The “Plankalkül” of Konrad Zuse: A Fore-runner of Today’s Programming Languages

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 the published literature.

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

CR Categories: 1.2, 4.22, 5.29

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, 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." 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

2. See [Z49].
3. In [Z49] a small o is used.
4. 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.

Heinz Rutishauser (1967)
such a study. In particular, the widespread ignorance about the Plankalkül should be diminished.

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

1. Data Structure

The only primitive objects in the Plankalkül are of the mode bool (or bit), which is denoted by $S_0$; 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 $n \times S_0$ and $m \times n \times S_0$, respectively.

If a variable indication (variables Strukturzeichen) $\sigma$ or a constant indication $S_2$ is used to denote the first of these two modes, then the second can be denoted by $m \times \sigma$ or $m \times S_2$, respectively.

There is also the possibility of using the abbreviated notation $S_1 \cdot n$ or $S_1 \cdot 8$ instead of $n \times S_0$ or $8 \times S_0$.

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 $\sigma, \tau$ or $A_2, A_3$, is specified by $(\sigma, \tau)$ or $(A_2, A_3)$.

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

$S_1 \cdot 4$ (tetrades)

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

$A_3 = \begin{pmatrix} S_1 \cdot 4 \\ B_3 \end{pmatrix}$

expresses that $S_1 \cdot 4$ is subjected to the restriction $B_3$.

Zuse calls objects Angaben, which pretty closely corresponds to "data".

Figure 1 shows an illustrative section from [Z59].

2. Standard Denotations

Standard denotations for Boolean objects ($S_0$) are $L$ and $0$

for bit sequences (for example $S_1 \cdot 4$)$^7$

$L L 00, L O L L$.

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, 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 $\phi$. The distinguishing number (Nummer) is attached to the letter in the line below. The letter classifies the objects.

Examples:

$V, Z, R$

$0 0 1 0$

Finally, programs and subroutines have their own identifiers like

$P_{12}, P_{3.7}$

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 \( R \) of a subroutine \( P17 \) is external to \( P17 \) characterized by the program index 17, i.e. by \( R_{17} \) which also involves a call of \( P17 \) (see Section 8).

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, \( V_0 \) denote an array of the mode \( I \times m \times S1 \cdot n \), then

\[
V_0 (0 \leq i < l)
\]

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

\[
V_0 (0 \leq j < m)
\]

selects the \( j \)th component of \( 0 \), a list of the structure \( i \)

\[
S1 \cdot n \), and finally
\[
V_0 (0 \leq k < n)
\]

selects the \( k \)th component of \( 0 \), a single bit. In today’s notation, this corresponds to \( V_0[i], V_0[i, j], V_0[i, j, k] \).

**Fig. 1.**

**Ein Beispiel aus der Schachtheorie**

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

\( S0 \) Ja-Nein-Wert
\( S1 \cdot n \) n-stellige Folge von Ja-Nein-Werten

\[
\begin{align*}
A1 & \quad S1 \cdot 3 = \text{Koordinate} \\
A2 & \quad 2 \times A1 = \text{Punkt} \\
A3 & \quad \begin{bmatrix} S1 \cdot 4 \\ B3 \end{bmatrix} = \text{Besetzt-Angabe} \\
A4 & \quad (A2, A3) = \text{Punkt-besetzt-Angabe} \\
A5 & \quad 64 \times A3 = \text{Feldbesetzung:} \\
A6 & \quad 64 \times A4 = \text{Feldbesetzung mit Punktangabe,} \\
A7 & \quad 12 \times S1 \cdot 4 = \text{AnzahlLISTE der Steine;} \\
A8 & \quad C5 \text{Anfangslage} \\
A9 & \quad (A5, S0, S1 \cdot 4, A2) = \text{Spielsituation;} \\
A10 & \quad C9 \text{Anfangssituation} \\
A11 & \quad (A2, A2, S0) = \text{Zugangabe}
\end{align*}
\]

Nach 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:**

<table>
<thead>
<tr>
<th>V</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>m × 2 × 1 · n</td>
</tr>
<tr>
<td>S</td>
<td>[\text{V_3}]</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>V</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>i</td>
</tr>
<tr>
<td>S</td>
<td>2 · 1 · n</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>V</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>i · 0</td>
</tr>
<tr>
<td>S</td>
<td>1 · 0</td>
</tr>
</tbody>
</table>

Von dem Vorderglied der Paarliste V_3 soll das Paar (1. Paar der gegebenen Liste) genommen werden (Struktur 1 · n).

<table>
<thead>
<tr>
<th>V</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>i · 0 · 7</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
</tr>
</tbody>
</table>

Von dem Vorderglied des i.Paares der Paarliste V_3 soll der Ja-Nein-Wert Nr. 7 genommen werden (Struktur S0 = Ja-Nein-Wert).

Beim Beispiel des Stabwerkes bedeutet für i = 4:

<table>
<thead>
<tr>
<th>V</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

die gesamte Paarliste des Stabwerkes

<table>
<thead>
<tr>
<th>V</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>m × 2 · n</td>
</tr>
<tr>
<td>S</td>
<td>[\text{V}_3]</td>
</tr>
</tbody>
</table>

die Kennzeichnung des Stabes 2—4

<table>
<thead>
<tr>
<th>V</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>m × 2 · n</td>
</tr>
<tr>
<td>S</td>
<td>[\text{V}_3]</td>
</tr>
</tbody>
</table>

Beim Beispiel des Stabwerkes bedeutet für i = 4:

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

Das Beispiel des Stabwerkes bedeutet für i = 4:

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

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 I 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,

\[S \mid S_1 \cdot n m \times S_1 \cdot n S_0 S_2 \sigma\]

can be shortened to

\[S \mid S_1 \cdot n m \times S_1 \cdot n 0 2 \sigma.\]

Moreover, Zuse allows the abbreviation of

\[S \mid A1 A2 S0 A3 \sigma\]

by

\[A \mid 1 2 0 3 \sigma\]

(using A0 synonymously with S0)

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

\[V \mid Z_0 \mid Z\]

\[V \mid 0 \mid 1\]

\[K \mid m \times 1 \cdot n\]

\[S \mid m \times 1 \cdot n 1 \cdot n\]

with the meaning of \(V_0[Z_1]\) in today's notation. (Note that \(Z_1\) is of structure \(S_1 \cdot n\); that is, the integer corresponding to the bit sequence \(Z_1\) is used as subscript, and a component of structure \(S_1 \cdot n\) 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** \(\Rightarrow\). For example, the assignment

\[Z + 1 \Rightarrow Z\]

\[V \mid 1 \mid 1\]

\[S \mid 1 \cdot n 1 \cdot n 1 \cdot n\]

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

\[(V, \ V) \Rightarrow R\]

\[V \mid 0 \mid 0\]

\[S \mid \sigma \sigma 2 \sigma\]
means the composition of the values $V_0$ and $V_1$ to a composite value, which is denoted by $R_0$.

The interpretation of the second example shows that the assignment comprises the semantic meaning of an (initialized) identity declaration for a variable: The identifier $R_0$ of the mode $2\sigma$ is used to denote the elaborated value on the left-hand side.

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, 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 strongly influenced Rutishauser’s ideas, as seen from ALGOL 58.

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

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 possibilities in today’s programming languages, including ALGOL 68.

Zuse uses both the “existence” and the “all” operator, and in particular the operator $\mu$:

$$\mu x (x \in V \land R(x))$$

0

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 Bedingungs-Zeichen $\rightarrow$ (or $\Rightarrow$) in the following form $
\begin{array}{c}
\text{Bedingung} \quad \Rightarrow \quad \text{Ausdruck}
\end{array}$

where the condition (Bedingung) $\text{Bedingung}$ is an expression with Boolean value, and $\alpha$ an arbitrary statement. The elaboration of this conditional statement (bedingter Planteil) begins with $\Rightarrow$ and ends with $\Rightarrow$ or is continued with $\alpha$, depending on whether $\Rightarrow$ produces the value $0 = \text{nein}$ or $L = ja$. An alternative for $\alpha$ 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 $\mu$-operation of the preceding section:

$$W \begin{array}{c}
\mu x (x \in V \land x \neq V) \Rightarrow\!Z
\end{array}$$

$$\begin{array}{|c|c|c|c|c|c|}
\hline
V & 0 & 1 & 0 & 1 & 0 \\
\hline
S & 0 & \sigma & m\sigma & \sigma & \sigma & 0 & 0 & 0 \\
\hline
\end{array}$$

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, $R_17(Z_0)$ is the call of a subroutine P17 (see Section 9), which is specified to have one input parameter and a result parameter $R_2$ (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 $R_3$.  

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 $\Rightarrow$ in [Z39].

The example in [Z59, p. 71] ends, however, with an expression $\varepsilon$ instead of $\Rightarrow$ $R_6$ FIN, where $R_6$ is the only result parameter.

Communications
July 1972
of the ACM
Volume 15
Number 7
If it was initialized by L, R₀ obtains thus, when the repetitive statement is finished, the value of the conjunction of all R₁₇₁(ᵧ) where 𝒧 is from the set of all elements of V₀ that are different from V₁.

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 (Anweisungssteil), 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

P₁₇ | R(ᵥ) ⇒ (R, R)  
V | o o 1  
S | σ σ 0

where V₀ is an input and R₀, R₁ are result parameters. The body must contain assignments to R₀ and R₁. If it contains intermediate values, then they are not readable directly from outside of P₁₇.

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

In the procedure (Figure 3), σ denotes the structure of these 8-bit sequences, and mσ with arbitrary m ≥ 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) ∨ Neg(x) ∨ Kla(x)
Sz(x) : Va(x) ∨ Klz(x)
Sq(x,y) : (Sz(x) ∧ ¬Az(y)) ∨ (¬Sz(x) ∧ Az(y))
Der weiße König kann einen Zug machen, ohne dabei in Schach zu kommen.

Die hierbei benutzten Unterprogramme sind:

1. \( R17(V, V) \)
2. \( R128(V, V, V) \)
3. \( S_{17}(1) \)
4. \( S_{128}(V, V, V) \)

Erklärung der Formel P148 in Worten:

(1) ist der Randauszug, der besagt, daß über eine Feldbesetzung \( A5 \) eine Aussage gemacht werden soll.

(2) Diejenige Punkt-Besetzt-Angabe \( x \) welche in der Liste der Spielbesetzung \( V0 \) enthalten ist, deren Komponente Nr. 1 = 1 ist (Zeichen für König in der Numerierung der Steintypen), ergibt den Zwischenwert \( Z0 \).

(3) Es gibt in der Liste der Spielbesetzung \( V0 \) einen Punkt \( x \) der zu \( Z0 \) (Punkt, auf dem der König steht) benachbart ist und der unbesetzt ist oder mit einem schwarzen Stein besetzt ist (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 \( A2(x) \)
2. Two symbols \( x, y \) following each other have to fulfill \( S_{4}(x, y) \)
3. The last symbol \( x \) has to fulfill \( S_{2}(x) \).

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) and (4) are initializations for the repetitive statement which checks condition 2 and the count 5. Condition 3 for the final case is then checked in (1) and the count 4 in (3). The program, by the way, contains mistakes: for example, a count corresponding to (7) is missing for the first symbol. More seriously, the condition \( x = V0[0] \) in (7) should be read as \( x = V0[i] \) and \( i \neq 0 \).

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

```
proc Sa = ([0 : either] bits V0) bool : begin
  bits Z0 := V0[0]; bool R := Az(Z0);
  int eps := 0;
  if Kla(Z0) then eps := 1 fi;
  for i to upb V0 while R do
    hitsZ1 := V0[i];
    R := R A Sq(Z0, Z1);
    if Kla(Z1) then eps := 1 fi;
    if Kla(Z1) then eps := 1 fi;
    R := R \ eps \ 0;
  end;
  Z0 := Z1
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 A5 and A6 are to be interchanged). Zuse's procedure, directly transcribed 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; instead 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 :
```

Communications 1972
of Volume 15
the ACM Number 7
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 \( \Rightarrow \) 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 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


Z49. Zuse, K. Über den allgemeinen Plankalkül als Mittel zur Formulierung schematisch-kombinatorer Aufgaben. Archiv Math. 1 (1948/49), 441-449. (Received Dec. 6, 1948.)


