		
A	Δ3	8.2	o	o	

	[z = ––+o ∧ |z| = 2]	⇒	[+ ⇒ R	(z , z) ⇒ R]					
V	o	1		3		o	1	4	
K	1					1.3	o		
A	Δ3	8.2		o		o	o	o	

	$\overline{R}$ ∧ $\overline{R}$ ⇒ R			
V	2	3	1	
K				
A	o	o	o	

PΔ.202 Development of the new game situation form the previous situation with the aid of subprogram PΔ.201.
It is true : PΔ.200 ~ PΔ.202

	R (V , V) ⇒ R				
V		o	1	o	
A		Δ10	Δ11	Δ10	

The variation of the various cases of castling is represented by a combination of the meaningful values of the points involved in two program constants Cp_o and Cp_1

PΔ.202

	V ⇒ z		z	V ⇒ z		z	⇒ z	V.1	0 ⇒ z	V.1
V	o	o	o	1	1	1	o	1	o	1
K	o			o		1		1		o
A	Δ6	Δ6	Δ3	Δ2	Δ4	Δ 3	Δ 3	Δ 2	Δ3	Δ 2

	R Δ 201 (z , V) ⇒ (□, □, z, z, z)				
V	o	1		2 3 4	
K					
A	Δ6	Δ11		o o 1.2	

	z → [0 ⇒ z V.1]		[o,o]	= Cp	[3,o]	= Cp
V	2	o o	o,7	o	3,7	1
K		3	7,o		5,o	
A	o	Δ3	7,7	4XΔ 2	5,7	4XΔ 2

	z → [0 ⇒ z (Cp z).1]			(+,–,+, V) ⇒ z (Cp z).1			
V	3	o	o 4		o	o 1	4
K					1		
A		Δ3	Δ2 1.2		o	Δ3 Δ2	1.2

	i	W [V ∧ $\overline{z = (Cp, (+,–,+,V))}$ ∧ $\overline{z = (–,–,+V) ⇒ R}$]						
V		o	1	o	o	1	o	o
K		2.i		i	1	1	1	2.i
A	1.2	o	Δ4	Δ2	o	Δ3	o	o

	z ⇒ R		$\overline{V}$ ⇒ R		V ⇒ R	
V	o	o	o	o	1	o
K		o	1	1	1	3
A	Δ6	Δ6	o	o	Δ2	Δ2

PΔ.203 Superversion of the game
" The move is allowed"
(compatible with the rules)
Data for check mate and draw

	R (V , V) ⇒ (R , R , R , R)					
V	o	1	o	1	2	3
A	Δ10	Δ 11	o	Δ1	o	o

V_o = given situation

V_1 = move

R_0 = " the move is allowed "

R_1 = new situation

R_2 = checkmate
R_3 = draw
} to the color, which did not have the move

The program is composed of the following parts:

1) Move analyses according to PΔ.201 (Z_0 to Z_4)
2) Investigation as to whether the color of the moving piece has the move
3) Investigation as to whether the move is allowed
   a) normal move (Z_{11})
   b) en-passant (Z_{11}, Z_{12})
   c) Castling
4) Developemtn of the new game situation according to P.202
5) Investigation for checkmate or draw.

The condition , that sequences of moves may only be repeated twice at the most, is not contained in PΔ.203. This can only be supervised by registration of all moves (the whole course of the game).

P△.203

```
|  R△201 (V ,  V ) ⇒ (  z  ,  z, z, z, z  )
V|       o    1        o    1 2 3 4
K|       o
A|       △6   △11      △3   o o o 1.2
```

1)

```
|  z ~  V ⇒ z .
V|  o    o    10 .
K|  3
A|  o    o    o
```

2)

```
|  z ∨ z ⇢[R△128 (Sp1 (V ),   V ,  V ) ⇒  z ]
V|  1   2  [                  o      1    1     11 ]
K|  .      [                  o      o    1        ]
A|  o   o  [       △5    △6      △2   △2    o    ]
```

3a,b)

```
|  z ⇢[V   [o.V    =  z ]
V|  2[o    [ o     6  ]
K|    [     [  3      ]
A|    [△3   [△2    △3 ]]
```

3 b)

```
|  z ⇢[( V̄ → R△14 ( V  , V )) ∧ ( V → R△15 ( V  , V )) ∧ z ≠ 0 ∧ z ~  V ⇒ z ]
V|  2[  o          1   o    o          1   o     6        6    o    12    ]
K|    [  1          o   3    1          o   3                3    1       ]
A|  o [  o          △2  △2   o          △2  △2    △3       o    o    o   ]
```

3 b)

```
|  z ⇢[(V̄ ∧ z →  R△193 (V  )) ∧ ( V̄ ∧ z →  R△193 (V  ))]⇒ z
V|  3[  o   4   o     o          o   4   1     o        ]  13
K|    [  1   1                    1   1                 ]
A|  o [  o   o   o     △10        o   o   o     △10      ]  o
```

3 c)

```
|     ∧(V ∧ z̄ →  R△194 (V  )) ∧ (V ∧ z →  R△194 (V  ))
V|     [  o   4   o     o          o   4   1     o        ]
K|     [  1   1                    1   1                 ]
A|     [  o   o   o     △10        o   o   o     △10      ]
```

```
|R△202 (V ,  V ) ⇒  z  |  ( V̄ ∧ R△134 (z )) ∨ (V ∧ R△135 (z )) ⇒  z
V|      o    1     5   |   o          5     o          5      14
K|                     |   1          o     1          o
A|      △10  △11   △10 |   o          △6    o          △6     o
```

4), 5)

```
|( V̄∧R△196 ( z)) ∨ ( V∧R△195 ( z )) ⇒ R
V|  o          5      o          5     2
K|  1                 1          o
A|  o          △10    o          △6    o
```

5)

	(V ∧ R△160 (z)) ∨ (V ∧ R△161 (z)) ⇒ R	
V	o 1 5 o 1 5 3	5)
K	1 o 1 o	
A	o o △6 o △6 o	

	z ∧ (z ∧ z) ∨ z ∧ $\bar{z}$ ⇒ R	z ⇒ R
V	10 11 12 13 17 o	5 1
K		
A	o o o o o o	△10 △10

To P△203

Meaning of the data : A

V_o = given situation

V_1 = move

Z_o = moving piece

Z_1 = " normal move "

Z_2 = " en – passant capture "

Z_3 = Castling

Z_4 = type of Castling

$\bar{Z}$ = q.s.c.	Z = k.s.c.
4	4
1	1

Z_5 = new situation

Z_6 = piece moved in the previous move (through en – passant captured)

Z_{10} = " the color of the moving piece has the move "

Z_{11} = " normal move is allowed and or the condition for en – passant is satisfied

Z_{12} = " condition for en – passant capture is satisfied "

Z_{13} = " condition for castling is satisfied. "

Z_{14} = " the king of the color which just moved is in check in the new situation " .

R_o = " the move is allowed "

R_1 = " new situation "

R_2 = " the opposing color is mated by the move "

R_3 = " the opposing color is drawn by the move " .

The case, in which only the two kings survice (Remis) is not covered by the program.

If only one knight or one bishop of one color, remains with the kings, this also results in Remis.

The promotion of a pawn is missing in the program. For this purpose, the move data must be extended.

Types of Data and Constants

So = Yes-No-Values

S1.n = Series of Yes-No-Values

A△1 = S1.3 Coordinate
T△1

A△2 = 2XA△1 = Point

A△3 = S1.4 = Occupation Notation
B△3

A△4 = (A△2, A△3) = Point Occupation Notation

A△5 = 64XA△3 = Field Occupation
C△5. Start Occupation

A△6 = 64XA△4 = Field Occupation with Evaluation of the Points
B△6 C△6. Start Occupation

A△7 = 12XS1.4 = List of the Numbers of Pieces
C△7. Situation at the Start

A△8 = Qz (C△o.2, A△7) = List of the Numbers of Pieces with Designation of the Pieces; C△8. Situation of the Start

A△9 = (A△5, So, S1.4, A△2) = Game Situation
C△9 Start Situation

A△10 = (A△6, So, S1.4, A△2) = Game Situation with Evaluation of the Points
C△10 Start Situation

A△11 = (A△2, A△2, So) = Move Data

A△12 = Extended Move Data

Other Constants

C△0.1 Valuation Table of the Pieces

C△0.2 List of Pieces