The “Plankalkül” of Konrad Zuse: A Forerunner of Today’s Programming Languages

F.L. Bauer and H. Wössner
Mathematisches Institut der
Technischen Universität München

> The very first attempt to devise an algorithmic language . . . but the proposal never attained the consideration it deserved.
>
> Heinz Rutishauser (1967)

Plankalkül was an attempt by Konrad Zuse in the 1940’s to devise a notational and conceptual system for writing what today is termed a program. Although this early approach to a programming language did not lead to practical use, the plan is described here because it contains features that are standard in today’s programming languages. The investigation is of historical interest; also, it may provide insights that would lead to advancements in the state of the art. Using modern programming terminology, the Plankalkül is presented to the extent it has been possible to reconstruct it from published literature.

Key Words and Phrases: higher programming languages, programming, theory of programming, history of programming

CR Categories: 1.2, 4.22, 5.29

Authors’ address: Mathematisches Institut der Technischen Universität München, 8000München 2,Postfach 202420, Germany.

Preface

In May 1945, Konrad Zuse, Berlin-born inventor and constructor, who had arrived with his relay computer Z4 at the little village of Hinterstein in the Allgäu Alps, found himself immobilized by the postwar situation and prevented from pursuing his business. He thus found time to resume his 1943 studies[1] on how to formulate data processing problems. Zuse understood and used the German word *Rechnen*, to compute, in the most general sense when he wrote, “*Rechnen heisst: Aus gegebenen Angaben nach einer Vorschrift neue Angaben bilden*”[2].

He used *Angaben* for data and *Vorschrift* for algorithm. Not having at his disposition the word *Programm*, he called a program *Rechenplan*. The notational and conceptual system of expressing a *Rechenplan* he called *Plankalkül*.

The *Plankalkül*, as a remarkable first beginning on the way to higher programming languages, deserves a place in the history of informatics, Although this early attempt to develop a programming language did not lead to practical use, it is nevertheless surprising to what extent the *Plankalkül* already contains standard features of today’s programming languages.

We are led to an investigation of Zuse’s *Plankalkül* not only because of historical interest, but also because the necessary critical reflection on the state of the art with its possible gaps and weaknesses may gain from such a study. In particular, the widespread ignorance about the *Plankalkül* should be diminished.

Using as a basis modern terminology in programming,[3] we will describe the Plankalkül as far as it can be reconstructed from the published literature.

---

[1] “Ansätze einer Theorie des allgemeinen Rechnens”, a planned Ph.D. dissertation. See [Z70, p. 112].

[2] See [Z49].

[3] In terminology and notation, we follow ALGOL 68. Whatever position one may have with respect to ALGOL 68, the difference from other reputable terminologies and notations, such as the one Hoare, Wirth and Dijkstra prefer, is not so great that it would hinder communication.

1. Data Structure

The only *primitive objects* in the *Plankalkül* are of the mode **bool** (or bit), which is denoted by S0[4]; they are called *Ja-Nein-Werte*. *Composite objects* are built up recursively, in particular *arrays* of arbitrary dimensions and *records*. For example, the array modes

$[0 : n - 1]$ **bool** and $[0 : m - 1, 0 : n - 1]$ **bool**

are denoted by

$n \times S0$ and $m \times n \times S0$, respectively.

If a *variable indication*[5] (*variables Strukturzeichen*) σ or a *constant indication* $S2$ is used to denote the first of these two modes, then the second can be denoted by

$m \times \sigma$ or $m \times S2$, respectively.

There is also the possibility of using the abbreviated notation

$S1 \cdot n$ or $S1 \cdot 8$

instead of

$n \times S0$ or $8 \times S0$.

In this case we have a new mode **bits** of word length n or 8, respectively; the array, however, can still be subscripted.

A record of, say, two components, which are denoted by some variable or constant indications σ, τ or A2, A3, is specified by

(σ, τ) or (A2, A3).

Here, too, subscripts will be used for the selection of components; they always start with zero.

Zuse says *Strukturen* for structured values and their corresponding modes; he says *Art* for the conglomerate consisting of a *Struktur* together with its pragmatic meaning (*Typ*) and a possible restriction (*Beschränkung*), which says which of the elements of a certain structure are meaningful, For example, objects of the structure

$S1 \cdot 4$ (tetrades)

may have the pragmatic meaning "decimal digit" and the restriction to the first 10 of the 16 lexicographic possibilities.

$$A3 = \begin{pmatrix} S1 \cdot 4 \\ B3 \end{pmatrix}$$

expresses that $S1 \cdot 4$ is subjected to the restriction $B3$.[6] Zuse calls objects *Angaben*, which pretty closely correspond to "data".

Figure 1 shows an illustrative section from [Z59].

2. Standard Denotations

Standard denotations for Boolean objects (S0) are

L and 0

for bit sequences (for example $S1 \cdot 4$)[7]

LL00, L0LL.

For integers and numerical-real objects, instead of bit sequences, conventional figures can also be used[8].

For the standard denotation of more general, composite objects, a denotation is used which is now conventional for input and output: The standard denotations of the components of composite objects are listed in the specified order, such that the additional mode indication for the object allows one to form the decomposition uniquely. For clearness only, a special separation mark (semicolons instead of commas) is used for the separation of composite objects.

3. Free Choice of Denotation

For all objects, freely chosen identifiers (*Bezeichnungen*) may be introduced; for example, a standard denotation can be associated (*zugeordnet*) with an identifier (see Section 6) such that both possess the same object as their value (*Wert*).

In a *Rechenplan* P, i.e. in a program or a subroutine (see Section 9), an identifier is a letter followed by a number. The letter is *V*, *Z*, *R*, or *C*, depending on whether the object in question is used as an input parameter (*Variable*), intermediate value (*Zwischenwert*), result parameter (*Resultatwert*), or as a constant in P. The distinguishing number (*Nummer*) is attached to the letter in the line below. The letter classifies the objects.

Examples:

$$\begin{array}{llll} V, & Z, & Z, & R \\ 0 & 0 & 1 & 0 \end{array}$$

Finally, programs and subroutines have their own identifiers like

P12, P3-7

the number following the letter P being a program index (*Programm-Index*), in the form of a component-subscript (see Section 4). The second example denotes "the program 7 of the program group 3". Thus, Zuse has arrays of programs and a corresponding block structure. He derives from this a system to denote the results of subroutines in external use; for example, the result $\underset{0}{R}$ of a subroutine P17 is external to P17 characterized by the program index 17, i.e. by

$$\begin{matrix} R17 \\ 0 \end{matrix}$$

which also involves a call of P17 (see Section 8).

---

[4] In [Z49] a small *o* is used.

[5] [Z49, p. 447]; see also Section 9

[6] For a chess example such a restriction is defined in [Z59, p. 72] by: "*A*3 is restricted to 13 possibilities: 12 binds of chessmen and 0 for unoccupied".

[7] [Z59, p. 70]. From a remark in [Z70, p. 157], one can infer that Zuse already during his Berlin period, that is before 1944, used L and 0, which he called *Sekundalziffern* (see also [Z70, p. 68] in his diary entry of June 20, 1937).

[8] It should be noted that Zuse already used floating point computation.

4. Subscripting

The selection of a component is achieved with the help of a component-subscript (*Komponenten-Index*), that is the denotation of a number (simple subscript) or a sequence of numbers (multiple subscript). The component-subscript is written immediately under the identifying number of the corresponding composite object.

Let, for example, $\begin{matrix} V \\ 0 \end{matrix}$ denote an array of the mode $l \times m \times S1 \cdot n$, then

$$\begin{array}{ll} V & \\ 0 & (0 \le i < l) \\ i & \end{array}$$

selects its *i*th component, a subarray of the structure $m \times S1 \cdot n$, while

$$\begin{array}{ll} V & \\ 0 & (0 \le j < m) \\ i \cdot j & \end{array}$$

selects the *j*th component of $\begin{matrix} V \\ 0 \\ i \end{matrix}$, a list of the structure $S1 \cdot n$, and finally

$$\begin{array}{ll} V & \\ 0 & (0 \le k < n) \\ i \cdot j \cdot k & \end{array}$$

selects the *k*th component of $\begin{matrix} V \\ 0 \\ i \cdot j \end{matrix}$, a single bit. In today's notation, this corresponds to *V*0[*i*], *V*0[*i*,*j*], *V*0[*i*,*j*,*k*].

Figure 1.

Ein Beispiel aus der Schachtheorie

Als Beispiel sei kurz auf die Schachtheorie eingegangen. Zunächst ist der Aufbau der auftretenden Angabenarten interessant.

*S*0	Ja-Nein-Wert
$S1 \cdot n$	*n*-stellige Folge von Ja-Nein-Werten

*A*1	$S1 \cdot 3$	= Koordinate
*A*2	$2 \times A1$	= Punkt (z. B.: L00, 00L entspricht Punkt e2 in üblicher Darstellung)
*A*3	$\begin{pmatrix} S1 \cdot 4 \\ B3 \end{pmatrix}$	= Besetzt-Angabe (z. B.: 00L0, Weißer König)
*A*4	(*A*2, *A*3)	= Punkt-besetzt-Angabe (z. B.: L00, 00L; 00L0 „Punkt e2 mit weißem König besetzt”)
*A*5	$64 \times A3$	= Feldbesetzung: *C*5 Anfangslage (Aufzählung der Besetzung der 64 Punkte in fester Reihenfolge)
*A*6	$64 \times A4$	= Feldbesetzung mit Punktangabe, *C*6 Anfangslage
*A*7	$12 \times S1 \cdot 4$	= Anzahl der Steine; *C*7 Anfangslage (Gibt an, wieviel Steine von jeder Sorte auf dem Feld sind, z. B. für Bewertungsrechnungen wichtig).
*A*9	(*A*5, *S*0, $S1 \cdot 4$, *A*2)	= Spielsituation; *C*9 Anfangssituation (Feldbesetzung [*A*5]; Angabe, ob Weiß oder Schwarz am Zuge [*S*0]; Angaben über Rochade-Möglichkeiten [4 Ja-Nein-Werte] Angabe der Punkte mit den Möglichkeiten, „en passant” zu schlagen).
*A*10	(*A*6, *S*0, $S1 \cdot 4$, *A*2)	= Spielsituation mit Punktangabe; *C*10 Anfangslage
*A*11	(*A*2, *A*2, *S*0)	= Zugangabe (zwei Punktangaben, gesetzt von ... nach ... Ein Ja-Nein-Wert „Es wird geschlagen”).

Pages 69,70,71 from “Über den Plankalkül” by Konrad Zuse, in Vol. 1, 1959, of *Elektronische Rechenanlagen*, Verlag R. Oldenbourg, Munich. Reprinted by permission of the publisher.

Figure 2.

Neben der Hauptzeile, welche die Formel im wesentlichen in der traditionellen Form enthält, wird eine zweite Zeile (V) für den Variablen-Index, eine dritte für den Komponenten-Index (K) und eine vierte für den Struktur-Index (S) eingeführt. Die letztere braucht, strenggenommen, nicht immer ausgefüllt zu werden, dient aber wesentlich zur Erleichterung des Verständnisses einer Formel. Die Zeilen werden durch Vorsetzen der zugeordneten Buchstaben (V, K, S) gekennzeichnet.

Beispiele:

$$\begin{array}{c|l} & V \\ V & 3 \\ K & \\ S & m \times 2 \times 1 \cdot n \end{array}$$

Die Variable V_3 ist eine Paarliste von *m* Paaren der Struktur 2 · 1 · *n* und soll als Ganzes in die Rechnung eingehen.

$$\begin{array}{c|l} & V \\ V & 3 \\ K & i \\ S & 2 \times 1 \cdot n \end{array}$$

Von der Paarliste V_3 soll das *i*.Paar genommen werden (Struktur 2 · 1 · *n*). (*I* kann dabei ein laufender Index sein.)

$$\begin{array}{c|l} & V \\ V & 3 \\ K & i \cdot 0 \\ S & 1 \cdot 0 \end{array}$$

Von dem i.Paar der Paarliste V_3 soll das Vorderglied (erstes Element des Paares) genommen werden (Struktur 1 · *n*).

$$\begin{array}{c|l} & V \\ V & 3 \\ K & i \cdot 0 \cdot 7 \\ S & 0 \end{array}$$

Von dem Vorderglied des i.Paares der Paarliste V_3 soll der Ja-Nein-Wert Nr. 7 genommen werden (Struktur *S*0 = Ja-Nein-Wert).

Beim Beispiel des Stabwerkes bedeutet für i = 4:

V 3	die gesamte Paarliste des Stabwerkes
V 3 4	die Kennzeichnung des Stabes 2-4 (4. Paar der gegebenen Liste)

5. Zuse's Two-Dimensional Notation

The form of denotation with a "main line" and "index lines" V and K for variable-number and component-subscript, respectively, is supplemented by an optional comment line S, in which the structure or mode of the value in question can be noted. To this end, the notation of Section 1 is used; Zuse calls these indications *Struktur-Indizes*.

Examples are given in Figure 2, an illustrative section from [Z59, p. 69].

The explicit marking of the lines by prefixed letters V, K, and S, allows one to omit empty K-lines. Furthermore, the prefix S in the mode denotation can be dropped. Thus,

$$S \,\big|\; S1 \cdot n \quad m \times S1 \cdot n \quad S0 \quad S2$$

can be shortened to

$$S \,\big|\; 1 \cdot n \quad m \times 1 \cdot n \quad 0 \quad 2 \qquad .$$

Moreover, Zuse allows the abbreviation of

$$S \,\big|\; A1 \quad A2 \quad S0 \quad A3$$

by

$$S \,\big|\; 1 \quad 2 \quad 0 \quad 3$$

(using *A*0 synonymously with *S*0)

Furthermore, variable component subscripts can be used [Z70, p. 123], for example by the help of an intermediate value in the form

$$\begin{array}{c|ll} & V & Z \\ V & 0 & 1 \\ K & & \\ S & m \times 1 \cdot n & 1 \cdot n \end{array}$$

with the meaning of *V*0[*Z*1] in today's notation. (Note that *Z*1 is of structure *S*1·*n*; that is, the integer corresponding to the bit sequence *Z*1 is used as subscript, and a component of structure *S*1·*n* is selected from *V*0.)

6. Assignment and Identity Declaration

The most important feature for the construction of programs is the assignment (*Rechenplangleichung*), expressed by means of the *Ergibtzeichen* ⇒.[9] For example, the assignment

$$\begin{array}{c|llll} & Z & +1 & \Rightarrow & Z \\ V & 1 & & & 1 \\ S & 1 \cdot n & 1 \cdot n & & 1 \cdot n \end{array}$$

means to augment the integer intermediate value Z_1 by 1, while

$$\begin{array}{c|llll} & (V, & V) & \Rightarrow & R \\ V & 0 & 1 & & 0 \\ S & \sigma & \sigma & & 2\sigma \end{array}$$

means the composition of the values V_0 and V_1 to a composite value, which is denoted by R_0.

If in a program more than one assignment to the same result or intermediate value variable occurs, then the (dynamically) first assignment is to be interpreted as an (initialized) identity declaration for a variable, while all others are ordinary assignments. This would give the genuine concept of a variable. On the other hand, the initialization of an input parameter in connection with a subroutine call,[10] as well as the initialization of constants, can be interpreted to be an ordinary identity declaration. However, these fine distinctions are reflected neither in the notation nor in the explanation of the semantics [Z59, p. 70]. Nevertheless, they have

---

[9] Originally, Zuse [Z49] introduced the $\succ\!\!=$, shaped equality sign. The arrow-like sign ⇒ is used in [Z59], after Rutishauser had helped to propagate it. In [R52], Rutishauser used in typescript the sign $\Rightarrow\!\!\!\!=$. At the Zürich **ALGOL** Conference 1958, the sign := was introduced under strong pressure from the American participants. The European group wished to use Zuse's sign.

[10] It cannot be excluded that Zuse considered the input parameters to be genuine variables whose values can be changed during the subroutine. This is indicated by an isolated occurrence of $\begin{array}{llll}(V, & V) & \Rightarrow & V \\ 5 & 6 & & 7\end{array}$ in [Z59],

strongly influenced Rutishauser's ideas, as seen from ALGOL 68.

The usual arithmetic and Boolean operations are provided for, and they allow one to form expressions (*Ausdrücke*) in connective formula notation.[11] Besides, comparison operations like =, ≠, ≤, with Boolean values as results, can be used. For arithmetic operations, objects of the mode **bits** (denoted by $S1 \cdot n$) are interpreted as numbers in direct (lexicographic) coding.

> *Der Operator x hat große Vorteile bei der systematischen Untersuchung einer sich evtl. In ihrem Umfang laufend ändernden Liste auf Glieder einer bestimmten Eigenschaft und Verarbeitung derselben.*

7. Further Operational Features

Apart from the possibility of selecting record and array components by (component) subscripts, certain operations from the predicate calculus are used to test components with respect to a specified property, with the result of selecting them or of obtaining a Boolean value. In this respect, the *Plankalkül* surpasses the potentialities in today's programming languages, including ALGOL 68.

Zuse uses both the "existence" and the "all" operator, and in particular the operator :

$$\begin{array}{lcccc} x(x & \in & V & \wedge & R(x)) \\ & & 0 & & \end{array}$$

means "The next component of V_0, for which the property R holds."

The property R, in the notation $R(x)$, is expressed by means of a computational rule which gives a Boolean value *(Ja-Nein-Wert)*, or of a result parameter of a suitable subroutine (see Section 8).

It is clear, that procedures can be defined in, say, ALGOL 68, which have the above effect. But it may be worthwhile to see whether Zuse's constructions could be introduced as original concepts in high level languages. See also [BG72].

8. Statements and Subroutine Calls

Statements are what Zuse calls *Planteile*. In particular, assignments are statements. Other statements, which we shall discuss, are conditional statements and repetitive statements. There is also a compound statement, formed with the help of parentheses. In order to separate statements, as well as the line marks (see Section 5), a vertical bar is used.

Conditional statements are formed with the help of the *Bedingt-Zeichen* $\dot{\rightarrow}$ (or $\underset{\cdot}{\rightarrow}$) in the following form

$$\mathsf{B} \dot{\rightarrow} \mathsf{A},$$

where the condition (*Bedingung*) B is an expression with Boolean value, and A an arbitrary statement. The elaboration of this conditional statement (*bedingter Planteil*) begins with B and ends with B or is continued with A, depending on whether B produces the value 0 = *nein* or L = *ja*. An alternative for A in the first case cannot be specified.

The following example of a repetitive statement, that is initiated by the letter *W*, shows an application of the -operation of the preceding section:

$$\begin{array}{c|l} & W \left[\begin{array}{cccc|cccc} \mu x(x \in & V \wedge x \neq & V) \Rightarrow & Z & (R \wedge & R17(& Z)) \Rightarrow & R \\ & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ \sigma & m\sigma & \sigma & \sigma & 0 & 0 & & 0 \end{array}\right] \\ V & \\ S & \end{array}$$

The elaboration of this *Wiederholungsplan* starts with the first assignment. The left-hand side formula of this assignment produces at each elaboration the next component $V_0[i]$ which is different from V_1, provided it exists. In this case, the following statement is elaborated and the process starts again. If, however, no component is found then Z_0 is unchanged *and the elaboration of the repetitive statement is finished*

In the second assignment of this example, where an initialization of R_0 is presupposed, $R17_1(Z_0)$ is the call of a subroutine P17 (see Section 9), which is specified to have one input parameter and a result parameter R_1 (see Section 3). The elaboration of this call means the identification of the actual parameter Z_0 with the formal input parameter, and following this, the elaboration of P17. The value of the call is the value which is obtained by R_1.

If it was initialized by L, R_0 obtains thus, when the repetitive statement is finished, the value of the conjunction of all $R17_1(x)$ where x is from the set of all elements of V_0 that are different from V_1.

9. Programs

Both programs and subroutines in the *Plankalkül* are expressed in the form of procedures (*Rechenpläne*); i.e. they are prefaced by a specification part (*Randauszug*), which specifies the parameters as being input parameters or result parameters together with their modes. The computational rule proper is then described in the body (*Anweisungsteil*), which consists of a sequence of statements. The end is marked by a symbol FIN.[12]

A call requires that the actual parameters have consistent mode. The subroutine P17 that was called in the preceding section may begin with the following specification part

$$\begin{array}{lc|cccc} P17 & & R(V) & \Rightarrow & (R, & R) \\ & V & 0 & & 0 & 1 \\ & S & & & & 0 \end{array}$$

where V_0 is an input and R_0, R_1 are result parameters. The body must contain assignments to R_0 and R_1. If it contains intermediate values, then they are not readable directly from outside of P17.

---

[11] [Z49, p. 447]; "The *Ergibt-Zeichen* ⋟ joins an expression which is to be calculated (left) with a result (right)." According to Zuse such expressions mean computational rules (*Rechenvorschriften*).

[12] The example in [Z59, p. 71] ends, however, with an expression ε instead of $\varepsilon \Rightarrow R_0$ FIN, where R_0 is the only result parameter.

> *Mein ‘Plankalkül’ war doch inzwischen längst veraltet.*
>
> K. Zuse (1970)

10. Algol 68 Translation of Some Plankalkül Programs

It should not be forgotten that Zuse did not only invent the *Plankalkül*, but that he used it to formulate some nontrivial programs of the nonnumerical kind (he called them *logistisch-kombinativ*) in order to demonstrate the potentialities of computing. The programs are by all means nontrivial for the year 1945 and more ambitious than the first task steps von Neumann did with his **Gedanken machine** (cf. [K70]). To illuminate this, we give in the following ALGOL 68 transcriptions of program examples from [Z49] and [Z59].

a. Syntax Checking for Boolean Expressions

A typical application of the *Plankalkül* [Z49, p. 446] contains a procedure for the syntax check of Boolean expressions. Zuse starts from the observation:

> Such expressions contain the following symbols: variable symbols, negation symbols, operation symbols, parentheses symbols, and space symbol that is needed for the separation of expressions. The symbols in question are coded in bit sequences.

Figure 3.

$$\begin{array}{c|ccc}
 & ① R(V) & \Rightarrow & R \\
V & 0 & & 0 \\
S & m\sigma & & 0
\end{array}$$

$$\begin{array}{c|ccc|ccc|c}
 & ② Az(V) & \Rightarrow & \& R & V & \Rightarrow & Z & ③ 0 \Rightarrow \varepsilon \\
V & 0 & & 0 & 0 & & 0 & \\
K & 0 & & & 0 & & & \\
S & \sigma & & 0 & \sigma & & \sigma & 1n
\end{array}$$

$$\begin{array}{c|c}
\begin{array}{c} \\ V \\ K \\ S \\ \\ \\ V \\ S \\ \\ \\ V \\ S \end{array} &
W \left[\begin{array}{l}
④ \mu x \begin{bmatrix} x & \in & V & \& x \neq V \\ & & 0 & 0 \\ & & & 0 \\ \sigma & & m\sigma & \sigma \end{bmatrix} \Rightarrow \begin{array}{c} Z \\ 1 \\ \\ \sigma \end{array} \left| \begin{array}{ccc} ⑤ Sq(Z,Z) & \Rightarrow & \& R \\ 0\;\;1 & & 0 \\ & & \\ \sigma\;\;\sigma & & 0 \end{array} \right. \\
\\
\begin{array}{c|c} ⑥ Kla(Z) \underset{\cdot}{\rightarrow} (\varepsilon + 1 \Rightarrow \varepsilon) & ⑦ Klz(Z) \underset{\cdot}{\rightarrow} (\varepsilon - 1 \Rightarrow \varepsilon) \\ 1 & 1 \\ \sigma & \sigma \end{array} \\
\\
\begin{array}{c|ccc} ⑧ \varepsilon \geq 0 \Rightarrow \& R & ⑨ Z & \Rightarrow & Z \\ 0 & 1 & & 0 \\ 0 & \sigma & & \sigma \end{array}
\end{array} \right]
\end{array}$$

$$\begin{array}{c|ccc|c}
 & ⑩ Sz(Z) & \Rightarrow & \& R & \varepsilon = 0 \Rightarrow \& R \\
V & 0 & & 0 & 0 \\
S & \sigma & & 0 & 0
\end{array}$$

In the procedure (Figure 3), σ denotes the structure of these 8-bit sequences, and $m\sigma$ with arbitrary $m \geq 1$ denotes the symbol sequences that are to be investigated. A call of the procedure with a (coded) symbol sequence x as its actual parameter means to test the predicate

$Sa(x)$: «x is a ‘meaningful expression’, i.e. a (syntactically correct) Boolean expression»

This predicate is introduced recursively by:

(i) A variable symbol is a meaningful expression.
(ii) A meaningful expression, prefixed by a negation symbol, yields a meaningful expression.
(iii) Two meaningful expressions, connected by an operation symbol, yield a meaningful expression.
(iv) A meaningful expression, put in parentheses, yields a meaningful expression.

To transform this definition into an algorithm, Zuse defines, now for symbols x, the auxiliary predicates:

$Va(x)$: «x is a variable symbol»
$Op(x)$: «x is operation symbol»
$Neg(x)$: «x is negation symbol»
$Kla(x)$: «x is opening parenthesis»
$Klz(x)$: «x is closing parenthesis»

and furthermore the predicates:

$Az(x)$: $Va(x) \vee Neg(x) \vee Kla(x)$
$Sz(x)$: $Va(x) \vee Klz(x)$
$Sq(x,y)$: $(Sz(x) \wedge \neg Az(y)) \vee (\neg Sz(x) \wedge Az(y))$

Figure 4.

„Der weiße König kann einen Zug machen, ohne dabei in Schach zu kommen.”

$$\begin{array}{r|ccc} \mathrm{P}\,148 & R(V) & \Rightarrow & R148 \\ V & 0 & & 0 \\ A & 5 & & 0 \end{array} \qquad (1)$$

$$\begin{array}{r|c} & x \\ V & \\ K & \\ A & 4 \end{array} \left[\begin{array}{ccccc} (x \in & V) & \wedge & (x & = \mathrm{L}\,0) \\ & 0 & & & \\ & & & 1 & \\ & 5 & & 3 & \end{array}\right] \begin{array}{cc} \Rightarrow & Z \\ & 0 \\ & \\ & 4 \end{array} \qquad (2)$$

$$\begin{array}{r|c} & (Ex) \\ V & \\ K & \\ A & 4 \\ & \\ V & \\ K & \\ A & \end{array} \left[\begin{array}{l} \begin{array}{ccccccccccc} (x \in & V) & \wedge & R17 & (Z, & x) & \wedge & (x & = 0) & \vee & x \\ & 0 & & & 0 & & & & & & \\ & & & & 0 & 0 & & 1 & & & 1.3 \\ 4 & 5 & & & 2 & 2 & & 3 & & & 0 \end{array} \\ \begin{array}{c} \wedge\,\overline{Ey} \\ \\ \\ 4 \end{array} \left[\begin{array}{ccccccccc} (y \in & V) & \wedge & y & \wedge & R128 & (V, & y, & x) \\ & 0 & & & & & 0 & & \\ & & & 1.3 & & & & 0 & 0 \\ & 5 & & 0 & & & 5 & 2 & 2 \end{array}\right] \end{array}\right] \begin{array}{c} (3) \\ \\ (4) \end{array}$$

Die hierbei benutzten Unterprogramme sind:

$$\begin{array}{r|ccc} & R17 & (V, & V) \\ V & & 0 & 1 \\ A & & 2 & 2 \end{array}$$ „Die Punkte V_0 und V_1 sind benachbart.”

$$\begin{array}{r|cccc} & R128 & (V, & V, & V) \\ V & & 0 & 1 & 2 \\ A & & 5 & 2 & 2 \end{array}$$ „Bei der gegebenen Feldbesetzung V_0 ist der Zug von Punkt V_1 nach Punkt V_2 erlaubt.”

Das Programm *R*128 ist verhältnismäßig kompliziert, da untersucht werden muß, welcher Stein auf Punkt V_1 steht, ferner ob der Punkt V_2 zu V_1 in einer solchen geometrischen Relation steht, daß der auf V_1 stehende Stein dorthin setzen kann, und schließlich muß untersucht werden, ob dazwischenliegende Punkte vorhanden sind und ob diese frei sind.

Erklärung der Formel P148 in Worten:

(1) ist der Randauszug, der besagt, daß über eine Feldbesetzung (*A*5) eine Aussage gemacht werden soll.

(2) Diejenige Punkt-Besetzt-Angabe (x), welche in der Liste der Spielbesetzung (V_0) enthalten ist, deren Komponente Nr. 1 = L0 ist (Zeichen für König in der Numerierung der Steintypen), ergibt den Zwischenwert Z_0.

(3) Es gibt in der Liste der Spielbesetzung (V_0) einen Punkt (*x*), der zu Z_0 (Punkt, auf dem der König steht) benachbart ist und der unbesetzt (= 0) oder mit einem schwarzen Stein besetzt ist ($x_{1\cdot 3}$) (das bedeutet Ja-Nein-Wert Nr. 3 der Besetzt-Angabe x_1; dieser charakterisiert schwarze Steine).

(4) Es gibt keinen weiteren Punkt, der mit einem schwarzen Stein besetzt ist, welcher nach Punkt *x* gesetzt werden kann.

He then postulates:

1. The first symbol *x* has to fulfill *Az*(*x*)
2. Two symbols *x*, *y* following each other have to fulfill *Sq*(*x*, *y*)
3. The last symbol *x* has to fulfill *Sz*(*x*).

Moreover, he uses the two parentheses counts:

4. The number of opening parentheses has to be equal to the number of closing parentheses.
5. For any segment of the symbol sequences, the number of opening parentheses must not be smaller than the number of closing parentheses.

The program (Figure 3) checks these conditions: ② serves for the special case of condition 1. ③ is an initialization for the repetitive statement which checks condition 2 and the count 5. Condition 3 for the final case is then checked in ⑩ and the count 4 after this. The program, by the way, contains mistakes: for example, a count corresponding to ⑥ is missing for the first symbol. More seriously, the condition $x \neq V0[0]$ in ④ should be read as $x = V0[i] \wedge i \neq 0$.

For a direct transliteration of Zuse's (corrected) procedure, we assume first that suitable Boolean procedures *Va*(*x*), *Op*(*x*), etc., are declared. Using these predicated, we obtain in ALGOL 68 (the encircled numbers refer to Figure 3):

```
①   proc Sa = ([0 : either] bits V0) bool : begin
②        bits Z0 := V0[0]; bool R := Az(Z0);
③        int eps := 0; if Kla(Z0) then eps := 1 fi;
④        for i to upb V0 while R do begin
         bits Z1 := V0[i];
⑤        R := R ∧ Sq(Z0,Z1);
⑥        if Kla(Z1) then eps +:= 1 fi;
⑦        if Klz(Z1) then eps –:= 1 fi;
⑧        R := R ∧ eps ≥ 0;
⑨        Z0 := Z1                          end;
⑩        R ∧ Sz(Z0) ∧ eps = 0                  end
```

(Of course, in ALGOL 68 there exist possibilities for a more efficient formulation.)

b. Checking a Move of the White King

Figure 4 shows one of the auxiliary procedures for a chess program formulated by Zuse in *Plankalkül* notations [Z59, p. 71]. The modes that are found in the program are seen from Figure 1 (note that *A*5 and *A*6 are to be interchanged). Zuse's procedure, directly transliterated into ALGOL 68 (the numbers 1 to 4 correspond to those in Figure 4) reads as follows:

```
mode  A1 = int          co coordinates 1, ··· , 8 instead of
                        0, ··· , 7 corresponding to [0:2] bool
                        co,
      A2 = [1:2] A1     co point co,
      A3 = int          co occupation by 1, ··· , 6 (9, ··· , 14)
                        for white (black) Q, K, R, B, S, P; in-
                        stead of 0 for unoccupied co,
      A4 = struct (A2 point, A3 occ) co occupation of the
                        point co,
      A5 = [1:64] A4    co occupation of the board co;
proc  R17 co adjacent co = (A2 V0,V1) bool ;
      abs (V0[1] - V1[1]) ≤ 1 ∧ abs (V0[2] - V1[2]) ≤ 1;
proc  R128 co move permissible co = (A5 V0, A2 V1, V2)
      bool ;
```

```
              «corresponding to the occupation occ of V0[i] that
              belongs to V1, where point of V0[i] = V1, the move
              from V1 to V2 is geometrically permissible» ∧
              «intermediate fields, if any, are free»;
1)     proc   R148 co move 2 (wK) permissible co =
              (A5 V0, ref A2 px) bool :
              co additional result parameter px for reference to target co
              begin bool c co if already checked, px refers to per-
                            missible target co := false;
2)                int i := 1; while occ of V0[i] ≠ 2 do i +:= 1; A4 Z0
                  = V0[i];
3)                for j to 64 while ¬c do
                  begin A4 x = V0[i]; px := point of x;
                      c := R17 (point of Z0, px) ∧ occ of x ≥ 8;
4)                    for k to 64 while c do
                      begin A4 y = V0[k];
                          if occ of y > 8 then c := ¬R128 (V0, point
                          of y, px) fi
                      end
                  end;
                  c
              end
```

> *Trotzdem glaube ich, daß der ... Plankalkül noch einmal praktische Bedeutung bekommen wird.*
>
> K. Zuse (1970)

Concluding Remarks

Altogether the *Plankalkül* turns out to be a highly developed programming language with structured objects that are built from a single primitive mode of objects—the two Boolean values (*Ja-Nein-Werte*) 0, L. Conceptually, this is certainly advantageous, but the existing plurality of modes in some predominant programming languages indicates the practical weakness of this approach. Apart from this, the *Plankalkül* shows many of the features of the programming languages of the sixties, sometimes obscured by an unorthodox notation, which disregarded some requirements of mechanical processing as well as some of the common notational habits. Some features—for example the structuring of objects—have only recently come into existing programming languages; others have yet to come. In particular, consideration of the features mentioned in Section 7 could be rewarding.

To assess the *Plankalkül* historically, one has to compare it with the flow diagram symbolism that originated at about the same time in the United States. Zuse's pioneering achievement of the forties should not be diminished by certain limitations, e.g. that the specification of modes is meant only to be an informal help for the correct use (in particular with respect to the parameters) of a procedure and not an intrinsic part of the program, or that the explicit formation of all modes from a single basic mode as well as the corresponding notation, are clumsy, or that questions of implementation have not been tackled.[13]

It is also interesting to indicate the features that are generally accepted today but which were not contained in the *Plankalkül*. Here we should first mention the reference concept—it is not even obvious whether ⇒ means an identity declaration or an assignment. Names or references as objects are also missing in ALGOL 60; in this respect the relation between *Plankalkül* and Rutishauser's influence[14] on ALGOL 60 is obvious. The essential restriction to numerical objects in ALGOL 60 was, as one knows today, not critical; the intention was to make the address calculation not accessible to the programmer, and this was motivated by the desire for error-free programming as well as by awareness of the frequent malfunction of machines in those years.[15] Thus, at that time, there was not enough justification to open, in ALGOL 60, the Pandora's box of manipulable names—i.e. addresses.[16] It was therefore left to Wirth to introduce this later into higher programming languages, and it can now be found in ALGOL 68 as well as in some "lower level languages".

References

Z43.
Zuse, K. Ansätze einer allgemeinen Theorie des Rechnens. 1943, unpublished.

Z45.
Zuse, K. "Plankalkül", Theorie der angewandten Logistik. 1945, unpublished.

Z49.
Zuse, K. Über den allgemeinen Plankalkül als Mittel zur Formulierung schematisch-kombinativer Aufgaben. *Archiv Math. I* (1948/49), 441-449. (Received Dec. 6, 1948.)

Z49a.
Zuse, K. Die mathematischen Voraussetzungen für die Entwicklung logistisch-kombinativer Rechenmaschinen. *ZAMM 29* (1949), 36-37. (Lecture, GAMM Conference Göttingen, Sept. 1948.)

R52.
Rutishauser, H. *Automatische Rechenplanfertigung bei programmgesteuerten Rechenmaschinen*. Mitteilungen aus dem Institut für angewandte Mathematik der ETH Zürich, No. 3. Birkhäuser, Basel, 1952.

Z59.
Zuse, K. Über den Plankalkül. *Elektron. Rechenanl. 1* (1959), 68-71.

Z68.
Zuse, K. Gesichtspunkte zur sprachlichen Formulierung in Vielfachzugriffssystemen unter Berücksichtigung des "Plankalküls". In: W. Händler (Ed.), *Teilnehmer-Rechensysteme*. Oldenbourg, Munich 1968.

Z70.
Zuse, K. *Der Computer mein Lebenswerk*. Verlag Moderne Industrie, Munich, 1970.

K70.
Knuth, D. E. Von Neumann's first computer program. *Computing Surveys 2* (1970), 247-260.

BG71.
Bauer, F. L., and Goos, G. *Informatik: Eine einführende Übersicht*. Springer, Berlin, 1971.

BG72.
Bauer, F. L., and Gnatz, R. Mengen in algorithmischen Sprachen oder: Arten und Prädikate. Mathematisches Institut der Technischen Universität München, Bericht No. 7202, 1972.

---

[13] K. Zuse in [Z70, p. 128]: "Der Plankalkül hätte noch 'compiler-gerecht' zugeschnitten werden müssen."

[14] F. L. Bauer. Heinz Rutishauser, Nachruf. *Computing 7* (1971), 129-130.

[15] "By this token one can calculate addresses. Symbolically, one can bring about this feature by a single wire. I had misgivings to do this step. [Z70, p. 99.]

[16] The question was, by the way, violently discussed at the Paris ALGOL Conference in January 1960. Proponent of "generated names" was Julian Green, who wanted ALGOL to have the possibility of describing its own translator.