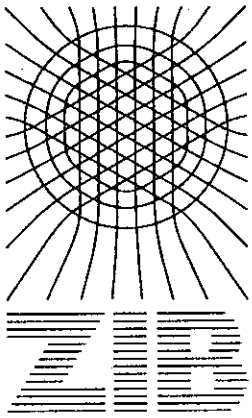
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Heilbronner Str. 10, D-1000 Berlin 31



Wolfgang K. Giloi

Konrad Zuses Plankalkül als Vorläufer moderner Programmiermodelle

Technical Report TR 90–13 (November 1990)

Konrad Zuses Plankalkül als Vorläufer moderner Programmiermodelle*

von

Wolfgang K. Giloi

Einleitung

Wir alle kennen Konrad Zuse als den Erfinder des Digitalrechners. Wir wissen, daß seine Idee des binär arbeitenden Rechenautomaten sein originärer Einfall war -- von Babbage und dessen Arbeiten bekam er erst Kenntnis, als er seine ersten Patente bereits angemeldet und seinen ersten Automaten bereits konstruiert hatte. Von Konrad Zuse stammt auch die geniale Erfindung der binären Gleitpunkt-Darstellung von Dezimalzahlen als Grundlage seiner "Maschinen-Arithmetik". Sie bildet bis heute die Grundlage der Maschinen-Arithmetik jedes wissenschaftlichen Rechners.

Die auf die Leibnizsche "Dyadik" zurückgehende Erkenntnis, daß man die arithmetischen Operationen auch in einem dualen Zahlensystem definieren und ausführen kann, führte bei Konrad Zuse zu der Erkenntnis, daß sich die Arbeitsweise der Komponenten seiner Maschine durch logische Ausdrücke formal beschreiben lässt. Er entwickelte hierfür zunächst einen eigenen, *Bedingungskombinatorik* genannten Formalismus. Erst später kam ihm das bereits bestehende theoretische Gebäude der Aussagenlogik zur Kenntnis. Zuse erkannte, daß die Aussagenlogik die Bedingungskombinatorik einschloß und verwendete von da an die letztere zur Verhaltensbeschreibung bzw. zum Entwurf seiner Relais-Schaltkreise. Damit ging er den gleichen Weg, den Shannon mit seiner Schaltalgebra gegangen war, von der Konrad Zuse aber keine Kenntnis hatte.

Es ist auch seit langem wohlbekannt, daß Konrad Zuses rastloser Geist in den Jahren 1943-45, insbesondere, als er durch die Kriegsereignisse schliesslich nicht mehr an der Realisierung seiner Maschinen weiterarbeiten konnte, sich damit beschäftigte, ein formales System für das maschinelle Rechnen aufzustellen, welches numerisch-algorithmisches Rechnen einschloss, darüberhinaus sich aber auch insbesondere zur Behandlung kombinatorischer Probleme eignete: den "Plankalkül". Aber haben wir wirklich einen genauen Begriff davon, welche Denkweisen und welche besonderen Konzepte im Plankalkül zu finden sind ? Wer hat den Plankalkül wirklich studiert ? In welchem Maße hat dieser die Entwicklung der höheren Programmiersprachen vorweggenommen, warum hat es sie nicht noch stärker beeinflusst ? Alles dies sind Fragen, für die wir in diesem Beitrag eine Antwort zu finden versuchen.

Von Konrad Zuse selbst, aber auch von anderen Autoren [Bauer 72] wird der Plankalkül als ein Vorläufer der gängigen, mit Fortran und Algol beginnenden algorithmischen Programmiersprachen angesehen. Diese Sicht stützt sich auf eine Reihe von äusseren Merkmalen, die der Plankalkül bereits mit den genannten Sprachen gemeinsam hat. Genannt werden hier unter anderem:

- der Variablenbegriff, einschliesslich Variablendeklaration und Wertezuweisung
- die bedingte Ausführungsanweisung und
- der Begriff des Unterprogramms.

---

* Vortrag, gehalten anläßlich der Akademischen Feier der Technischen Universität Berlin am 29. Oktober 1990 aus Anlaß des 80. Geburtstages von Konrad Zuse.

Konrad Zuse hat verschiedentlich seine Enttäuschung darüber geäussert, daß der Plankalkül in den frühen Jahren der Entwicklung höherer Programmiersprachen sowenig Beachtung gefunden hat. Nun ist es nicht weiter verwunderlich, daß diese grundlegende Arbeit international bis heute fast unbekannt geblieben ist.

Die Herrschaft des faschistischen Ungeistes in Deutschland hatte bereits seit 1933 dazu geführt, daß dieses Land nicht mehr wie zuvor Wissenschaftler aus aller Welt anzog und daß die deutsche Sprache ihre Bedeutung als eine der Haupt-Wissenschaftssprachen verlor. Dies galt erst recht während des Krieges und nach der Niederlage. Erst im vergangenen Jahr führte die GMD das verdienstvolle Unternehmen aus, Zuses Schrift über den Plankalkül und der damit im Zusammenhang stehende Entwurf einer Doktorarbeit aus dem Jahre 1944 in englischer Sprache zu veröffentlichen [Zuse 89]. Man muß dazu aber leider anmerken, daß dieses Unternehmen noch sehr viel verdienstvoller ausgefallen wäre, wenn die englische Übersetzung von einem dieser Sprache Kundigen durchgeführt worden wäre, und wenn nicht fast jede Seite von Druckfehlern wimmeln würde, ein Tatbestand, der den Genuß der Lektüre naturgemäss stark beeinträchtigt.

Konrad Zuse zeigt sich besonders darüber enttäuscht, daß aber auch im eigenen Land die Wissenschaftler, die in der zweiten Hälfte der fünfziger Jahre an der Entwicklung der Sprache Algol mitarbeiteten, die Priorität einiger wesentlicher Konzepte, die vor Algol bereits im Plankalkül vorhanden sind, nicht gebührend anerkannten, obwohl diesen Kollegen seine Schrift sehr wohl bekannt war [Zuse 89, S.24].

Dies ist nach unserer Einschätzung aber wahrscheinlich weniger auf Böswilligkeit als auf Unverständnis zurückzuführen. Um unsere Vermutung zu belegen, wollen wir in diesem Beitrag herauszuarbeiten versuchen, was die wesentlichen Merkmale des Plankalküls sind, inwiefern sich dieser von den frühen algorithmischen Programmiersprachen wie Fortran oder Algol unterscheidet, und wo der Plankalkül nach heutiger Sicht einzuordnen ist. Dabei wollen wir uns nicht bei Äusserlichkeiten aufhalten, wie z.B. der Ähnlichkeit von Kontrollkonstrukten, sondern zeigen, daß der Plankalkül in seinem Modell des Computing sehr viel allgemeiner als die frühen algorithmischen Sprachen ist.

Die "Von Neumann" - Sprachen

Eine Programmiersprache ist die Ausprägung eines bestimmten *Programmiermodells*. Das Programmiermodell kann durch die Eigenschaften einer *abstrakten Maschine* definiert werden.

Die den Sprachen vom Algol-Typ (wir schliessen hierin PL/I und PASCAL mit ein) zugrunde liegende abstrakte Maschine ist die "von Neumann"-Maschine, und zwar in der speziellen Ausprägung der *Kellermaschine*. Diese Maschine ist vor allem durch folgende Merkmale gekennzeichnet.

- Die Objekte, auf denen die Maschine arbeitet, sind die einzelnen Speicherplätze. Somit gibt es nur skalare Objekte. Variablennamen sind gleichbedeutend mit symbolischen Adressen; dies geht soweit, daß die Algol-Sprachen explizit den Datentyp *Speicheradresse* (Zeiger) beinhalten.

- Eine Berechnung besteht in der schrittweisen (sequentiellen) Veränderung des Speicherzustands der Maschine. Somit gibt es auch nur skalare Operationen.

- Das Auftreten von Datenstrukturen (z.B. Arrays) in den Anwendungsalgorithmen äussert sich somit in Programmschleifen, in denen die einzelnen, durch Indizierung adressierten (skalaren) Elemente der Arrays nacheinander verarbeitet werden. Die Sprache selbst enthält aber keine Datenstruktur-Typen.

- Die Anweisungen des Programms können zusammengesetzte Ausdrücke enthalten, die über einen Keller-Mechanismus abgearbeitet werden.

- Prozeduren sind in einer Blockstruktur geschachtelt, so daß sie über einen "Prozedurkeller" abgearbeitet werden können.

Ein hervorstechender Unterschied zwischen den "von Neumann" - Sprachen und dem Plankalkül besteht darin, daß die "von Neumann - Sprachen" im Gegensatz zum Plankalkül keine Datenstruktur-Typen kennen. Unter dem im vergangenen Jahrzehnt zu grosser Bedeutung gekommenen Aspekt der Parallelarbeit heisst das, daß "von Neumann" - Sprachen keinerlei Ausdrucksmittel besitzen, mit denen der Programmierer Anweisungen zur parallelen Ausführung geben könnte. Sollen solche Programme dennoch parallel ausgeführt werden, dann bedarf dies einer vorhergehenden "Parallelisierung", die vom Compiler vorgenommen werden kann. Es hat lange, bis Mitte der siebziger Jahre gedauert, bis sich in der "Algol-Welt" die Erkenntnis durchzusetzen begann, daß das Fehlen von Datenstruktur-Typen ein Nachteil ist [Deijkstra 76].

Den Grund dafür, daß der Plankalkül nicht als "von Neumann" - Sprache konzipiert wurde, sehen wir darin, daß die frühen Zuse-Maschinen eben keine "von Neumann" - Rechner waren. Es fehlten bei ihnen zunächst die Programmschleifen wie auch der Keller. Erst die Z4 liess in ihrer Planung (1942) wie auch in ihrer späteren Ausführung (1950) Programmschleifen zu.

Nun hatte Konrad Zuse selbstverständlich immer die prinzipielle Möglichkeit gesehen, nicht nur die Daten, sondern auch die Befehle eines Programms im Datenspeicher ablegen und von dort aus aufrufen zu können. Der Datenspeicher als der aufwendigste Teil der frühen Zuse-Maschinen war aber dazu viel zu kostbar. Daher zog Zuse es vor, für das Programm einen gesonderten Lesespeicher vorzusehen, der in der Form eines Lochstreifens realisiert wurde. Dies setzte aber eine *schleifenfreie Programmierung* voraus. Iterationen gibt es natürlich auch im Plankalkül - es muß sie geben, sonst wären viele Algorithmen nicht ausführbar. Bei den im Plankalkül gegebenen Beispielen erstrecken sich Iterationen in der Regel über eine Anweisung oder *Programmgleichung*, wie Zuse das nennt (iterationen über mehrere Anweisungen sind aber durchaus möglich). Dabei muß hinzugefügt werden, daß die Programmgleichungen des Plankalküls aber sehr viel mächtiger sein können als die Anweisungen der "von Neumann" - Sprachen, da sie komplexe Operationen auf Datenstrukturen aufzurufen gestatten.

Damit hat der Plankalkül mehr konzeptionelle Ähnlichkeiten mit frühen "nicht-von- Neumann" - Sprachen als mit Fortran oder Algol.

"Nicht-von-Neumann" - Sprachen

Programmiersprachen, die stark vom "von Neumann" - Modell abweichen, sind zum Beispiel die Sprachen APL und SETL. Beide kann man als "mengen-orientierte" Sprachen ansehen, wobei Mengen in dem um 1960 entstandenen APL *geordnete Mengen* sind, dargestellt in der Form von beliebig-dimensionalen rechteckigen Feldern (*arrays*). Mengen in dem ein Jahrzehnt später von Jack Schwarz definierten SETL entsprechen hingegen dem üblichen, allgemeinen Mengenbegriff, und die Operationen von SETL schliessen damit die üblichen Mengenoperationen ein.

Der Entwicklung von APL durch Kenneth Iverson lag das erklärte Ziel der Definition einer neuen, ausdrucksstärkeren Algebra zugrunde, in der algebraische Operationen auf Strukturen mit einem prozeduralen Ablauf verbunden werden. Dies impliziert die Einführung des *Zustandsbegriffs*, eine Sicht, die es in der mathematischen Algebra ja nicht gibt. Im Plankalkül wird ein ähnliches Ziel verfolgt.

Aus heutiger Sicht sehen wir APL hauptsächlich als eine Programmiersprache, die sehr mächtige Operationen auf Array-Strukturen und damit eine *weitgehend schleifenfreie*

Programmierung erlaubt. Dieser Aspekt von APL bildet somit eine sehr gute Grundlage für Parallelarbeit, und in der Tat sind die Array-Datentypen von Fortran-90 -- das ist die modernste Version eines parallelen Fortrans -- von APL entlehnt.

APL enthält unter anderem eine Fülle von Operationen, die dazu genutzt werden können, Elemente von mehrdimensionalen Feldern aufgrund bestimmter Attribute zu selektieren. Man kann nun die Arrays von APL als geordnete Mengen sehen, und die auf ihnen definierten Selektionsoperationen als Operatoren zur Untermengenbildung aufgrund gegebener Prädikate. Auch hierin besteht, wie wir noch sehen werden, eine besondere Verwandtschaft zum Plankalkül.

Konzepte des Plankalküls

Die Grundidee des Plankalküls

Die wesentlichste Grundidee des Plankalküls ist die binäre Repräsentation von Objekttypen. Dabei gibt es neben skalaren Datentypen auch Datenstrukturtypen (im Plankalkül unterschiedslos mit *Struktur* bezeichnet). Dabei werden nicht nur die Werte der Elemente der Datenstrukturen binär kodiert, sondern auch die Struktur der Anordnung der Elemente.

Ein Programm besteht im Plankalkül aus einer Folge von *Programmgleichungen*. Syntaktisch sehen die Programmgleichungen, wie wir noch ausführen werden, den Anweisungen einer "von Neumann" - Programmiersprache ähnlich. Semantisch spielen die Programmgleichungen jedoch die Rolle des *Prozeduren* (oder *Blocks*) in den Algol-ähnlichen Sprachen. Das heisst, Programmgleichungen sind die Einheiten der Programmausführung, die mit Eingabewerten versorgt werden und als Folge ihrer Ausführung einem Ausgabeparameter einen Wert zuweisen, der dann seinerseits ein Eingabe-parameter einer nachfolgenden Programmgleichung sein kann.

Die Syntax der "Programmgleichungen" des Plankalküls

Konrad Zuse verwendete im Plankalkül eine aus der Sicht der späteren Programmiersprachen-Entwicklung ungewöhnliche Syntax, die dadurch gekennzeichnet ist, daß jede Programmgleichung aus mehreren Zeilen besteht. Die oberste Zeile entspricht dabei der üblichen Anweisung, bestehend aus einem Ausdruck, einem Zuweisungssymbol und einer Resultatvariablen, der das Ergebnis der Ausdrucksauswertung zugewiesen wird. Dabei werden aber noch keine konkreten Variablennamen eingesetzt, sondern "generische" Bezeichner, die angeben, ob es sich um eine Eingabegrösse oder um das Resultat handelt (diese Darstellung stellt eine gewisse Redundanz dar, da die Unterscheidung zwischen Ein- und Ausgabevariablen bereits durch deren Position links oder rechts vom Zuweisungssymbol definiert ist). Die konkreten Namen - in der Regel sind dies Indizes - der in der Anweisung auftretenden Variablen werden in der zweiten Zeile hinzugefügt.

Die dritte Zeile der Programmgleichung beinhaltet die Typendeklaration der Variablen. Diese Festlegung resultiert aus einer Sicht von Variablen als im wesentlichen *lokale Parameter* der Prozedur. Damit ist der Gültigkeitsbereich einer Typendeklaration die Programmgleichung. Dies schliesst nicht aus, daß eine Resultatvariable an eine nächste Anweisung als Eingabe weitergereicht wird (was wie üblich dadurch geschieht, daß sie an der entsprechenden Stelle wieder referenziert wird).

Kontrollkonstrukte im Plankalkül

Es wurde bereits erwähnt, daß im Plankalkül die *Programmgleichung* bzw. auch das *Unterprogramm* die Rolle der *Prozedur* der Algol-ähnlichen Sprachen spielt. Kontrollkonstrukte beziehen sich daher in der Regel auf Programmgleichungen.

Der elementarste und dabei wichtigste Programmkonstrukt erlaubt die bedingte Ausführung einer Programmgleichung: diese wird nur dann ausgeführt, wenn die Bedingung wahr ist. Zu vergleichen ist dieses Konstrukt damit mit den *guarded commands*, welche über 30 Jahre später von Dijkstra erfunden wurden [Dijkstra 76]. Selbstverständlich kann man mit diesem Konstrukt die gleiche Wirkung erzielen, wie mit dem IF- oder dem IF...THEN...ELSE - Konstrukt der späteren höheren Programmiersprachen.

Eine zweites, ebenso wichtiges Konstrukt erlaubt die Iteration von Programmgleichungen. Das Iterationskonstrukt entspricht den FOR-Schleifen der späteren höheren Programmiersprachen, wobei es im Plankalkül auch bereits die üblichen Varianten der Inkrementierung oder Dekrementierung des Iterationszählers gibt.

Skalare Datentypen im Plankalkül

Es gibt im Plankalkül vordefinierte Standard-Typen und benutzerdefinierte Typen. Skalare Standard-Typen sind zum Beispiel [Zuse 89, S.147)]:

- komplexe Zahlen
- reelle Zahlen
- Ganzzahlen
- nur positive oder nur negative Ganzzahlen
- rationale Zahlen
- positive rationale Zahlen
- Dualzahlen

Für die Standard-Typen wurden feste symbolische Bezeichner eingeführt. Die Bezeichner können noch dahingehend verfeinert werden, daß sie Eigenheiten der binären Repräsentation ausdrücken, wie zum Beispiel die Unterscheidung zwischen der Zahlendarstellung nach Betrag und Vorzeichen und der Komplement-Darstellung. Auf den numerischen Standard-Typen ist etwa die Fülle von arithmetischen Funktionen definiert, wie sie in jeder höheren Programmiersprache einschliesslich der mathematischen Funktionsbibliothek vorkommen.

Benutzerdefinierte Typen können alles mögliche sein. Typische Beispiele sind [Zuse 89, S.12]:

- Personen
- Alter
- Geschlecht
- Familienstand,

oder andere, personenbezogene Daten, aber auch:

- die Felder eines Schachbretts
- die Figuren des Schachspiels einschliesslich ihrer Zugmöglichkeiten.
- die Kante eines Graphen,

oder was immer die Anwendung erfordert.

Die Möglichkeit der Definition beliebiger Datentypen durch den Programmierer wird dadurch gewonnen, daß dieser für jeden Typ, den er einführt, auch dessen binäre Repräsentation einschliesslich ihrer Semantik definieren muß. D.h. die binäre Repräsentation ist immer eine Bitkette, deren Interpretation durch die Typendefinition festgelegt wird.

Alle Operationen der benutzerdefinierten Typen sind jetzt Operationen auf den binären Repräsentationen, d.h. Ausdrücke des Aussagenkalküls bzw. der Prädikatenlogik. Damit lassen sich dann aus einer Menge von Objekten eines Typs Untermengen aufgrund prädikatenlogischer Aussagen bilden. Wir illustrieren dies durch das im folgenden gegebene Beispiel.

BEISPIEL

BITKETTE: $A_9\ A_8\ \ A_7\ \ A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0$

DEFINITIONEN:

$A_6 A_5 A_4 A_3 A_2 A_1 A_0$ = Alter, $0 \leq \text{Alter} \leq 127$

A_7 = Geschlecht, A_7 = 0: weiblich
1: männlich

$A_9 A_8$ = Familienstand, $A_9 A_8$ = 00: ledig
01: verheirated
10: verwitwet
11: geschieden

PRÄDIKAT:

$$\text{Präd}(x) \equiv \bar{A}_7(x) \wedge A_9(x) \vee \bar{A}_8(x) \wedge \bar{A}_6(x) \wedge \bar{A}_5(x) \wedge A_4(x)$$

$\equiv$ "alle unverheirateten weiblichen Personen im Alter zwischen 16 und 31 Jahren"

Stattdessen könnte man den obigen Booleschen Ausdruck auch als Suchschlüssel für ein assoziatives Suchen durch eine Liste interpretieren.

Natürlich könnte man dieses Beispiel auch in Algol oder einer ähnlichen Sprache formulieren oder - effizienter - in Maschinencode. Ein formales System für diese Art von Rechneranwendungen hat es nach dem Plankalkül aber eigentlich erst viele Jahre später mit der

Coddschen Relationalen Algebra gegeben, entsprechend dem Stand der siebziger Jahre dann natürlich in einer universelleren und vollständigeren Form und auf einer wesentlich höheren Abstraktionsebene.

Datenstrukturen im Plankalkül

Wir haben bereits ausgeführt, daß im Plankalkül nicht nur die skalaren Datentypen binär repräsentiert werden - implizit bei den Standard-Typen und explizit bei den benutzerdefinierten Typen, sondern auch die Zusammenfassung von skalaren Datenelementen in Datenstrukturtypen.

Der Grund-Datenstrukturtyp im Plankalkül ist der *binäre Baum*, dessen Knoten sich ja unmittelbar durch Bitketten adressieren lassen. Andere Strukturen, wie insbesondere Felder (Arrays) müssen auf binäre Bäume abgebildet werden. Der Benutzer sieht damit auch bei den Arrays deren Repräsentation, eine Umständlichkeit, mit der die Allgemeinheit des Prinzips der Darstellung von Datenstrukturen im Plankalkül bezahlt werden muß.

Eine andere, bedeutsame Datenstruktur des Plankalküls ist der eine Relation darstellende *verallgemeinerte Graph* [Knuth 75]. Im Plankalkül werden Relationen durch einfaches Aufzählen der Liste aller Relationspaare dargestellt. Damit lassen sich dann auch geometrische Strukturen darstellen. Konrad Zuse hat sich in den Beispielen zum Plankalkül mit einer Reihe graphentheoretischer Darstellungen und Untersuchungen beschäftigt.

Auch hier gilt, daß solche Darstellungen natürlich auch in der Maschinensprache oder einer Algol-ähnlichen prozeduralen Programmiersprache implementiert werden können. Der Unterschied ist wieder der, daß die einfache Implementierung einer Struktur durch ein entsprechendes Programm natürlich noch kein formales System darstellt.

Zusammenfassung

Wir hoffen, mit dieser Übersicht aufgezeigt zu haben, daß Konrad Zuse bereits 1943-45 einen ganz anderen, sehr viel weitergehenderen Begriff von *Computing* hatte, als es über ein Jahrzehnt später John Backus mit Fortran oder die Väter von Algol hatten. Von Motivation und Zielsetzung her stand ihm da Iverson mit APL und später Codd mit der relationalen Algebra wesentlich näher. Insbesondere die starke Betonung einer prädikatenlogischen Fundierung des Plankalküls berechtigt sogar in gewissem Grade dazu, den Plankalkül als einen sehr frühen Vorläufer des *logischen Programmierens* [Kowalski 74] anzusehen, ist doch die Hornklausellogik von Prolog eine Untermenge der Prädikatenlogik erster Stufe. Allerdings darf man diese Parallele auch wieder nicht zu weit ziehen, da es im Plankalkül natürlich kein Resolutionsprinzip [Robinson 65] gibt.

Konrad Zuse sollte sich aber nicht allzusehr darüber grämen, daß die Väter von Algol, die einige wesentliche Konzepte ihrer Sprache bereits im Plankalkül vorweggenommen sehen mussten, ihm die gebührende Anerkennung vorenthielten. Sein Ansatz ging weit über die enge Sicht der algorithmischen Sprachen hinaus. In der Tat haben gerade die Algol-Hohepriester später selbst dazu beigetragen, daß Algol sich nicht zu einer der in der praktischen Anwendung führenden Sprache entwickeln konnte. Dafür war Algol68 mit seinem übertriebenen Deklarations-Overhead zu umständlich. Und der schon scholastisch anmutende Glaubensstreit, wie schädlich denn nun das GOTO sei, hat hier auch nicht gerade geholfen. Der Siegeszug von C, einer vom puristischen Software Engineering - Standpunkt der siebziger Jahre greulichen Sprache, zeigt, daß Flexibilität und Effizienz am Ende die gefragteren Eigenschaften einer Programmiersprache sind.

Für die Entwicklung der Informatik war es ein Verlust, daß durch die Zeitumstände bedingt, der Plankalkül praktisch unbekannt blieb. Hätte dieser den gleichen Bekanntheitsgrad erfahren

wie die berühmte Schrift von Burks, Goldstine und von Neumann [TAUB 63], einige Entwicklungen der Computergeschichte wie das logische Programmieren, die relationale Datenbank, aber auch die Entwicklung standardisierter Formen der Wissensrepresentation in der Künstlichen Intelligenz wären sicher davon befruchtet worden und hätten sich wahrscheinlich um einiges früher entwickelt.

Literatur

[Bauer 72] Bauer F.L., Wössner H.: *The Plankalkül of Konrad Zuse, a Forerunner of Today's Programming Languages*, Elektronische Rechenanlagen 1972, H.2

[Zuse 89] Zuse K.: The Plankalkül, GMD-Bericht Nr. 175, Oldenbourg-Verlag München-Wien 1989

[Dijkstra 76] Dijkstra E.W.: A Discipline of Programming, Prentice-Hall, Englewood Cliffs N.J. 1976

[Knuth 75] Knuth D.E.: The Art of Computer Programming, Vol.1, Addison-Wesley, Reading Mass. 1975 (2nd. ed.)

[Kowalski 74] Kowalski R.: *Predicate Logic as Programming Language*, Proc. IFIP Congress 1974

[Robinson 65] Robinson J.A.: *A Machine Oriented Logic Based on the Resolution Principle*, J. ACM 12 1965