Seminar Report

Plankalkül

Bernhard Behr
bernhard.behr@student.uibk.ac.at

31 July 2015

Supervisor: Dr.Bertram Felgenhauer, Ph.D.

Abstract

This paper gives a short overview over history and design of ”Plankalkül” as projected by Konrad Zuse in the 1940ies. It will cover basic elements of the syntax trying to put it into the context of modern day programming languages. It will than proceed to try and measure the impact and historic importance of *Plankalkül* in computer science. It will finally look at *Plankalkül* in the context of esoteric programming languages

Contents

1 Introduction

Computer science has become an ever more commanding element of society in the outgoing 20^{th} century. This was arguably made possible, among other things, by the evolution from writing machine code to high level programming languages.

While it was never actually applied in the field, *Plankalkül* is often given credit as the first specimen of the latter type of language. This makes it a rather interesting subject.

This paper will start off by briefly describing the history leading up to and concerning the creation of *Plankalkül* . Next it will describe the basic structure along with the syntax, moving on to more advanced structures. This section will try to showcase parallels to modern programming languages along the way. Finally it will look into impact and classification of *Plankalkül* as an esoteric programming language.

2 History

Konrad Zuse was born in 1910 in Berlin, Germany and brought up in the formerly German town of Braunsberg. following his studies in civil engineering he started working for a German aircraft manufacturer. However, day to day work consisting of often repetitive calculations took its toll prompting Zuseto design a machine capable of completing such tasks. He started buliding his first all mechanic computer around 1936 being, by his own accounts, 'too lazy to do the maths'.[6] However, his Z1 never worked too well. When World War II broke out his endeavours were briefly halted until he managed to convince administration that his efforts were potentially relevant. During the war he was funded by the Third Reich's Aerodynamic Research Institute. His research concerned flutter testing for German fighter planes, in which his Z3 computer was actually used.[3, 6] During this time he started designing *Plankalkül* as a means to describe calculation processes in a general, machine-readable way. He also projected a special *logic machine* to run this language, which was never realized [6]. When post-war chaos prevented him from working on his computers he finished up *Plankalkül* but as first steps towards publication failed to generate feedback a full publication did not come to be until 1972 when nearly 30 years of development had for the most part rendered the language obsolete.[5] Zuse did however get credit for his vision, albeit late.

3 Design of *Plankalkül*

This section will give a short introduction to the language. It will start by looking at the type system, followed by a look at advanced syntactical constructs.

3.1 Type System

3.1.1 Basic Types

Plankalkül only has one basic type. Zuse calls those *truth values*. A truth value can take one of two values, yes or no, or in Zuse's notation '0' and 'L'. Zuse calls the type

of a piece of data its structure. Accordingly, different types may be distinguished via a structure index *Sn*.
Truth values are globally indexed as *S0*. It is worth noting that Zuse seems to allow for both globally fixed indices like *S0* for truth values and 'temporary' assignments of (unused) Structure indices.

3.1.2 Composed Types

Using the basic type of truth values types of arbitrary structure may be derived. These more complex types can be combined to new types as well. This construction principle is represented in the structure index. Assume a binary representation of the digits 0 through 9. It takes four truth values (or bits) to represent those, thus the structure of an according type will consist of four times *S0*. Zuse allows for two different notations: $S1.4$ or $4 \times S0$ may be used interchangeably.

Plankalkül also allows for types of variable structure. In this case the structure index is denoted by the σ symbol.

All non-primitive types can be divided into components; only the primitive truth value is indivisible and thus the smallest component of any type. Whilst this is hardly surprising, what is a genuine feature of *Plankalkül* is its ability to address each component separately. To facilitate this, Zuse introduced a *component index* with Kn[1] denoting the nth component of a type. In a way any non-primitive type can be considered an array of its subtypes, the elements of which can be accessed, read and modified via the component index.
Additionally, a type may also have a name and a set of constraints associated with it. Those are optional features, with the name being a type index An [2], and a set of constraints indexed as Bn [3].

Consider the decimal digits; while we need four bits to express all decimal digits, we don't need all possible combinations. For example we could issue a set of restrictions as:

$$B3 = \neg a3 \vee (\neg a2 \vee \neg a1)$$

With *B3* denoting our restriction set and *a0* to *a3* our single bits. *B3* states that either the first bit is not set (reading the number in a traditional left-to-right way) or neither of the middle bits is set. Thus we can only express the binary representation of our digits. An object *A3* representing digits may now be defined as:

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

Note that an optional type name of the form *Tn* is not used. [1, 7]

[1] K for the German word 'Komponente' for component
[2] Zuse uses the German word 'Art' for type
[3] from the German term 'Beschränkung'

3.2 Advanced Syntactical Constructs

3.2.1 Variables

Plankalkül does feature variables for storing values. A variable is accessed via its variable index *Vn*. Accessing a variable *V1* of the Structure *S1.n* would be written as:

$$\begin{array}{r|l} & V \\ V & 1 \\ K & \\ S & S1.n \end{array}$$

In contrast, to only access the first bit of *V1* we write

$$\begin{array}{r|l} & V \\ V & 1 \\ K & 1 \\ S & S0 \end{array}$$

This illustrates Zuse's concept of a two dimensional notation, where all the required indices have their own line, an odd concept on first sight.

Variable assignment is denoted by the assignment operator:

$$\Rightarrow$$

The component index may be given as another variable. It is written as:

$$\begin{array}{r|l|l|l} & V & \lceil & Z \\ V & 1 & | & 1 \\ K & & \rfloor & \\ A & S1.n & & 9 \end{array}$$

[7]

3.2.2 Calculation Plans

As the name already implies, *Plankalkül* is designed to do calculations according to plans. Those plans can be functions, subroutines or whole programs. Plans are indexed via a plan index *Pn*.

Every plan is divided in two parts, a header[4] defining in and output format, and a body describing the operations carried out on execution. A header of a plan *P17* taking one parameter of structure σ and returning one value of type σ and one single truth value would take the form:

$$\begin{array}{rl|l|l|l} P17 & & R(V) & \Rightarrow & (R, \quad R) \\ & V & 0 & & 0 \quad 1 \\ & S & \sigma & & \sigma \quad 0 \end{array}$$

[4] Zuse calls those parts 'Randauszug' but they are fairly similar in concept to headers

[1] A plan can deal with certain types of data: Input parameters, intermediate results, constants and results.

A plan may also call other plans. It is worth noting though that Zuse did not plan for recursive calls.

Calling a plan is quite similiar to a function call. to call plan *Pn* with parameter x we write:

$$Pn(x)$$

However, this does not return anything. To obtain a result parameter, we need to address it directly. Suppose we need result parameter *Rm* of *Pn* with input parameter x:

$$Pn.m(x)$$

To store this in a variable *V1* we write:

$$Pn.m(x) \Rightarrow V1$$

What catches the eye is the possibility to have one routine of function produce several different results that can be accessed separately just as needed.

You could also arguably see a certain notion similar to the idea of lazy evaluation: A program producing or capable of producing several output parameters is only used with respect to a certain aspect of its capabilities. [7]

3.2.3 Control Flow

The body of a plan is the imperative part of *Plankalkül* . It can contain input parameters, local variables, intermediate results, constants and result parameters and calls to other plans as well as subroutines like loops and conditionals. We will now take a look at the more important elements of control flow in *Plankalkül*

1. Conditionals

   Plankalkül supports the conditional execution of branches. Zuse's construct is mostly equivalent to an if-then-else statement, still the else has to be expressed explicitly via a negation of the if-branch as only one new operator is introduced:

   $$\begin{matrix}\rightarrow \\ \cdot\end{matrix}$$

   For example, a function taking two numbers as arguments and returning the bigger one of the two might look like this:

   $$\begin{array}{c|cccccc} & Max & (V0 & V1) & \Rightarrow & R) \\ V & & 0 & 1 & & 0 \\ A & & 8 & 8 & & 8 \end{array}$$

   $$\begin{array}{cccccccc} V & \geq & V & \begin{matrix}\rightarrow \\ \cdot\end{matrix} & (V & \Rightarrow & R) \\ 0 & & 1 & & 0 & & 0 \end{array}$$

$$\begin{array}{cccccccc} \neg(V & \geq & V) & \vec{\cdot} & (V & \Rightarrow & R) \\ 0 & & 1 & & 0 & & 0 \end{array}$$

A8 refers to a fixed index for the type number [7]

2. Loops

   Another remarkable feature are what Zuse calls *repetition plans*. This term refers to a number of variations of loops, covering both the traditional while- and for loops as well as several specialized cases of the latter. Generally speaking a repetition plan is indexed as Wn[5]. However, every n refers to a certain type of loop. First we consider the traditional while-loop:

$$W \begin{bmatrix} F & \vec{\cdot} & P \\ \bar{F} & \Rightarrow & Fin^2 \end{bmatrix}$$

   The plan P is executed as long as condition F holds. If F is no longer true, the loop stops. Execution resumes with the next command. The *Fin* symbol will be discussed later. The basic for loop takes the form:

$$\begin{array}{ccc} W0 & (v) & \left[\begin{array}{ccccc} Z & \times & v & \Rightarrow & Z \\ 0 & & 0 & & 0 \end{array}\right] \left| \begin{array}{ccc} Z & \Rightarrow & R \\ 0 & & 0 \end{array}\right. \\ & 1 & \end{array}$$

   In this example, *v1* is a variable specifying how often the loop is to be executed. The body of the loop states that an intermediate result *Z0* is multiplied with another variable *v0* and the result is stored in *Z0*. After the last iteration the value in *Z0* is stored in *R0*.

   Note that, contrary to the traditional for loop, only a variable giving the number of loop executions is needed; creating, initializing, incrementing and comparing of loop variables is handled implicitly. In addition there are several specialized forms of for loops:

$$\begin{array}{lllcll} W1(n) & [P(i)] & 0 \Rightarrow i & | & W & \left[i < n \ \vec{\cdot} \ \left[P(i)|i+1 \Rightarrow i\right]\right] \\ W2(n) & [P(i)] & n-1 \Rightarrow i & | & W & \left[i > 0 \ \vec{\cdot} \ \left[P(I)|i-1 \Rightarrow i\right]\right] \\ W3(n,m) & [P(i)] & n \Rightarrow i & | & W & \left[i < m \ \vec{\cdot} \ \left[P(i)|i+1 \Rightarrow i\right]\right] \\ \frac{n \leq m}{W4(n,m)} & [P(i)] & n \Rightarrow i & | & W & \left[i > m \ \vec{\cdot} \ \left[P(i)|i-1 \Rightarrow i\right]\right] \\ \frac{n \geq m}{W5(n,m)} & [P(i)] & n \Rightarrow i & | & W & \left[i \neq m \ \vec{\cdot} \ \begin{bmatrix} P(i) \\ m > n \ \vec{\cdot} \ (i+1 \Rightarrow i) \\ m < n \ \vec{\cdot} \ (i-1 \Rightarrow i) \end{bmatrix}\right] \end{array}$$

   [7]

[5] from the German term 'Wiederholungsplan'

3. Fin Symbol

   The Fin symbol is on first sight a rather basic feature much like a break statement. However, the corresponding Fin^2 symbol is an interesting concept: It does not only stop the current routine, but also the next higher, i.e. calling routine. This is supposed to work for all Fin^n with arbitrary n. Interestingly this is a feature that even today is not standard. [7]

4 Impact of *Plankalkül*

As was mentioned earlier, at the time of its conception *Plankalkül* barely had any impact at all. Still it has since often been called the world's first high level programming language. It has also been brought up in discussions about esoteric languages, both of which aspects will be addressed briefly in this concluding section.

4.1 *Plankalkül* as a Forerunner of Modern Programming Languages

The English edition of Wikipedia states *Plankalkül* to be "the first high-level non-von Neumann programming language to be designed for a computer"[5]. While not necessarily being the most scientifically accurate of sources, this arguably is a great indicator for public perception of a topic. It also has a kind of 'happy end' ring to it, the visionary who finally gets his credit. But is this accurate?

There are certainly points in favour of *Plankalkül* as a milestone in computer science. As the third section of this paper tries to illustrate, the basic features and elements of *Plankalkül* are rather similar to the likes of *Java* or *C*. Also one must not forget those are concepts that were at the time rather less obvious than they appear today.

Konrad Zuse also planned on building a machine running *Plankalkül* and considering his record, who is to say he would not have succeeded, potentially changing the development of computer science drastically, had it not been for the throwbacks of the war.

This argument gains momentum when you consider it took about a decade for computer science to even start moving away from writing machine code.

However, *Plankalkül* was, at least at the time, not implemented. Now one has to wonder how Konrad Zuse would have gone about doing this. There are several fields of concern. Maybe the most striking is the one of building a compiler; after all, without a compiler *Plankalkül* is little more than a convention for writing down commands. Saying a certain construct will execute a certain operation a certain number of times is one thing, making a computer actually do this on the other hand seems to be quite a bit more of a challenge. It is of course hard to judge the difficulties of conceiving all this with the benefit of hindsight, but one could at least argue its easier said than done. Another point is the practicality of the proposed

syntax. especially the two dimensional element to the notation seems like some challenge to achieve[6]. Speaking of the potential woes of *Plankalkül* concerning input, there is of course also the question of output; for example, looking at *Plankalkül* there appears to be no feasible way to implement a '99-Bottles-of-Beer' program[2], at least not without getting reasonably abstract.

Of course, those are things that might have been worked out had it been implemented, but this leads to another point: At times it seems unfinished, with certain conventions open to interpretation and, at times, apparently contradictions regarding several details.[7]

Still, the extremely innovative nature of Zuse's ideas that turned out to be leading the way decades later probably outweighs those arguments by quite some margin.

4.2 *Plankalkül* as an Esoteric Language

In the context of this seminar report, which after all is a seminar about esoteric languages, it is also fitting to have a short look at this classification of *Plankalkül* . According to Wikipedia, an esoteric programming language is "a programming language designed to test the boundaries of computer programming language design, as a proof of concept, as software art, or as a joke."[4] Considering this definition, it appears justified to call *Plankalkül* esoteric, as it certainly was testing the boundaries of computer programming language design at the time.

On the other hand looking at the likes of *Piet* or *Lolcode* it's not exactly obvious to lump those together. Many esoteric languages seem to put much more weight on the humorous element than on pushing technical boundaries. Putting those together in the same category does seem to diminish Zuse's work quite a bit.

Also, even though the language certainly was very different than anything of its time, and does look odd today as well, supporting the claim for it to be esoteric, the things making it look as different today for the most part are merely cosmetics. That is to say,once you look under the surface of weird letters and numbers atop each other Zuse did come up with a system that is strikingly similar to today's languages designed decades later. While esoteric languages go out of their way to be different, Zuse simply did not have much to be different from. So you might just as well say all the rest of programming languages is esoteric for making up strange new naming conventions instead of using Zuse's system.

All things considered, you can certainly call *Plankalkül* an esoteric language, but it's worth keeping in mind that it may just be a matter of perspective.

[6] At least typing it in LaTeXis a bit of a hassle

[7] for example the index 'A' seems to be used as 'Art' for type as well as 'Angabe' for object

5 Fin

Plankalkül was conceived during and after World War II. Its design is strikingly similar to modern programming languages like *Java*. It features most if not all basic constructs needed in today's languages like data types, functions, conditionals or loops. Considering the time of its conception it is hard to argue against it being a major milestone in computer science. However its historic influence was cut short by adverse circumstances.

References

[1] F. Bauer and H. Wössner. The.

[2] Oliver, Gregor, and Stefan. 99 Bottles-of-Beer. Website, 2015. available online at http://www.99-bottles-of-beer.net/.

[3] R. Slater. *Portraits in Silicon*. The Massachusetts Institute of Technology, ...Massachusetts?, 1987.

[4] Wikipedia. Esoteric programming language. Website, 2015. available online at https://en.wikipedia.org/wiki/Esoteric_programming_language.

[5] Wikipedia. Plankalkül. Website, 2015. available online at https://en.wikipedia.org/wiki/Plankalkül.

[6] H. Zuse. Konrad Zuse Biografie. Website, 2007. available online at http://www.konrad-zuse.de/.

[7] K. Zuse. Der Plankalkül. Website, 1946. available online at http://zuse.zib.de/item/DB2j_t_w1fbxvaiq.