Geschichte der deutschsprachigen Informatik Programmiersprachen und Übersetzerbau

Gerhard Goos
Fakultät für Informatik
KIT

7. August 2017

Zusammenfassung

Zusammenfassung: Kurzdarstellung wichtiger Entdeckungen, Entwicklungen und Erfindungen der deutschsprachigen Informatik im Bereich Programmiersprachen und Übersetzerbau.

1 Der Beginn

Den Beginn der modernen Informatik im deutshsprachigen Raum bilden die Unentscheidbarkeitssätze von Kurt Gödel, (Gödel, 1931), also ein Ergebnis der Logik.

Den praktischen Beginn der modernen Informatik im deutschsprachigen Raum markieren die Arbeiten von Konrad Zuse ab 1936. Dazu zählt seine Entscheidung das Binärsystem zu benutzen, vor allem aber die Verwendung der binären Gleitkommadarstellung für die Verarbeitung von Zahlen in der Relaismaschine Z1. Zwar waren halblogarithmische Zahldarstellungen der Form $m \times 60^e$ bereits den Sumerern vor 4700 Jahren bekannt und die Mathematiker und Naturwissenschaftler benutzen seit langem dezimal $m \times 10^e$. Aber die Zahldarstellung im Binärsystem $m \times 2^e$ mit normalisiertem $m = 1, m_1 m_2 \ldots$ war neu. Auch, daß die führende Ziffer $m_0 = 1$ wie auch im heutigen Standard (IEEE 754, 2008) nicht explizit gespeichert wird, findet sich bereits bei Zuse. Die allgemeine Entwicklung von Rechenanlagen übernahm die Gleitkommaarithmetik erst in den 50er Jahren. Die IBM 704 war 1954 der erste weit verbreitete Rechner mit Gleitkommaoperationen in Hardware. Zuvor arbeitete man mit Festkommaarithmetik, bei der die Lage des Dezimalkommas mühsam getrennt festgelegt wurde. Beim Umgang mit Geld ist die Art der dezimalen Rundung gesetzlich festgeschrieben und mit binärer Arithmetik nicht einfach nachvollziehbar.

Die zweite große Leistung Konrad Zuses im Bereich Programmiersprachen war die Erfindung des *Plankalküls* 1944/45, der ersten höheren Programmiersprache, (Zuse, 1972; Bauer und Wössner, 1972). Zwar wurde dies erst 1972 veröffentlicht und die zweidimensionale Scchreibweise fand bisher keine Nachahmer. Aber es handelt sich um die erste **goto**-freie Programmiersprache mit Datentypen, die sich anderswo erst im Laufe der 60er

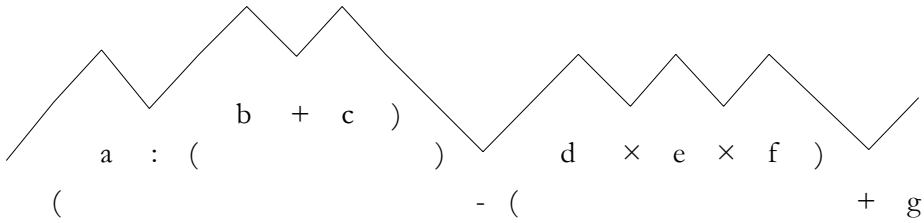

Abbildung 1: Rutishausers Klammergebirge

Jahre etablierten. Konrad Zuse war übrigens auch ein begnadeter Maler, wie viele Bilder z. B. aus seiner Zeit in Hopferau/Allgäu am Ende des zweiten Weltkriegs zeigen. []

2 Die 50er Jahre

(Wilkes et al., 1951) hatten 1951 das erste Programmierhandbuch für das Programmieren in Maschinensprache auf der EDVAC veröffentlicht. Zugleich begann die Suche nach Verarbeitungsmöglichkeiten für arithmetische Ausdrücke, die in gewöhnlicher mathematischer Schreibweise dargestellt waren.

Die erste Lösung hierfür fand Heinz Rutishauser in seiner Habilitationsschrift, (Rutishauser, 1951), mit dem sogenannten Klammergebirge. Rutishauser implementierte diese Lösung auch auf der ERMETH, dem unter Leitung von Prof. Stiefel gebauten Rechner der ETH Zürich. Das Verfahren hatte allerdings quadratischen Aufwand in der Anzahl der Symbole einer Formel. Später und bis heute benutzt man die umgekehrte polnische Normalform oder Postfixform von Jan Łukasiewicz (1878-1956), mit der sich die Berechnung mit linearem Aufwand erledigen läßt.

Rutishauser war auch eine der treibenden Kräfte bei der Definition von ALGOL 58 und ALGOL 60.

Die eigentliche Arbeit an höheren Programmiersprachen und dem Bau von Übersetzern begann im deutschsprachigen Raum 1955 mit den Herren Heinz Rutishauser, Klaus Samelson und Friedrich L. Bauer. Dazu stieß etwas später der Darmstädter Hermann Bottenbruch. Ihr Ziel war eine einheitliche, maschinenunabhängige Programmiersprache, vgl. (Bauer, 1994).

Dazu formulierte Herr Samelson bereits 1955, (Samelson, 1957), die Grundzüge des Kellerprinzips und zwar nicht nur für die Berechnung arithmetischer Ausdrücke, sondern auch für offene und geschlossene Unterprogramme, also für Programmblöcke. Die Grundidee des Verfahrens geht auf Helmut Angstl zurück, der bereits 1950 in einem Vortrag im Logik-Seminar von Prof. Britzelmayer an der LMU eine Vorstufe des Kellerprinzips zur Überprüfung der Wohlgeformtheit einer aussagenlogischen Formel in Präfixform vorgestellt hatte, damals als ein mechanisches Gerät. Seine Idee wurde dann bis 1957 als Relais-Rechner zur Berechnung aussagenlogischer Formeln realisiert, (Bauer, 1960; Bauer, 1977). Auf Anraten des Münchner Elektroingenieurs Hans Piloty, der das Prinzip für einen Rechner PERM2 nutzen wollte, wurde das Verfahren, übertragen auf arithmetische Ausdrücke, zunächst als Patentschrift veröffentlicht, (Bauer und Samelson, 1957). Bauer und Samelson konstruierten dazu eine Rechenmaschine, die einen Operanden- und einen

Operationskeller eingebaut hatte. Dafür erhielten sie ein deutsches Patent und später auch Patente in anderen Ländern, darunter den USA. Die Idee getrennter Operanden- und Operationskeller lag später auch vielen Implementierungen funktionaler Programmiersprachen zugrunde.

Ab 1957/58 verfolgten Bauer, Bottenbruch, Rutishauser und Samelson dann den Gedanken einer einheitlichen „algebraischen" Programmiersprache, um Algorithmen unabhängig von den Eigenheiten von Rechnern beschreiben zu können. Sie warben dafür um Mitstreiter. Aus den USA kooperierten John Backus, Charles Katz, Alan Perlis und Joseph Henry Wegstein. Das Ergebnis der Arbeit nannten die Amerikaner IAL, *international algebraic language*, in Europa hieß die Sprache ALGOL 58, (Perlis und Samelson, K. (Hrsg.), 1958, 1959). Der Begriff *algorithmic language* und die Abkürzung ALGOL stammen der Überlieferung nach von Herrn Bottenbruch. Die Sprache konnte sich nicht durchsetzen, führte aber in Amerika zu zwei Dialekten Jovial und Neliac, die bis zur Einführung von Ada die Echtzeitprogrammierung vor allem im militärischen Bereich dominierten. Indirekt stammt nach einigen Zwischenschritten auch die Programmiersprache C von Algol 58 ab.

Das Kellerprinzip führte K. Samelson umstandslos zur Erfindung des Codeblocks **begin . . . end** und damit des Prinzips der Blockschachtelung in Algol 58. Damit waren alle Elemente vorhanden, die zur Definition von Algol 60, benötigt wurden.

3 Die 60er Jahre

Algol 60 wurde auf einer Konferenz in Paris im Januar 1960 mit den Teilnehmern Friedrich L. Bauer, Peter Naur, Heinz Rutishauser, Klaus Samelson, Bernard Vauquois, Adriaan van Wijngaarden, Michael Woodger aus Europa und John W. Backus, Julien Green, Charles Katz, John McCarthy, Alan J. Perlis, Joseph Henry Wegstein aus den USA definiert. Peter Naur war der Herausgeber des Berichts, der als (Naur P. (Hrsg.), 1960) veröffentlicht wurde. Die noch heute gültige Sprachnorm ist die Revision (Naur P. (Hrsg.), 1962). Das Urteil von Sir C.A.R. Hoare lautete „Here is a language so far ahead of its time that it was not only an improvement on its predecessors but also on nearly all its successors", Hoare (1973).

Die Teilnehmer der Pariser Konferenz bildeten hernach den Kern der Working Group 2.1 der International Federation for Information Processing, die 1962 gegründet wurde und die weitere Entwicklung von Algol betreute.

Der Sprache fehlte eine Definition der Ein/Ausgabe. Auch verstand sich Algol 60 ausschließlich als Sprache zur Beschreibung numerischer Algorithmen. Kommerzielle Anwendungen, die naturgemäß auch den Umgang mit Geldbeträgen und der gesetzlich vorgeschriebenen dezimalen Rundung vorausgesetzt hätten, sowie Anwendungen zur Manipulation von Texten waren nicht vorgesehen.

Rekursive Prozeduren und Funktionen wurden in Algol 60 erst in letzter Minute auf Wunsch von van Wijngaarden und Edsgar W. Dijkstra zugelassen. Die deutschen Mitglieder lehnten das ab und die ab Juni 1959 aus der ZMMD-Kooperation (Zürich, München, Mainz, Darmstadt) entstandene ALCOR-Gruppe implementierte Algol überwiegend ohne Rekursion, (Baumann, 1961). Die ersten Übersetzer in Deutschland und Österreich wurden für die Zuse Z22 (M. Paul), PERM (G. Seegmüller), ERMETH

(H. R. Schwarz), Mailüfterl (P. Lucas, H. Bekic), Zebra (W. L. van der Poel, van der Mey) , Siemens 2002 (U. Hill-Samelson, H. Langmaack) Ende 1961/Anfang 1962 nach einheitlichen Bauplänen fertiggestellt. Die Methodik für die syntaktische Analyse war eine Verallgemeinerung des Kellerprinzips von Samelson und Bauer. Im Anfang gab es noch keine statische semantische Analyse, sondern es wurde eine weitgehend maschinenunabhängige Zwischensprache erzeugt, die Datentypen wie real und integer interpretativ zur Laufzeit unterschied. Dies war vor allem auch durch die Speicherbeschränkungen verursacht: Keine der Maschinen hatte im Anfang einen Hintergrundspeicher brauchbarer Größe, um ein übersetztes Programm dauerhaft zu speichern. Das Programm mußte also unmittelbar nach der Übersetzung ausgeführt werden.

Die Grundzüge dieser Vorgehensweise wurden in Samelson und Bauer (1959) veröffentlicht. Der Artikel war so bedeutsam, daß ihn die Communications ACM, (Samelson und Bauer, 1960), in englischer Übersetzung nachdruckten. Er war für viele Jahre einer der internationalen Eckpfeiler des Übersetzerbaus.

Ab 1962 begann eine zweite Phase der ALCOR-Kooperation. Die Rechner waren etwas schneller und größer geworden und man konnte nun bequem volles Algol 60 effizient implementieren. Dies führte zu Seegmüllers Algol 60-Übersetzer für den Telefunken TR4 von 1962 und dem Übersetzer für die Siemens 2002, die bereits lineare Adreßfortschaltung, den praktisch wichtigsten Aspekt von *strength reduction* also einer Optimierung enthielten, Hill et al. (1962); Grau et al. (1967).

Eine weitere, lange Zeit wenig beachtete Leistung war die Erfindung des *post mortem Speicherabzugs*, (Seegmüller, 1962; Bayer et al., 1967). Er lieferte nach einem Programmabsturz eine komplette Darstellung des Programmspeichers, gegliedert entsprechend den gerade aufgerufenen Prozeduren, mit den Namen der vorhandenen Variablen und den Variablenwerten gemäß dem Variablentyp; das galt auch für Reihungen. In einer Zeit, wo man oft einen Tag warten mußte, um das Ergebnis eines Programmlaufs vom Rechenzentrum zu erhalten, war dies eine bedeutende Erleichterung beim Testen. Ein österreichischer Kollege schrieb damals, daß Maschinensprache besser sei als die Benutzung von Cobol, da man dann wenigstens den (sedezimalen) Speicherabzug nachvollziehen könne. Auch heute noch wäre ein gegliederter Speicherabzug oft besser als die Benutzung des gdb.

Die Arbeit (Eickel et al., 1963) faßt die Grundlagen des Syntaxanalyseverfahrens für kontextfreie Sprachen mit beschränktem Kontext zusammen, das die ALCOR-Gruppe verwendete. Ausgangspunkt war die Dissertation (Paul, 1962). (m, k)-beschränkte Grammatiken sind eine echte Teilmenge der $LR(k)$-Grammatiken. Im Vergleich sieht man für $k = 1$ keine wesentlichen Unterschiede zu den später gebräuchlichen LALR(1)-Analysatoren. Daß man damit sämtliche deterministische kontextfreie Grammatiken erfassen kann, wurde erst später erkannt.

F.L. Bauer erklärte 1965 die Forschung im Bereich Übersetzerbau für beendigt, das Weitere sei Sache der Industrie. Das war zweifellos eine Fehlprognose, deren Ursache die damals weit verbreitete Meinung war, daß semantische Analyse, Codeoptimierung und Codeerzeugung ganz einfache Angelegenheiten seien, die keiner weiteren Aufarbeitung bedürfen. Dies wurde danach sehr schnell widerlegt, z.B. durch die Typenvielfalt der ersten objektorientierten Programmiersprache Simula 67, (Dahl et al., 1968), und die zunehmende Komplexität der Maschinensprachen.

Wie wenig Industrie und Hochschulen in Deutschland in den 60er Jahren zusammenwirkten, konnte man 1967 in München erleben, als IBM Software-Spezialisten schickte, die den Mitarbeitern der TU die Vorzüge von

Algol 60 und die Verfügbarkeit dieser Sprache auf der IBM 360/81 darstellen sollten. Die Herren ahnten nicht, daß sie wesentliche Erfinder dieser Sprache vor sich sitzen hatten. Das Verhältnis änderte sich erst in den 70er Jahren.

Mit der Definition von LISP hatte J. McCarthy, (McCarthy, 1960), die Idee geboren, die Semantik von Programmiersprachen durch einen formalen Interpretierer zu beschreiben. Heinz Zemanek, der Konstrukteur des Rechners Mailüfterl an der TU Wien, wechselte nach seiner Assistentenzeit zur Firma IBM, übernahm die Leitung des neu gegründeten IBM-Labors Wien und griff den Gedanken auf, um zusammen mit einer großen Zahl von Mitarbeitern eine formale Definition der Programmiersprache PL/1 zu erreichen. Der Grundgedanke dieser als *Vienna Definition Language* (VDL) bekannten Methode besteht darin, sowohl ein Programm als auch seine Daten als Bäume aufzufassen. Die wesentliche Operation lautet $\mu(x; s : y)$, sie liefert das Objekt x, in dem das durch den Selektor s ausgewählte Teilobjekt durch y ersetzt ist. Gibt es in x keinen Selektor s, so wird $s : y$ neu hinzugefügt, bei $y = \Omega$ wird $s : y$ gestrichen. Mit der μ-Operation lassen sich Operationen in Datenräumen, aber auch der Fortschritt der Programmausführung beschreiben: ein Prozeduraufruf ersetzt den Prozedurnamen durch seinen Rumpf; das Prozedurende macht das rückgängig. Das Verfahren ist beschrieben in (Lucas und Walk, 1971) und (Wegner, 1972).

Auf der Basis von VDL entwickelte sich dann die *Vienna Development Method* (VDM), (Björner und Jones, 1978).

4 Die 70er Jahre

Zu Anfang der 70er Jahre war aus dem deutschsprachigen Raum vor allem die ETH Zürich international sichtbar. Niklaus wirth, der 1968 aus Berkeley nach Zürich gekommen war, definierte die Programmiersprache Pascal, (Wirth, 1971), nachdem er zuvor schon die Sprachen Euler, Algol W und die maschinennahe Sprache PL360 geschaffen hatte. Pascal erweiterte Algol um die *record*-Datentypen, *files*, sowie die *case*-Anweisung, die in einfacherer Form schon in Cobol vorhanden waren. Pascal wurde in Zürich implementiert, wobei Wirths Mitarbeiter als Zwischensprache den P-Code definierten, (Nori et al., 1976). Der P-Code ist die Urform der virtuellen Maschinensprachen wie der JVM oder .NET. Ein Interpretierer für den P-Code umfaßte nur wenige 100 Zeilen Maschinensprache; da auch der Übersetzer selbst in P-Code vorlag, konnte man also mit geringem Programmieraufwand ein zwar langsames, aber einwandfrei funktionierendes Pascal-System herstellen. Dies sorgte für die rasche, weltweite Verbreitung der Sprache.

Die Arbeit (Hoare und Wirth, 1973) definierte den Kern der Semantik von Pascal mit Hilfe der Hoare-Logik, zwar ebenfalls operativ, aber anders als VDL. Diese Methodik wurde von vielen weiteren Autoren aufgegriffen, vgl. z.B. (Loeckx et al., 1986).

Die Übersetzerbauer brachten derweil die Arbeiten zum Thema Syntaxanalyse zum Abschluß, z.B. in Form des PGS-Systems von Peter Dencker, einer Karlsruher Diplomarbeit aus dem Jahre 1977, (Dencker et al., 1984), das als Teil des Eli-Systems, (Gray et al., 1992), heute noch in Gebrauch ist.

D. Knuth führte mit der Arbeit (Knuth, 1968, 1971) attributierte Grammatiken (AG) als allgemeine Methodik zur Beschreibung der statischen Semantik von kontextfreien Sprachen ein. Bei Verwendung von rekursivem Abstieg für die Syntaxanalyse erweisen sich

AGs als abstrakte Beschreibungstechnik für bereits seit langem eingesetzte Verfahren für die semantische Analyse.

C.H.A. „Kees“ Koster, einer der Koautoren von Algol 68 und 1972 - 76 Professor an der TU Berlin (später an der Universität Nijmwegen), entwickelte in den frühen 70er Jahren die sogenannten *Affix-Grammatiken*, eine Spezialisierung der van Wijngaarden-Grammatiken, bei der Teile der Nichtterminale von kontextfreien Produktionen als Prozedurparameter gewertet wurden, die bei der Berechnung, z.B. mit rekursivem Abstieg, mit Werten versehen werden konnten. Affix-Grammartiken und das damit konstruierte System CDL zur Konstruktion von Übersetzern war in Deutschland, Belgien, den Niederlanden und Ungarn verbreitet im Einsatz.

Die Erkenntnis, in dieser Notation auch sehr systematisch programmieren zu können, führte zur Erweiterung der Sprache zu CDL2, und zur Entwicklung einer auf CDL basierenden Programmierumgebung. Eingeführt wurde mit CDL2 ein statisches Modulkonzept zur Realisierung eines Schichtenmodells von Software-Architekturen und eine regulierte Kommunikationsstruktur sowohl zwischen Schichten und den Komponenten einer Schicht. Für die unterste Ebene, die so genannte Basisschicht wurden vordefinierte Pakete mit Basisoperationen (z.B. Arithmetik) zur Verfügung gestellt.

Herr Koster definierte dann die Programmiersprache ELAN, (Hommel et al., 1979), die mit CDL2 implementiert wurde. ELAN wurde als Sprache für den Programmierunterricht entwickelt. Zwei wesentliche als Sprachkonstrukte umgesetzte Prinzipien der so genannten strukturierten Programmierung waren die schrittweise Verfeinerung und das Geheimnisprinzip beim Zugriff auf Datenstrukturen.

Das Konzept der schrittweise Verfeinerung wurde als eine Erweiterung des ALGOL60-Blockkonzepts umgesetzt. Blöcke wurden benannt und konnten so aufgerufen werden - allerdings ohne Parameter und umsetzbar durch textuelles Kopieren.

Primär zur Umsetzung des Geheimnisprinzips wurde für die Definition abstrakter Datenstrukturen ein statisches Modulkonzept (Pakete) eingeführt, mit dem Datenstrukturen und Zugriffsoperationen gekapselt werden konnten und der Zugriff auf die Datenstruktur nur mit den definierten Zugriffsoperationen möglich war.

Jochen Liedtke implementierte 1978 Elan für den Prozessor Z80 als Diplomarbeit an der Universität Bielefeld, wobei er die Sprache um (rekursive) Prozesse und Datenräume erweiterte.

ELAN war ähnlich Pascal als Programmiersprache für Schulen empfohlen und wurde zu diesem Einsatzzweck auch von der GMD in Birlinghoven unterstützt. In diesem Zusammenhang entwickelte Jochen Liedtke in einer Erweiterung von ELAN ein Betriebssystem Eumel, ein Mehrbenutzersystem mit virtuellem Speicher auf dem Z80. Das System wurde in Schulen und in Einzelbüros, z.B. bei Rechtsanwälten, viele 1000 Male eingesetzt, auch im Ausland, vor allem in Japan. Eumel wurde auf den Intel 8086 übertragen und hieß dann L3.

Bei einer Demonstration von L3 in Tokio 1985 auf einem Siemens PC konnte Siemens Japan in der Eile keinen Transformator für 100V auftreiben und stellte nur einen Trafo für 110V zur Verfügung. Das Resultat war, daß das System wegen zu geringer Spannung immer wieder abstürzte. Nun speicherte Eumel/L3 seine Daten regelmäßig auf dem Sekundärspeicher; hier geschah das alle 5 Minuten. Beim Neustart hatte das System daher nur die Arbeit der letzten 5 Minuten verloren. Während sich die Deutschen über die Abstürze ärgerten, hatten die Japaner den Eindruck, die Abstürze seien Absicht, um die Erhaltung der Speicherinhalte zu demonstrieren.

Das Nachfolgesystem L4, ein μKern-System, erreichte auf der SOSP 1993 Weltruf,

als Liedtke nachwies, daß seine Implementierung des Fernaufrufs, (*remote procedure call*), 20 mal schneller war als die Implementierung im CMU μKern Mach, (Liedtke, 1993). Die amerikanischen Kollegen hielten das für unglaublich, bis es ihnen demonstriert wurde. Gernot Heiser und seine Mitarbeiter an der *University of New South Wales* bewiesen die Korrektheit von L4, (Klein et al., 2010). Es wird heute millionenfach eingesetzt, u.a. in *smartphones*.

5 Die 80er Jahre

Uwe Kastens konnte in seiner Dissertation 1976, (Kastens, 1980), an der Universität Karlsruhe *geordnete attributierte Grammatiken* (OAG) definieren, die sich als ausreichend für die Beschreibung der semantischen Analyse von beliebigen Programmiersprachen erwiesen. Geordnete attributierte Grammatiken sind ein polynomiell berechenbarer Spezialfall von partitionierten attributierten Grammatiken, (Waite und Goos, 1984), einer Eigenschaft, die im allgemeinen nur mit Aufwand NP festgestellt werden kann. Die Eigenschaft garantiert, daß man programmunabhängig die Reihenfolge festlegen kann, in der die Attribute eines abstrakten Syntaxbaums berechnet werden können. Mit dem GAG-System, (Kastens et al., 1982), implementierte Kastens diese Reihenfolgebestimmung für OAGs samt der semantischen Analyse für Anwendungsprogramme. Eine weiter entwickelte Form von GAG, das Liga-System, ist Teil des bereits zitierten Eli-Systems, (Gray et al., 1992).

Damit waren die Karlsruher in der Lage die semantische Analyse von Ada komplett zu spezifizieren und zu testen, bevor sie eine effiziente Implementierung vornahmen, (Uhl et al., 1982). Dies verschaffte den Karlsruhern einen großen zeitlichen Vorsprung bei ihrer Implementierung von Ada 83. Sie steuerten überdies einen großen Teil der inhaltlichen Spezifikation der Zwischensprache Diana bei, einem *de facto* Standard für die interne Darstellung von Ada 83-Programmen, (Goos und Wulf, 1983).

6 Die 90er Jahre und danach

Reinhard Wilhelm und seine Mitarbeiter an der Universität des Saarlandes brachten die statische Programmanalyse in mehreren Punkten weiter: Zusammen mit Mooly Sagiv, Tel Aviv, und Thomas Reps, Madison, entwickelte Herr Wilhelm eine statische Programmanalyse auf der Grundlage einer 3-wertigen Logik, (Sagiv et al., 2002), die sogenannte *shape*-Analyse. Der Ausgangspunkt dieser Entwicklung war der Versuch, herauszufinden, wie Programme verzeigerte Datenstrukturen auf der Halde manipulieren. Eine solche Aussage könnte sein, daß ein Programm dem man eine doppelt verkettete Liste mit einem Kopfzeiger und ohne weitere Zeiger gibt, eine solche Liste zurückgibt. Die besonderen Probleme dabei sind, daß solche Datenstrukturen aus anonymen Objekten bestehen und daß die Belegung der Halde prinzipiell unendlich wachsen kann. Mithilfe der entwickelten kanonischen Abstraktion kann jeder Haldeninhalt beliebiger Größe auf einen abstrakten Inhalt beschränkter Größe abgebildet werden. Es zeigte sich, daß die kanonische Abstraktion noch weit mehr schwierige Programmeigenschaften approximativ heraus finden kann, z.B. Synchronisationseigenschaften nebenläufiger Programme.

Ab Ende der 90er Jahre entwickelte die Gruppe von Reinhard Wilhelm die erste Lösung des Problems, die Echtzeiteigenschaften von eingebetteten Programmen und komplexen

Prozessorarchitekturen nachzuweisen, (Ferdinand et al., 2001). Wesentliche Bestandteile der Technologie sind mehrere statische Programmanalysen, also Techniken aus dem Übersetzerbau. Vollkommen neu sind die statischen Analysen, welche an allen Programmpunkten Invarianten über die Menge der dort möglichen Ausführungszustände berechnen, z.B. die Menge der dort möglichen Cacheinhalte. Mithilfe solcher Invarianten lassen sich die bei modernen Prozessoren extrem hohen Schwankungen der Ausführungszeiten einschränken. Die Firmenausgründung AbsInt industrialisierte diese Entwicklung und bietet bis heute die einzigen industriell weithin eingesetzten Werkzeuge zur Echtzeitverifikation an.

7 Danksagung

Die Herren Stefan Jähnichen, Hans Langmaack, Jacques Loeckx und Reinhard Wilhelm haben durch ihre Beiträge und Kritik erheblich zu diesem Papier beigetragen.

Literatur

Bauer, F. (1960): The Formula-controlled Logical Computer "Stanislaus". *Math. Comput.*, 14: 64–67.

Bauer, F. L. (1977): Angstl's Mechanism for Checking Wellformedness of Parenthesis-Free Formulae. *Math. Comp.*, 31(137): 318–320.

Bauer, F. L. (1994): Die ALGOL-Verschwörung. Techn. Ber. 9409, CAU Kiel.

Bauer, F. L. und Samelson, K. (1957): *Patentschrift: Verfahren zur automatisierten Verarbeitung von kodierten Daten und Rechenmaschine zur Ausübung des Verfahrens*. Deutsches Patentamt. Patent 1 094 019.

Bauer, F. L. und Wössner, H. (1972): Zuses „Plankalkül", ein Vorläufer der Programmiersprachen — gesehen vom Jahre 1972. *Elektronische Rechenanlagen*, 14(3): 111–118.

Baumann, R. (1961): ALGOL-*Manual der* ALCOR-*Gruppe*. Oldenbourgh-Verlag, München.

Bayer, R., Gries, D., Paul, M. und Wiehle, H. R. (1967): The ALCOR Illinois 7090/7094 Post Mortem Dump. *Commun. ACM*, 10(12): 804–808.

Björner, D. und Jones, C. B. (1978): *The Vienna Development Method: The Meta-Language*, Bd. 61 von *Lecture Notes in Computer Science*. Springer.

Dahl, O. J., Myhrhaug, B. und Nygaard, K. (1968): *Simula 67, Common Base Language*. Norwegisches Rechenzentrum, Oslo.

Dencker, P., Dürre, K. und Heuft, J. (1984): Optimization of Parser Tables for Portable Compilers. *ACM Transactions on Programming Languages and Systems*, 6(4): 546–572.

Eickel, J., Paul, M., Bauer, F. L. und Samelson, K. (1963): A syntax controlled generator of formal language processors. *Commun. ACM*, 6(8): 451–455.

Ferdinand, C., Heckmann, R., Langenbach, M. et al. (2001): Reliable and Precise WCET Determination for a Real-Life Processor. In *Embedded Software, First International Workshop, EMSOFT 2001, Tahoe City, CA, USA, October, 8-10, 2001, Proceedings*, herausgegeben von Henzinger, T. A. und Kirsch, C. M., Bd. 2211 von *Lecture Notes in Computer Science*, S. 469–485. Springer. ISBN 3-540-42673-6.

Gödel, K. (1931): Über Formal Unentscheidbare Sätze Der Principia Mathematica Und Verwandter Systeme I. *Mh. Math. Phys.*, 38: 173–198.

Goos, G. und Wulf, W. A. (Hrsg.) (1983): *Diana, an Intermediate Language for Ada*, Bd. 161 von *LNCS*. Springer.

Grau, A., Hill, U. und Langmaack, H. (1967): *Translation of Algol 60*, Bd. 1b von *Handbook for automatic computation*. Springer, Heidelberg.

Gray, R. W., Heuring, V. P., Levi, S. P. et al. (1992): Eli: A Complete, Flexible Compiler Construction System. *Comm. ACM*, 35(2): 121–131. Eli is available at http://www.cs.colorado.edu/ēliuser/ or http://www.uni-paderborn.de/project-hp/eli.html.

Hill, U., Langmaack, H., Schwarz, H. und Seegmüller, G. (1962): Efficient handling of subscripted variables in ALGOL 60 compilers. In *Proc. 1962 Rome Symposium on Symbolic Languages in Data Processing*, S. 311–340, New York. Gordon and Breach.

Hoare, C. (1973): Hints on Programming Language Design. In *Symposium on Principles of Programming Languages*. ACM. reprinted in (Horowitz, 1983, pp. 31 – 40).

Hoare, C. A. R. und Wirth, N. (1973): An Axiomatic Definition of the Programming Language Pascal. *Acta Informatica*, 2: 335–355.

Hommel, G., Jäckel, J., Jähnichen, S. et al. (1979): *ELAN-Sprachbeschreibung*. Akademische Verlagsgesellschaft, Wiesbaden.

Horowitz, E. (Hrsg.) (1983): *Programming Languages — A Grand Tour*. Springer.

IEEE 754 (2008): IEEE Standard for binary floating-point arithmetic. Techn. Ber., ANSI/IEEE. Std. 754 - 2008.

Kastens, U. (1980): Ordered Attribute Grammars. *Acta Informatica*, 13(3): 229–256.

Kastens, U., Hutt, B. und Zimmermann, E. (1982): *GAG: A Practical Compiler Generator*. Nr. 141 in Lecture Notes in Computer Science. Springer Verlag.

Klein, G., Andronick, J., Elphinstone, K. et al. (2010): seL4: Formal Verification of an Operating-System Kernel. *Commun. ACM*, 53(6): 107–115.

Knuth, D. E. (1968): Semantics of Context-Free Languages. *Mathematical Systems Theory*, 2(2): 127–146. The invention of AGs.

Knuth, D. E. (1971): Semantics of Context-Free Languages: Correction. *Mathematical Systems Theory*, 5: 95–96. The invention of AGs.

Liedtke, J. (1993): Improving IPC by Kernel Design. In *14th ACM Synposium on Operating System Principles*, Asheville, North Carolina. ACM.

Loeckx, J., Mehlhorn, K. und Wilhelm, R. (1986): *Grundlagen der Programmiersprachen*. Teubner, Stuttgart.

Lucas, P. und Walk, K. (1971): On the Formal Description of PL/I. *Annual Review of Automatic Programming*, 6: 105–182.

McCarthy, J. (1960): Recursive Functions of Symbolic Expressions and their Computation byMachine. *Communications of the ACM*, 3(4): 184 – 195.

Naur P. (Hrsg.) (1960): Report on the Algorithmic Language Algol 60. *Numer. Math.*, 2: 106–136. und Comm. ACM 3(1960), 299-314.

Naur P. (Hrsg.) (1962): Revised Report on the Algorithmic Language Algol 60. *Numer. Math.*, 4: 420–453. und Comm. ACM 6(1963), 1-17.

Nori, K. V., Ammann, U., Jensen, K. et al. (1976): The PASCAL <P> Compiler: Implementation Notes. Techn. Ber., Institut für Informatik, ETH Zürich.

Paul, M. (1962): *Zur Struktur formaler Sprachen*. Dissertation, Universität Mainz.

Perlis, A. J. und Samelson, K. (Hrsg.) (1958): Preliminary report: International algebraic Language. *CACM*, 1(12): 8–22.

Perlis, A. J. und Samelson, K. (Hrsg.) (1959): Report on the Algorithmic Language ALGOL. *Numerische Mathematik*, 1: 41–60.

Rutishauser, H. (1951): Über automatische Rechenplanfertigung bei programmgesteuerten Rechenmaschinen. *Z. angew. Math. Mech.*, 31: 255ff.

Sagiv, M., Reps, T. und Wilhelm, R. (2002): Parametric Shape Analysis via 3-Valued Logic. *ACM Transactions on Programming Languages and Systems*, 24(3): 217–298.

Samelson, K. (1957): Probleme der Programmierungstechnik. In *Internationales Kolloquium über Probleme der Rechentechnik 1955*, S. 61–68, Berlin.

Samelson, K. und Bauer, F. L. (1959): Sequentielle Formelübersetzung. *Elektron. Rechenanlagen*, 1: 176–182.

Samelson, K. und Bauer, F. L. (1960): Sequential Formula Translation. *Commun. ACM*, 3(2): 76–83.

Seegmüller, G. (1962): Some Remarks on the Computer as a Source Language Machine. In *Proc. IFIP Congress*, Bd. 62, S. 524–525.

Uhl, J., Drossopoulou, S., Persch, G. et al. (1982): *An Attribute Grammar for the Semantic Analysis of Ada*, Bd. 139 von *LNCS*. Springer.

Waite, W. M. und Goos, G. (1984): *Compiler Construction*. Springer Verlag.

Wegner, P. (1972): The Vienna Definition Language. *Comp. Surveys*, 4(1): 5–63.

Wilkes, M., Wheeler, D. J. und Gill, S. (1951): *The Preparation of Programs for an Electronic Digital Computer*. Addison-Wesley.

Wirth, N. (1971): The Programming Language Pascal. *Acta Informatica*, 1: 35–63.

Zuse, K. (1972): Der Plankalkül. Bericht Nr. 63, GMD, St. Augustin.