GI-Edition

Lecture Notes
in Informatics

Michael Fothe, Thomas Wilke (Hrsg.)

Keller, Stack und automatisches Gedächtnis – eine Struktur mit Potenzial

Tagungsband zum Kolloquium
14. November 2014 in Jena

Thematics

7

Fothe, Wilke (Hrsg.): Keller, Stack und automatisches Gedächtnis

Michael Fothe, Thomas Wilke (Hrsg.)

Keller, Stack und automatisches Gedächtnis – eine Struktur mit Potenzial

Kolloquium

14. November 2014
in Jena, Deutschland

Gesellschaft für Informatik e.V. (GI)

Lecture Notes in Informatics (LNI) - Thematics
Series of the Gesellschaft für Informatik (GI)

Volume T-7

ISBN 978-3-88579-426-4

Volume Editors
Prof. Dr. Michael Fothe
Universität Jena, Didaktik der Informatik und Mathematik, Ernst-Abbe-Platz 2, 07743 Jena, Email: michael.fothe@uni-jena.de
Prof. Dr. Thomas Wilke
Universität Kiel, Theoretische Informatik, Christian-Albrechts-Platz 4, 24118 Kiel, Email: thomas.wilke@email.uni-kiel.de

printed by Köllen Druck+Verlag GmbH, Bonn

Vorwort

Dieser Band befasst sich mit dem *Keller* (auch *Kellerspeicher*, *Stapel*, *Stapelspeicher* und *stack* genannt) – einer wichtigen Struktur der Informatik. Wiedergegeben werden die Beiträge zu einem wissenschaftlichen Kolloquium, das am 14. November 2014 vom ersten Herausgeber in Jena organisiert wurde und hochkarätig besetzt war: Unter den Vortragenden befanden sich ein Fellow der Gesellschaft für Informatik (GI), ein Gründungsmitglied der GI und ein Empfänger der höchsten Auszeichnung der GI, der Konrad-Zuse-Medaille. Die Struktur „Keller" ist facettenreich, was auch daran kenntlich wird, dass die Autoren ganz unterschiedlichen Gliederungen der GI zuzuordnen sind. Der zweite Herausgeber ist Sprecher des Fachbereichs „Grundlagen der Informatik" der GI.

Die Beiträge berichten über den Platz des Kellers in der universitären Lehre und über Bemerkenswertes aus seiner Geschichte. Sie betrachten ihn aus theoretischer und aus praktischer Sicht. Des Weiteren wird auf Kellerstrukturen in der aktuellen Forschung genauer eingegangen.

Die Erfinder des Kellers sind Friedrich L. Bauer und Klaus Samelson. Bauer wurde für die Erfindung des *computer stacks* im Jahr 1988 der IEEE Computer Pioneer Award zuerkannt; Samelson war bereits 1980 gestorben. In den 1950er-Jahren gab es auch andere, die sich der Struktur näherten, wie der Australier Charles Leonard Hamblin oder Wilhelm Kämmerer. Des Weiteren gab es Vorarbeiten wie die von N. Joachim Lehmann für die Konstruktion des frühen Dresdner Computers D1.

Birgitta König-Ries, Dekanin der Fakultät für Mathematik und Informatik, fragte in ihrem Grußwort: Warum gerade jetzt ein solches Kolloquium und warum gerade in Jena? Für das *gerade jetzt* lassen sich keine wirklich überzeugenden Gründe nennen: Zwar wurde Herr Friedrich L. Bauer im vergangenen Jahr 90 Jahre alt. Aber das war es dann auch mit den einigermaßen runden Zahlen. Vielleicht braucht es aber auch gar keinen Jahrestag und kein Jubiläum, um sich eine solch fundamentale Struktur wie den Keller wieder etwas genauer anzusehen. Die Frage nach dem *warum gerade in Jena* ist einfacher zu beantworten. In dem Kolloquium ging es auch um einen fast in Vergessenheit geratenen Vordenker in Sachen Keller, der in Jena wirkte: den bereits erwähnten Wilhelm Kämmerer.

Wilhelm Kämmerer wurde 1905 in Hessen geboren, studierte in Göttingen und Gießen Mathematik und Physik, promovierte in Gießen und war anschließend zunächst Gymnasiallehrer in Naumburg. Ab 1943 war Kämmerer bei Zeiss in Jena tätig. An seinem dort erworbenen Knowhow zu Steuer- und Zieleinrichtungen von Waffen im Speziellen und der Zeiss-Optik im Allgemeinen war die Sowjetunion nach Kriegsende stark interessiert, was ihm und seinen Kollegen Herbert Kortum und Fritz Straube einen mehrjährigen Zwangsaufenthalt in der Region Moskau einbrachte. In den letzten anderthalb Jahren dieses Aufenthalts begannen die drei intensiv über den Bau einer automatischen Rechenanlage nachzudenken, sodass Kämmerer, Kortum und Straube, als sie 1953 zurück nach Jena und zurück zu Zeiss kamen, im Grunde die Pläne für eine Optikrechenmaschine in der Tasche hatten. Die Oprema wurde 1954/55 als einer der ersten Computer Deutschlands fertiggestellt. Sie war frei programmierbar und diente vor allem (aber nicht nur) zur Berechnung von optischen Systemen. Im Jahr 1958 habilitierte sich Wilhelm Kämmerer an der Universität

Jena mit einer Schrift, in der er ein kellerähnliches „automatisches Gedächtnis“ beschrieb. Nachzutragen ist, dass Kämmerer von 1960 bis 1970 eine Professur für Kybernetik an der Universität Jena inne hatte und dass er 1991 für seine Arbeiten die Konrad-Zuse-Medaille erhielt. Kämmerer verstarb 1994 in Jena.

Allen Autoren sei herzlich für ihre Bereitschaft zum Vortrag und für ihren schriftlichen Tagungsbeitrag gedankt, Christopher Schneider für die technische Unterstützung bei der Herausgabe des Bandes.

Michael Fothe, Thomas Wilke

Nachtrag: Friedrich L. Bauer verstarb am 26. März 2015, kurz vor der Fertigstellung dieses Bandes, der inhaltlich sehr mit seinen frühen Arbeiten verbunden ist. Wir trauern um ihn.

Inhaltsverzeichnis

Der Keller – ein fundamentaler Baustein der Informatik

Martin Mundhenk

Friedrich-Schiller-Universität Jena
Fakultät für Mathematik und Informatik
Ernst-Abbe-Platz 2
07743 Jena
martin.mundhenk@uni-jena.de

Kurzfassung: Die Datenstruktur Keller begegnet Informatik-Studierenden gleich am Beginn des Studiums als eine fundamentale informatische Denkweise. Wir machen einen Ausflug zu grundlegenden Algorithmen und sehen, wie ein Keller beim Auswerten von Formeln oder beim schnellen Potenzieren hilft. Neben einzelnen Algorithmen gibt es auch algorithmische Prinzipien, die auf dem Keller basieren. Das wird am Beispiel des Teile-und-herrsche-Prinzips verdeutlicht.

1 Der Keller

Ein *Keller* ist ein Speicher für beliebig viele Elemente. Wenn man ein neues Element speichern will, dann legt man es oben in den Keller. Wie bei einem Kisten-Stapel kann man nur sehen, was in der obersten Kiste drin ist. D.h. wenn man in den Keller schaut, sieht man nur das zuletzt gespeicherte Element. Dieses sichtbare Element ist auch das, was man aus dem Keller entfernen kann. Wurde es entfernt, ist das zuvor darunter liegende Element sichtbar. Die Abbildung zeigt die Wirkungen einer Folge von Keller-Operationen auf einem anfangs leeren Keller (a). Wird 5 in den Keller eingefügt, ergibt sich Situation (b). Fall (c) zeigt den Keller nach Einfügen der 13. Wird nun das oberste Element gelesen, ergibt das 13. Durch Einfügen der 4 ergibt sich Situation (d). Fall (e) zeigt den Keller, nachdem das oberste Element gelöscht wurde.

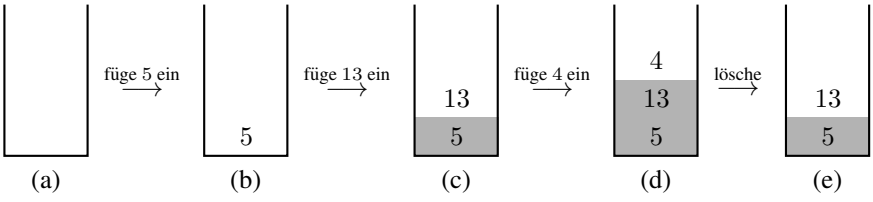


Abbildung 1: Ein Keller und seine Veränderung

Einen Keller benutzt man mit folgenden Keller-Operationen:

- füge x ein (d.h. lege x oben auf den Keller),

- lösche (d.h. entferne das oberste Element des Kellers),
- lies (das oberste Element des Kellers),
- stelle fest, ob der Keller leer ist.

Die Arbeitsweise eines Kellers wird als LIFO (*last in, first out*) bezeichnet.

2 Das Auswerten arithmetischer Ausdrücke

Arithmetische Ausdrücke kennt man bereits aus der Schule als Rechenaufgaben wie z.B.

$$(3 + 4) \cdot 5 - (8 - 2 \cdot 3 + 7)\,.$$

Wenn man sie ausrechnen will, muss man die Klammerung und die unterschiedliche Bindungsstärke der arithmetischen Operatoren – „Punktrechnung geht vor Strichrechnung“ – beachten. Es gibt verschiedene Schreibweisen für arithmetische Ausdrücke. Eine Schreibweise, bei der man sich weder um Klammerung noch um Bindungsstärke der Operatoren kümmern muss, ist die *Präfixnotation*[1]. Dabei schreibt man z.B. für $3 + 4$ den Operator $+$ wie ein Funktionssymbol vor die beiden Operanden 3 und 4, also $+(3, 4)$. Für

$$(3 + 2 \times 4 - 1) \times (7 - 5)$$

schreibt man durch Vorziehen der Operatoren zunächst

$$\times(-(+(3, \times(2, 4)), 1), -(7, 5))\,.$$

Da wir hier nur zweistellige Operatoren verwenden (das „$-$“ als negatives Vorzeichen lassen wir mal weg), kann man Klammern und Kommas weglassen, ohne die Eindeutigkeit des Ausdrucks zu verlieren. In diesem Beispiel erhalten wir dann

$$\times\ -\ +\ 3\ \ \times\ 2\ 4\ 1\ -\ 7\ 5\,.$$

Diese Schreibweise nennt man *Präfixnotation*. Die übliche Schreibweise heißt auch *Infixnotation*, da der Operator stets *zwischen* den Operanden steht. Bei der Infixnotation kann man nicht auf die Klammern verzichten. Das macht das Erkennen der Struktur des Ausdrucks und damit auch seine Auswertung schwieriger.

2.1 Auswertung von Ausdrücken in Präfixnotation

Zum Auswerten eines Ausdrucks in Präfixnotation liest man ihn von rechts nach links! Liest man eine Zahl, dann legt man sie auf den Keller. Da jeder Operator auf die beiden Operanden folgt, wird ein Operator auf die beiden obersten Elemente des Kellers angewendet. Die Operanden waren sozusagen Zwischenergebnisse, die jetzt nicht mehr benötigt werden und aus dem Keller entfernt werden. Das Ergebnis dieses Rechenschrit-

[1] Sie wird auch als „Polnische Notation“ bezeichnet, um an den Logiker Jan Łukasiewicz zu erinnern, der sie um 1920 erfunden hat.

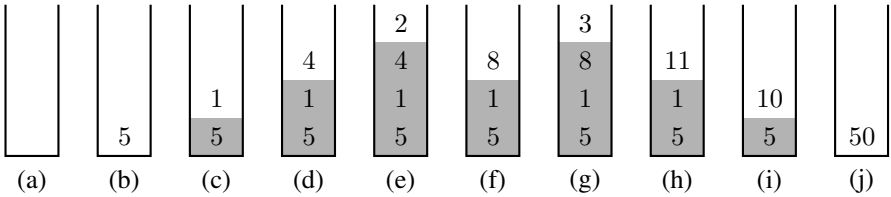


Abbildung 2: Auswertung des Ausdrucks in Präfixnotation $\times - + \ 3 \ \times \ 2 \ 4 \ 1 \ 5$

tes wird auf den Keller gelegt, um so Zwischenergebnisse für weitere Rechenschritte im Keller zu sichern.

Abbildung 2 zeigt in groben Schritten die Auswertung des Ausdrucks $(3+2\times4-1)\times5$ mit der Präfixnotation $\times - + \ 3 \ \times \ 2 \ 4 \ 1 \ 5$. Die Auswertung startet mit dem leeren Keller (a). Anschließend werden von rechts nach links die Zahlen $5, 1, 4, 2$ gelesen und nacheinander auf den Keller gelegt (b)–(e). Im Schritt von (e) zu (f) wird das Multiplikationszeichen $\times$ gelesen. Um es auszuwerten, wird zunächst ein Element vom Keller genommen. Man merkt sich dieses Element als linken Operanden – hier ist es die 2. Anschließend wird noch ein Element vom Keller genommen, das man sich als rechten Operanden merkt – hier ist es die 4. Nun hat man den auszuwertenden Ausdruck 2×4 zusammengestellt. Er wird ausgewertet und das Ergebnis wird im Keller gespeichert – hier ist es die 8. Danach wird die 3 gelesen und im Keller gespeichert (g). Es folgt die Auswertung des $+$ (h), des $-$ (i) und schließlich des $\times$ (j). Am Ende ist der Wert des Ausdrucks das einzige im Keller gespeicherte Element.

2.2 Auswertung üblicher arithmetischer Ausdrücke

Beim Auswerten eines Ausdrucks in Präfixnotation haben wir einen Keller zum Speichern von Zahlen benutzt. Einen solchen Keller werden wir auch zum Auswerten üblicher Ausdrücke benutzen, und wir nennen ihn hier *Zahlenkeller*. Die Operatoren brauchte man nicht zu speichern, da man sie sofort auswerten konnte. Das geht nicht mehr, wenn man einen üblichen arithmetischen Ausdruck auswerten will und sie von links nach rechts liest[2]. Hat man zum Beispiel den Anfang $142+20\cdots$ eines Ausdrucks gelesen, dann weiß man nach dem Lesen der 20 noch nicht, welche Werte von dem gerade gelesenen $+$ addiert werden. Kommt hinter der 20 ein $\times$, dann beginnt mit 20 ein Teilausdruck, dessen Wert berechnet werden muss, bevor er zur 142 addiert wird. Kommt hinter der 20 ein $+$, dann erhält man mit der Auswertung von $142 + 20$ das linke Argument für diese spätere Addition. Im ersten Fall wird das erste $+$ nach dem darauffolgenden Operator ausgewertet, im zweiten Fall davor. Um die Operatoren in der richtigen Reihenfolge auszuwerten, benutzt man einen zweiten Keller – wir nennen ihn hier *Operatorenkeller*.

Zuerst definieren wir als grundlegende Operation das *Auswerten eines Operators*. Es besteht aus folgenden Schritten.

[2]Die Leserichtung spielt hier keine Rolle.

1. Entferne ein Element aus dem Zahlenkeller und nenne es b.
2. Entferne ein Element aus dem Operatorenkeller und nenne es $\otimes$.
3. Entferne ein Element aus dem Zahlenkeller und nenne es a.
4. Berechne $a \otimes b$ und lege es auf den Zahlenkeller.

Die folgende Abbildung zeigt in einem Beispiel, wie sich die beiden Keller bei einer Operatorauswertung verändern. Nach der Operatorauswertung sind also beide Keller eins kleiner geworden.

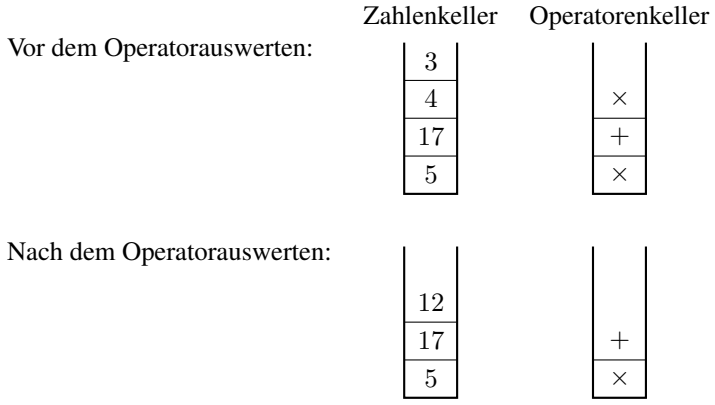


Abbildung 3: Auswertung eines Operators

Zur Auswertung des gesamten Ausdrucks wird dieser von links nach rechts gelesen. Jedes Zeichen X des Ausdrucks (Zahl, Klammer oder Operator) wird gemäß folgender Regeln behandelt. Jeder Operator hat eine *Priorität*. „Punktrechnung geht vor Strichrechnung“ wird dadurch ausgedrückt, dass $\times$ eine höhere Priorität hat als $+$ und $-$. Wir geben $\times$ hier die Priorität 2, und $+$ und $-$ erhalten Priorität 1.

1. Ist X eine Zahl, dann wird X auf den Zahlenkeller gelegt.
2. Ist X ein Operator, dann werden solange Operatoren ausgewertet, bis (1) auf dem Operatorenkeller ein Operator mit kleinerer Priorität als X oder eine öffnende Klammer liegt oder (2) der Operatorenkeller leer ist. Abschließend wird X auf den Operatorenkeller gelegt.
3. Ist X die öffnende Klammer „(“, dann wird X auf den Operatorenkeller gelegt.
4. Ist X die schließende Klammer „)“, dann werden solange Operatoren ausgewertet, bis die öffnende Klammer „(“ auf dem Operatorenkeller liegt. Sie wird dann auch noch vom Operatorenkeller entfernt. (Die schließende Klammer kommt nie in den Keller.)

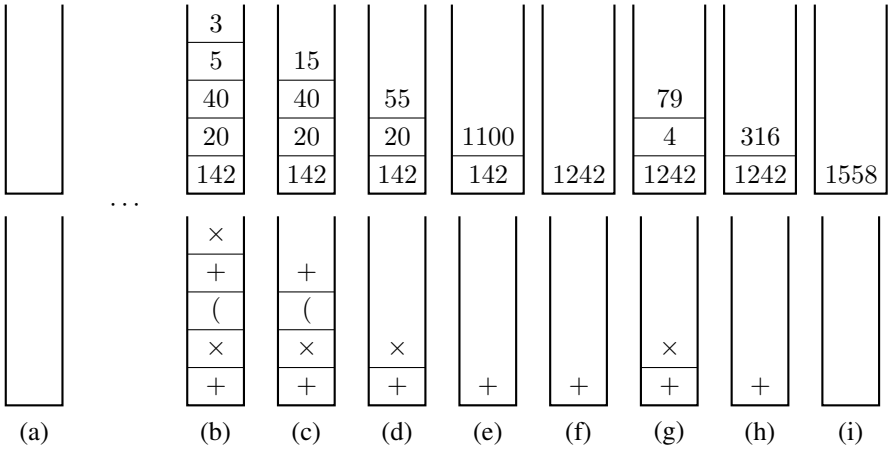


Abbildung 4: Die beiden Keller im Verlauf der Auswertung von $142 + 20 \times (40 + 5 \times 3) + 4 \times 79$

Nachdem der ganze Ausdruck von links nach rechts abgearbeitet wurde, werden abschließend solange Operatoren ausgewertet, bis der Operatorenkeller leer ist. Das Ergebnis steht dann im Zahlenkeller.

Den Algorithmus, der auf diesen vier Regeln beruht, wollen wir uns in einem Beispiel verdeutlichen. Dazu betrachten wir den arithmetischen Ausdruck

$$142 + 20 \times (40 + 5 \times 3) + 4 \times 79\,.$$

Die Kellerinhalte im Verlauf der Auswertung sind in Abbildung 4 dargestellt. Am Anfang sind beide Keller leer (a). Beim Lesen des Anfangsabschnitts $142 + 20 \times (40 + 5 \times 3$, wird jedes gelesene Zeichen gemäß den Regeln auf einen Keller gelegt – jede Zahl auf den Zahlenkeller und jeder Operator auf den Operatorenkeller (b). Beim Lesen der darauffolgenden schließenden Klammer werden solange Operatoren ausgewertet, bis die öffnende Klammer auf dem Operatorenkeller liegt. Sie wird dann auch entfernt (d). Anschließend wird $+$ gelesen. Da auf dem Operatorenkeller ein $\times$ liegt, das höhere Priorität als $+$ hat, wird ein Operator ausgewertet (e). Danach liegt ein $+$ auf dem Operatorenkeller, das die gleiche Priorität wie das gerade gelesene $+$ hat. Also wird wieder ein Operator ausgewertet, und schließlich kommt das zuletzt gelesene $+$ in den Operatorenkeller (f). Danach wird 4×79 gelesen und auf die Keller verteilt (g). Da nun der gesamte Ausdruck gelesen wurde, werden solange Operatoren ausgewertet, bis der Operatorenkeller leer ist. Abschließend steht das Ergebnis 1558 im Zahlenkeller.

3 Berechnung induktiv definierter Funktionen

Die Binärdarstellung[3] einer natürlichen Zahl $n \geqslant 1$ ist die $0/1$-Folge $b_m b_{m-1} \cdots b_1 b_0$, sodass $n = \sum_{i=0}^{m} b_i \cdot 2^i$ und $b_m = 1$. Zum Beispiel hat 19 die Binärdarstellung 10011, da

$$19 = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \,.$$

Beim Betrachten der Binärdarstellung $b_m b_{m-1} \cdots b_1 b_0$ von n erkennt man Folgendes.

- b_0 ist der Rest beim Teilen von n durch 2 – diesen Wert bezeichnet man mit $n \bmod 2$ (b_0 ist 1, falls n ungerade ist, und b_0 ist 0, falls n gerade ist).
- $b_m \cdots b_1$ (also die Binärdarstellung von n ohne b_0) ist die Binärdarstellung der (abgerundeten) Hälfte von n – diesen Wert beschreibt man als $\lfloor \frac{n}{2} \rfloor$.

Beim Beispiel 19 ist das letzte (rechteste) Bit der Binärdarstellung 1 (da 19 ungerade ist), und die übrigen Bits sind 1001 – das ist genau die Binärdarstellung von $1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 9$, der abgerundeten Hälfte von 19.

Diese Beobachtung kann man sich zu Nutze machen, wenn man die Binärdarstellung einer Zahl berechnen will. Das Bit, das man als erstes berechnet, ist jedoch das letzte Bit. Bevor man es ausgeben kann, muss man die Binärdarstellung der (abgerundeten) Hälfte der Zahl ausgeben. Um dieses „Aufschieben“ der Ausgabe hinzubekommen, kann man das berechnete letzte Bit in einen Keller legen und dann mit der Berechnung der Binärdarstellung der (abgerundeten) Hälfte fortfahren. Hat man durch das wiederholte Halbieren schließlich die Zahl 0 erreicht, dann weiß man, dass man mit der Berechnung der einzelnen Bits fertig ist. Man muss sie nur noch ausgeben. Das macht man, indem man die Kellerelemente nacheinander entfernt und ausgibt, bis der Keller leer ist. Etwas formaler aufgeschrieben, sieht der Algorithmus wie folgt aus.

Algorithmus zur Ausgabe der Binärdarstellung von n

Phase 1 (bis $n = 0$):	lege $n \bmod 2$ auf den Keller $n := \lfloor \frac{n}{2} \rfloor$
Phase 2 (bis Keller leer):	lies oberstes Kellerelement b und entferne es Ausgabe b

Abbildung 5 zeigt, wie sich die Kellerinhalte bei der Berechnung der Binärdarstellung von 19 mit diesem Algorithmus verändern. Zu Beginn ist der Keller leer. Während Phase 1 wird der Keller immer größer. Am Ende von Phase 1 steht die Binärdarstellung von 19 im Keller und muss nur noch ausgelesen werden.

Die arithmetischen Operationen $n \bmod 2$ und $\lfloor \frac{n}{2} \rfloor$ sehen kompliziert aus, sind aber ganz elementare Rechenoperationen. In der Rechnerhardware nutzt man Register, in denen

[3] Wir betrachten im Folgenden nur die Binärdarstellungen von Zahlen $n \geqslant 1$, da diese stets mit 1 beginnen.

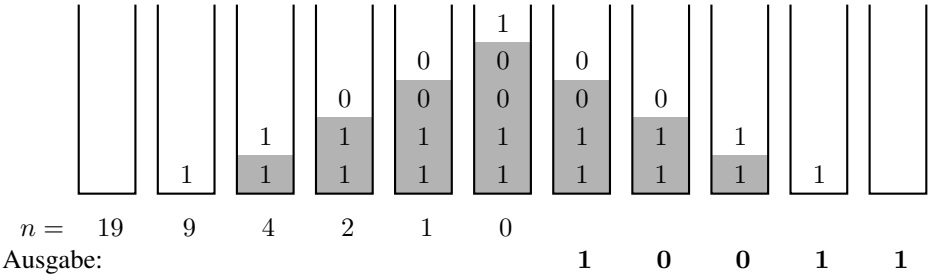


Abbildung 5: Der Verlauf der Kellerinhalte bei der Berechnung der Binärdarstellung von 19

natürliche Zahlen in Binärdarstellung gespeichert werden. Ein elementarer Befehl ist es, alle Bits in einem Register um eine Stelle nach rechts zu schieben (*shift right*). Wenn in einem Register die Zahl n gespeichert ist, steht dort nach einer *shift right* Operation die Zahl $\lfloor \frac{n}{2} \rfloor$. Das Bit, das dabei herausgeschoben wurde, ist $n \bmod 2$ und steht z.B. als *carry* Bit zur Verfügung. In vielen Programmiersprachen gibt es die *shift right* und die entsprechende *shift left* Operation, da man manche Divisionen und Multiplikationen schneller berechnen kann als mit den klassischen arithmetischen Operationen.

Nun ist das Berechnen der Binärdarstellung ein Beispiel, von dem man nicht unbedingt glaubt, dass man es tatsächlich mal gebrauchen kann. Hinter dieser Vorgehensweise steckt jedoch mehr. Dazu schauen wir uns an, wie man eine Potenz a^n berechnen kann. Das naheliegendste Vorgehen ist, n-mal a zu multiplizieren.

$$\underbrace{a \cdot a \cdot \ldots \cdot a}_{n\text{-mal}} \;=\; a^n$$

Dafür benötigt man ungefähr n Multiplikationen. Geht es auch mit deutlich weniger Multiplikationen? Die Antwort ist ja! Wie die Binärdarstellung von n auf die Binärdarstellung von $\lfloor \frac{n}{2} \rfloor$ und $n \bmod 2$ zurückgeführt wurde, können wir auch die Berechnung von a^n auf die Berechnung von $a^{\lfloor \frac{n}{2} \rfloor}$ und ein oder zwei weitere Multiplikationen zurückführen. Betrachten wir zuerst die Berechnung von a^{16}. Gemäß den Potenzregeln ist $a^{16} = a^{8 \cdot 2} = (a^8)^2$. Zur Berechnung von a^{16} braucht man also so viele Multiplikationen wie für a^8 und dann noch eine weitere Multiplikation zum Quadrieren. Damit haben wir gegenüber der naheliegenden Methode bereits 7 Multiplikationen gespart. Entsprechend braucht man zur Berechnung von $a^8 = a^{4 \cdot 2} = (a^4)^2$ die Multiplikationen für a^4 und eine zusätzliche Multiplikation. Zur Berechnung von a^4 braucht man schließlich 2 Multiplikationen. Nutzt man das wiederholte Quadrieren, kann man a^8 mit 4 Multiplikationen ausrechnen. Allgemeiner kann man a^{2^n} mit n Multiplikationen ausrechnen, also zum Beispiel a^{1024} mit 10 Multiplikationen. Das ist schon eine enorme Einsparung. Ist die Potenz keine Zweierpotenz, dann verdoppelt sich die Zahl der Multiplikationen im ungünstigsten Fall. Allgemein gilt für $n \geqslant 1$

$$a^n \;=\; (a^{\lfloor \frac{n}{2} \rfloor})^2 \cdot a^{n \bmod 2}\,.$$

Der verbleibende Fall $a^0 = 1$ (also für $n = 0$) ist einfach. Daraus ergibt sich ein Algorithmus zum Potenzieren, der dem Algorithmus zur Berechnung der Binärdarstellung sehr

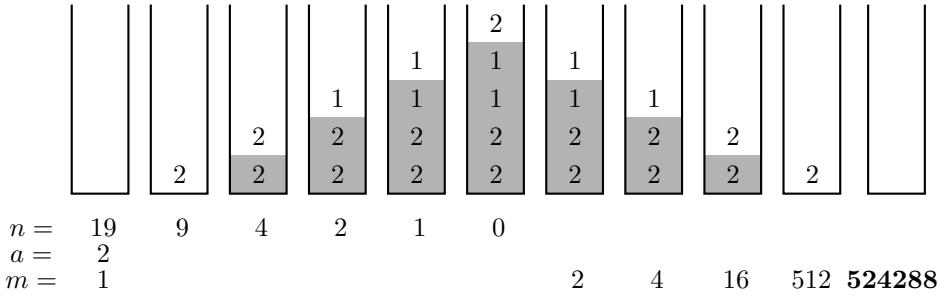


Abbildung 6: Der Verlauf der Kellerinhalte bei der Berechnung von 2^{19}

ähnlich sieht.

Berechne a^n

Phase 1 (bis $n = 0$):	lege $a^{n \bmod 2}$ auf den Keller $n := \lfloor \frac{n}{2} \rfloor$
Phase 2 (bis Keller leer):	lies oberstes Kellerelement b und entferne es $m := m^2 \cdot b$

Das Ergebnis ist der Wert von m, wenn der Algorithmus endet. Die Anzahl der Elemente, die in Phase 1 auf den Keller gelegt werden, ist die Anzahl, wie oft man n wiederholt halbieren muss, bis 0 erreicht wird. Dieser Wert ist (ungefähr) der Logarithmus von n zur Basis 2. Statt n Multiplikationen kommt unser Algorithmus also mit $\log_2 n$ Halbierungen und höchstens $2 \cdot \log_2 n$ Multiplikationen aus. Das ist eine gewaltige Verbesserung. Abbildung 6 zeigt den Verlauf der Kellerinhalte beim Berechnen von 2^{19} mittels wiederholtem Quadrieren.

Das wiederholte Quadrieren kann man auch zum Potenzieren von Matrizen benutzen. Das hilft bei der schnellen Berechnung von Fibonacci-Zahlen. Die Fibonacci-Zahlen bilden die unendliche Folge natürlicher Zahlen $\mathit{fib}_0, \mathit{fib}_1, \mathit{fib}_2, \ldots$ mit der Eigenschaft $\mathit{fib}_0 = 0$, $\mathit{fib}_1 = 1$ und $\mathit{fib}_{n+2} = \mathit{fib}_{n+1} + \mathit{fib}_n$ (für alle $n \geqslant 0$). Um z.B. fib_4 zu berechnen, kann man zuerst $\mathit{fib}_2 = \mathit{fib}_1 + \mathit{fib}_0 = 1$, dann $\mathit{fib}_3 = \mathit{fib}_2 + \mathit{fib}_1 = 2$ und schließlich $\mathit{fib}_4 = \mathit{fib}_3 + \mathit{fib}_2 = 3$ bestimmen. Man berechnet also alle Fibonacci-Zahlen bis hin zu der, die man erreichen will. Um fib_n zu berechnen, benötigt man dann ungefähr n Additionen. Nun gibt es auch eine Darstellung der Fibonacci-Zahlen als Potenzen einer 2×2-Matrix. Es gilt für alle $n \geqslant 1$:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} \mathit{fib}_{n+1} & \mathit{fib}_n \\ \mathit{fib}_n & \mathit{fib}_{n-1} \end{pmatrix}$$

Setzt man

$$a = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

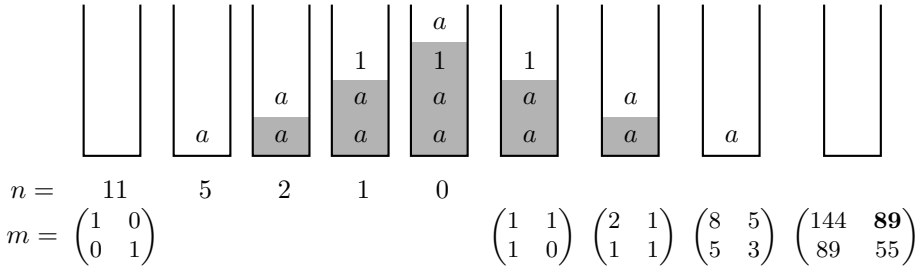


Abbildung 7: Der Verlauf der Kellerinhalte bei der Berechnung von $fib_{11} = 89$

im Algorithmus für das Potenzieren und gibt am Ende nicht die ganze Matrix sondern nur den Eintrag für fib_n aus, dann benötigt man $c \cdot \log_2 n$ Additionen und Multiplikationen, wobei c eine von n unabhängige Konstante ist.

4 Das Teile-und-herrsche-Prinzip

Die drei betrachteten Algorithmen zur Berechnung induktiv definierter Funktionen haben gemeinsam, dass eine „große“ Aufgabe in kleinere Aufgaben aufgeteilt wird, deren Lösungen dann zur Lösung für die große Aufgabe zusammengesetzt werden. Diese algorithmische Herangehensweise nennt man das Teile-und-herrsche-Prinzip. Beim Auswerten einer Formel der Art $(\cdots) \times (\cdots)$ werden zum Beispiel zuerst die beiden geklammerten Teilformeln ausgewertet und deren Werte schließlich zum Wert der ganzen Formel multipliziert. Bei der Berechnung der Potenz a^n werden zunächst die Potenzen $a^{\lfloor \frac{n}{2} \rfloor}$ und $a^{n \bmod 2}$ berechnet, und aus den beiden Teilergebnissen wird dann durch Quadrieren des ersten und Multiplikation mit dem zweiten Teilergebnis der Wert von a^n bestimmt. Um die Potenz $a^{\lfloor \frac{n}{2} \rfloor}$ auszurechnen, geht man genauso vor, bis man schließlich bei der Potenz a^0 angelangt ist, deren Wert 1 man kennt. Dieses Vorgehen kann man induktiv beschreiben

$$a^n = \begin{cases} 1, & \text{falls } n = 0 \\ a^{\lfloor \frac{n}{2} \rfloor} \cdot a^{n \bmod 2}, & \text{falls } n \geqslant 1 \end{cases}$$

und man programmiert es üblicherweise rekursiv, ohne im Programm explizit einen Keller zu benutzen. Die Datenstruktur Keller ist in diesem Sinne in alle üblichen Programmiersprachen eingebaut. Die Daten, die in solchen Kellern gespeichert werden, sind dann nicht nur Zwischenwerte sondern auch Angaben über noch zu lösende Teilaufgaben. Das soll am folgenden Beispiel verdeutlicht werden. Wir betrachten den Algorithmus Mergesort, der nach dem Teile-und-herrsche-Prinzip ein Feld mit Zahlen sortiert. Das „Teilen“ besteht darin, dass man das zu sortierende Feld in zwei (fast) gleich große Hälften aufteilt. Besteht ein Feld nur aus einem Eintrag, so ist es bereits sortiert. Zwei sortierte Felder werden nun zu einem sortierten Feld verbunden. Die organisatorische Arbeit des Algorithmus besteht darin, die Reihenfolge festzulegen, in der zwei (Teil-)Felder verbunden werden. Abbildung 8 zeigt grob den rekursiven Teil von Mergesort zur Strukturie-

```
procedure teile(von-bis)
  hälfte_links und hälfte_rechts sind die beiden Punkte in der Mitte
                                                    des Intervalls von-bis
  falls von + 1 < bis dann
    teile(von-hälfte_links)
    teile(hälfte_rechts-bis)
  verbinde(von-hälfte_links, hälfte_rechts-bis)
```

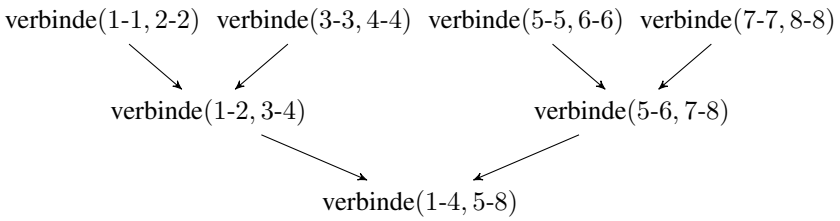


Abbildung 8: Die rekursive Prozedur für Mergesort und die zum Sortieren eines Feldes mit 8 Einträgen zu lösenden Teilaufgaben

rung der Verbindungsoperationen, und den darin implizit beschriebenen Struktur-Graph der auszuführenden Verbindungsoperationen zum Sortieren eines Feldes mit 8 Einträgen. So bedeutet „verbinde(1-4, 5-8)“, dass das Teilfeld mit den ersten vier Einträgen mit dem Teilfeld mit den zweiten vier Einträgen verbunden werden soll. Die Verbindungsoperation funktioniert allerdings nur, wenn beide Teilfelder bereits sortiert vorliegen. Die eingehenden Pfeile geben an, welche Verbindungsoperationen deshalb bereits vorher ausgeführt worden sein mussten.

Goldstine und von Neumann – die Erfinder von Mergesort – hatten 1947 in ihrer Arbeit über Mergesort [GvN48] weder Keller noch Rekursion zur Verfügung. Deshalb mussten sie einen recht hohen technischen Aufwand betreiben, um den Strukturierungsprozess zu beschreiben. Durch die Verinnerlichung der Denkweise des Kellers und der darauf basierenden Rekursion ist uns dieses Problem abgenommen worden.

Literatur

[GvN48] Herman H. Goldstine und John von Neumann. Planning and coding for an electronic computing instrument. Bericht, The Institute for Advanced Study, Princeton, NJ, 1948.

Friedrich L. Bauers und Klaus Samelsons Arbeiten in den 1950er-Jahren zur Einführung der Begriffe Kellerprinzip und Kellerautomat

Hans Langmaack

Christian-Albrechts-Universität zu Kiel
Institut für Informatik
Christian-Albrechts-Platz 4
24118 Kiel
hl@informatik.uni-kiel.de

Kurzfassung: F. L. Bauer und K. Samelson leisteten außerordentliche Pionierbeiträge zur Informatik, insbesondere zur Programmierung elektronischer Rechenanlagen und zur Entwicklung, Definition und Implementierung höherer Programmiersprachen. Über die im Titel genannte Thematik Kellerprinzip und -automat hinaus wird auf einige Vorläufer des Kellergedankens eingegangen.

1 Einleitung

Für die ehrenvolle Einladung, im Jenaer „Kellerkolloquium“ diesen Vortrag zu halten, danke ich sehr. Die beiden Informatikpioniere kenne ich seit 1960. Am 1. Juni trat ich meinen Dienst als (an der Universität Münster frisch Promovierter) wissenschaftlicher Assistent von Herrn Samelson am Mathematischen Institut der Johannes-Gutenberg-Universität in Mainz an. Herr Bauer war Chef des Instituts für Angewandte Mathematik, das seit 1958 den Zuse-Rechner Z22 in Betrieb hatte, wofür Manfred Paul den ALGOL 60-Übersetzer ALCOR MAINZ Z22 konstruierte und programmierte, der Anfang 1961 fertiggestellt war.

Im Herbst 1960 wurde der Rechner SIEMENS 2002 installiert. Weil die Herren Bauer und Samelson über den mitgelieferten ALGOL 60-Übersetzer überhaupt nicht glücklich waren, erhielten Frau Ursula Hill (später Frau Hill-Samelson) und ich den Auftrag, den ALGOL 60-Übersetzer ALCOR MAINZ 2002 zu konstruieren, der Anfang 1962 fertig wurde. Damals lernte ich auch Herrn Wilhelm Kämmerer kennen. Er besuchte uns ca. 1961 in Mainz und interessierte sich für unsere Arbeiten am Übersetzer. Herrn Kämmerers Buch [Kaem60] „Ziffernrechenautomaten“ steht in meinem Bücherregal. Ich kaufte es mir als mein erstes Buch über reale Rechenmaschinen im Kontrast zu den Turing-Maschinen, die ich während meines Nebenfachstudiums „Mathematische Logik“ in Münster kennengelernt hatte [Her61].

Bauer und Samelson leisteten in den 1950/60er-Jahren außerordentliche Pionierbeiträge zur Informatik, insbesondere zur Programmierung elektronischer Rechenanlagen und zur Entwicklung, Definition und Implementierung höherer Programmiersprachen. Die Beiträge

sind heute selbstverständliches informatisches Ideengut; es musste aber erst durch harte Gedankenarbeit und gehöriges Abstrahieren aus dem praktischen Umgang mit elektronischen Rechnern herausdestilliert werden. Der Umgang war damals ein recht handwerkliches Betreiben von Mathematik; die meisten Mathematiker sahen darin überhaupt keine adäquate mathematische Tätigkeit.

Nach Physik- und Mathematikstudium (das Samelson im Dritten Reich verwehrt war) an der Ludwig-Maximilians-Universität München und Promotion in Theoretischer Physik bot sich für die beiden Lehramtskollegen Bauer und Samelson 1952 die große Gelegenheit, wissenschaftliche Mitarbeiter von Robert Sauer am Mathematischen Institut der Technischen Universität (damals Technische Hochschule) München zu werden und am von Hans Piloty und Robert Sauer initiierten Bau und Einsatz (Programmierung) der PERM, der Programmgesteuerten Elektronischen Rechenanlage München, von Anbeginn mitzuwirken. Die ersten Ergebnisse wurden auf der Göttinger Rechenmaschinentagung 1953 vorgestellt und in Journalartikeln 1953, 1954 ausgeweitet [BaSa53, BaSa54]. Es ging zunächst um Bedeutung nichtnormalisierter Gleitkommarechnung und Kompaktifizierung von Elementarbefehlen; für spätere Übersetzertechnik war besonders wegweisend die tiefe Auseinandersetzung mit Unterprogrammtechnik und automatisierter Adressmodifikation in Fortführung der Gedanken von H. Goldstine und J. v. Neumann von 1947 [GovN47].

2 K. Samelsons Dresdener Vortrag 1955

Einen ganz entscheidenden Schritt machte Samelson in seinem Vortrag „Probleme der Programmierungstechnik“ 1955 auf dem Internationalen Kolloquium über Probleme der Rechentechnik in Dresden [Sam55/57]. Die PERM war fertiggestellt und lief, für Bauer und Samelson war es das Signal, sich von den konkreten Maschinen zu lösen und abstrakter zu werden. Abstraktion führt nämlich nicht zu Unbrauchbarkeit, im Gegenteil, sie führt zum Eigentlichen, auch in der Informatik. Samelson forderte, „[. . .] die Programmierung soweit wie nur irgend möglich der in jahrhundertelanger Entwicklung entstandenen Symbolik der Mathematik anzugleichen“. Dieser Gedanke, nämlich die Entwicklung und die Programmierung der realen Computer ganz und gar in die große mathematische Tradition stellen zu sollen und das auch zu können, wurde hier wohl zum ersten Mal so deutlich formuliert.

Bauer und Samelson griffen den Rutishauser’schen Gedanken von 1951 [Rut51] des sog. Superplans auf, um diesen Gedanken erheblich zu konkretisieren. Ein Superplan war ein Übersetzerprogramm, ein Compiler, der aus sog. höherem Pseudocode (aus Pseudoprogrammen) Maschinenprogramme erzeugte. Samelson wollte Superpläne mit den folgenden drei Hilfsmitteln realisieren:

- Mit Bibliotheksprogrammen als geschlossenen Unterprogrammen,
- mit Teilprogrammen als offenen Unterprogrammen,
- mit den wohl von Wilkes [Wil53] erstmalig angegebenen algebraischen Adressen, später symbolische Adressen genannt.

Samelson zog den interessanten Vergleich, dass die offenen Unterprogramme gut zur Verdrahtung geeignet seien, während die geschlossenen zu komplex seien, sodass man deren Verdrahtung nicht mehr geschafft habe. Und die Maschinenadressen zu symbolischen Adressen habe man in sog. Telephonbüchern zu suchen.

Bemerkenswert war, dass er die Grundprobleme der Programmierungstechnik zum Bau von programmerzeugenden Superplänen als gelöst ansah und den intellektuellen Mut zum Propagieren universeller höherer Programmiersprachen aufbrachte: „Wir kommen zum Pseudoprogramm und damit zu *dem* Problem der Programmierungstechnik. Der Grund für diese kühn klingende Behauptung ist folgender: Mit der Verwendung von Pseudoprogrammen hat man sich vom speziellen Maschinencode völlig gelöst [...]. [Bisher] nicht gelöst ist das Problem der Verständigung zwischen den verschiedenen um je eine Maschine gruppierten Rechenzentren. Und dieses Problem bedarf unbedingt einer Lösung, die nur in der Einführung eines universellen Pseudocodes bestehen kann." Für Bauer und Samelson war völlig klar, woher ein solcher universeller Pseudocode zu nehmen sei. Es gebe den Kern des Codes ja bereits seit 500 Jahren bei den Rechenmeistern des ausgehenden Mittelalters und der Renaissance. Wir alle haben die arithmetische Formelsprache seit der Sexta kennengelernt. Samelson schrieb insbesondere, dass „die Programmierung, also die formale Seite des Maschinenrechnens, die Entwicklung der Mathematik an sich völlig wiederholt [...]". Wenn der universelle Pseudocode von den Rechenmaschinen gänzlich gelöst werde, dann stehe i. Allg. keine Maschine zur Verfügung, sodass er durch Übersetzen in bekannten Maschinencode interpretiert werden könne. Es müsse aber eine Interpretationsmöglichkeit geschaffen werden. Für Bauer und Samelson war die Semantik der arithmetischen Formeln durch deren Auswertung oder Durchrechnung klar definiert, die alle schulmathematisch gebildeten Menschen im Prinzip in gleicher Weise durchführen. Das reiche aus, um arithmetische Formelsprache und selbst kompletten universellen Pseudocode weltweit zu verstehen. Dazu brauche man keine konkreten Maschinen zu kennen.

Samelson diskutierte dann das systematische Auswerten arithmetischer, teilweise geklammerter Formeln in der wohlbekannten Infixnotation, um daraus Rezepte für richtiges Interpretieren bzw. Übersetzen zu gewinnen. Samelsons Auswerten ist ein Formelabbauen, wobei er immer wieder die linkeste Abbaumöglichkeit sucht und ausnutzt. Dass so ein Procedere funktioniert und semantikgemäß erlaubt ist, ist damals selbst Fachleuten durchaus nicht geläufig gewesen. Denn Backus et al. [Bac$^+$57] mit ihrem 1954–57 gebauten FORTRAN-Compiler propagierten, zuerst fehlende Klammern einzufügen und dann sukzessive innerste Klammerpaare abzubauen, wie es schon Rutishauser 1951 [Rut51] lehrte.

Woher rührt die Unentschiedenheit über bestmögliches Vorgehen? Die schon in der Schule gelernten Klammereinsparungs- und Vorrangregeln wie

- Potenzieren $\uparrow$ geht vor Multiplizieren $\times$ und Dividieren $/$,
- diese gehen vor Addieren $+$ und Subtrahieren $-$,
- bei gleichem Rang von Operatoren geht (meist) links vor rechts,
- und geklammerte Operanden haben Vorrang vor zugehörigem Operator

erlauben nämlich durchaus nichtdeterministisches Auswerten. Z. B. hat die Formel

$$a - 5 + c \times (12 - e) + f \uparrow 2$$

15 verschiedene korrekte Auswertungsreihenfolgen mit gleichem Endergebnis. Wenn die Variablen a, c, e, f die Werte $12, 2, 3, 3$ haben, dann ist stets 34 das Endergebnis. Bei vollständiger Klammerung, etwa in

$$(\, (\, \underset{7}{\underline{(a-5)}} + (c \times \underset{9}{\underline{(12-e)}} \,) \,) + \underset{9}{\underline{(f \uparrow 2)}} \,),$$

ist die Auswertungskonfluenz (in der Theorie der λ-Kalküle spricht man von Church-Rosser-Eigenschaft) völlig evident, weil gefährliche Überlappungen, die zu Mehrdeutigkeiten führen können, ausgeschlossen sind. Gleichgültig in welcher Reihenfolge die lokalen Gipfel des Rutishauser'schen Klammergebirges abgetragen werden, gelangt man stets zu gleicher Normalform, zu gleichem Endergebnis. Bauer und Samelson haben den großen Hin-und-her-Leseaufwand aber nicht mitmachen wollen, weil an den eingefügten Klammern niemand interessiert ist. Die wegweisende Einsicht ist gewesen, dass hinter den Klammereinsparungs- und Vorrangregeln Auswertungsregeln, d. h. echt verkürzende Ersetzungs- oder Reduktionsregeln, stehen, die genau die gleichen Auswertungsschritte wie bei vorheriger vollständiger Klammerung ohne gefährliche Überlappungen zulassen.

$$\underset{\substack{< \quad \geq \\ 7}}{\underline{a-5}} + c \times (\, \underset{\substack{< \quad \geq \\ 9}}{\underline{12-e}} \,) + \underset{\substack{< \quad \geq \\ 9}}{\underline{f \uparrow 2}}$$

$$\begin{array}{c} \text{- - - -} \\ \geq \quad < \end{array}$$

So z.B. kommt die überlappende Addition $5 + c$ nicht zum Zuge, sogar aus zwei Gründen, wobei einer genügte: Der linke Nachbaroperator $-$ hat gleiche oder höhere Priorität, der rechte $\times$ hat echt höhere Priorität als der Operator $+$.

1955 spricht Samelson noch nicht vom Kellern, aber das Auftreten von Zwischenresultaten $H_i, H_{i+1}, \ldots$ (abgelegt im linearen Hilfsspeicher H), deren letztgewonnenes Resultat als erstes seinen Speicherplatz wieder freigibt (*last-in-first-out*-pulsierendes Verhalten), ist bereits klare Beschreibung des Begriffs Zahlkeller, der zuerst in der Patentschrift [BaSa57] „Verfahren zur automatischen Verarbeitung von kodierten Daten und Rechenmaschine zur Ausübung des Verfahrens“ 1957 von Bauer und Samelson auftritt.

3 F. L. Bauers und K. Samelsons Patentschrift 1957

Die o. g. Formel mit den erwähnten Variablenwerten führt zu folgenden sukzessive gelesenen und abgebauten Formelvorderteilen mit zuletzt gelesenen Symbolen, wobei es aus technischen Gründen zweckmäßig sein kann, ein(en) Formelvorderteil(-kellerinhalt) in Operations- und Zahlkellerinhalt aufzuspalten, wie es die Patentschrift empfiehlt:

bisher gelesenes und abgebautes Formelvorderteil	zuletzt gelesenes Symbol	Operationskellerinhalt	Zahlkellerinhalt
$a - 5$	$+$	$-$	$a\,5$
7	$+$	$\emptyset$	7
$7 + c \times (12 - e$	$)$	$+ \times (-$	$7\,c\,12\,e$
$7 + c \times (9$	$)$	$+ \times ($	$7\,c\,9$
$7 + c \times 9$	$+$	$+ \times$	$7\,c\,9$
$7 + 18$	$+$	$+$	$7\,18$
25	$+$	$\emptyset$	25
$25 + 3 \uparrow 2$	$\emptyset$	$+ \uparrow$	$25\,3\,2$
$25 + 9$	$\emptyset$	$+$	$25\,9$
34	$\emptyset$	$\emptyset$	34

In der Patentschrift geht es um die Erfindung eines Rechenautomaten, der gegenüber den bisherigen maschinenprogrammgesteuerten Rechenanlagen sog. formelgesteuert ist, d. h. gesteuert von Programmen einer höheren Programmiersprache mit Ergibtanweisungen, aufgebaut aus mathematischen Formeln in üblicher Infixnotation, Zahlen, Wahrheitswerten, einfachen und indizierten Variablen. Abbildung 1 zeigt den Schaltplan eines formelgesteuerten Rechenautomaten für eine einfachste Programmiersprache, deren Programme nur arithmetische Formeln mit Zahlenwerten als Operanden sind. Ich zitiere aus Spalte 2 von [BaSa57]: „Die Erfindung beruht i. w. auf dem Gedanken, den Komponenten einer Rechenmaschine einen Analysator beizuordnen, dem die mathematischen Formeln in üblicher Schreibweise zugeführt werden. [...]. [D]ie den einzelnen Zeichen entsprechenden Signale [werden] in der Reihenfolge der Aufschreibung dem Analysator zugeführt und in diesem [...] entsprechend der Reihenfolge des Eingangs geprüft, ob die Operationen sofort ausführbar sind oder ob der Eingang weiterer Signale abgewartet werden muß; in diesem letzteren Falle werden die noch nicht verarbeitbaren Zeichen in einen Speicher (Keller) eingeführt; beim Eintreffen neuer Zeichen im Analysator, die die Ausführung einer Operation mit gespeicherten Zeichen ermöglichen, werden diese gespeicherten Zeichen in der durch die Art der Einführung festgelegten, umgekehrten Reihenfolge entnommen und verarbeitet“.

Das oberste Geschoss des Operationskellers nennt die Patentschrift Diskriminatorregister D. Am Ausgang des Vorentschlüsslers, im Entschlüsselungsregister E findet sich das gerade neu gelesene Symbol ein. Hier haben wir das Kennzeichnende des Bauer-Samelson'schen Kellerprinzips und Kellerautomaten: Jedes Paar von Inhalten von D und E entscheidet über die nächstfolgende Handlung des Operationsumsetzers (ausgebildet als Matrixschaltung Abbildung 2), d. h. ob und wie oberste Kellereinträge modifiziert werden, ob und welche Maschinenoperationen ausgeführt werden, ob weitergelesen wird oder ob der Auswertungsprozess als erfolgreich beendet angesehen werden kann.

Technikhistorisch interessant ist die folgende Beobachtung: Obwohl in den Jahren 1955–57 die Idee des Compilierens von höheren Programmiersprachen nach Maschinencode bekannt ist, halten die Autoren Bauer und Samelson wie auch die kooperierenden Kollegen auf der elektrotechnischen Seite, Hans und Robert Piloty (letzterer später an der TH Darm-

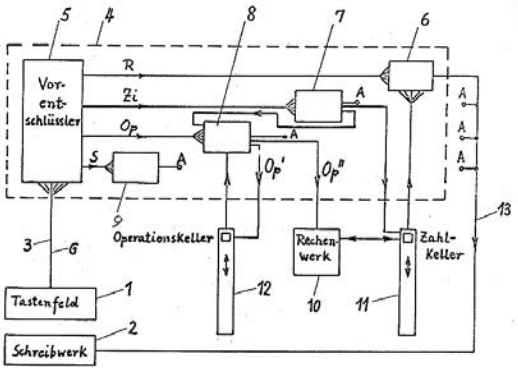


Abbildung 1: Schaltplan eines formelgesteuerten Rechenautomaten

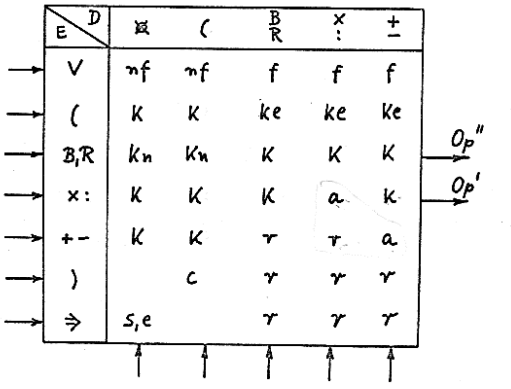


Abbildung 2: Matrixschaltung

stadt), die damals mögliche maschinelle Unterstützung durch Speicherkapazitäten und Rechengeschwindigkeit kaum für ausreichend, um real einsatzfähige Compiler herzustellen. Das Vertrauen in das Verstehen komplexer Schaltnetze für einen formelgesteuerten Rechenautomaten und in die materielle Umsetzungsmöglichkeit ist bei den Elektro- und Nachrichtentechnikern sehr hoch gewesen. Die Pläne für den entsprechenden Ausbau der PERM wurden dann aber doch nicht umgesetzt. In Hardware realisiert wurde der von Angstl und Bauer in den 1950er-Jahren konzipierte Relaisrechner STANISLAUS [Bau60, Bau77], der aussagenlogische, in polnischer Notation geschriebene Formeln auswertet (1950 entworfen als Mechanik, 1957 als Relaisrechner vorgeführt).

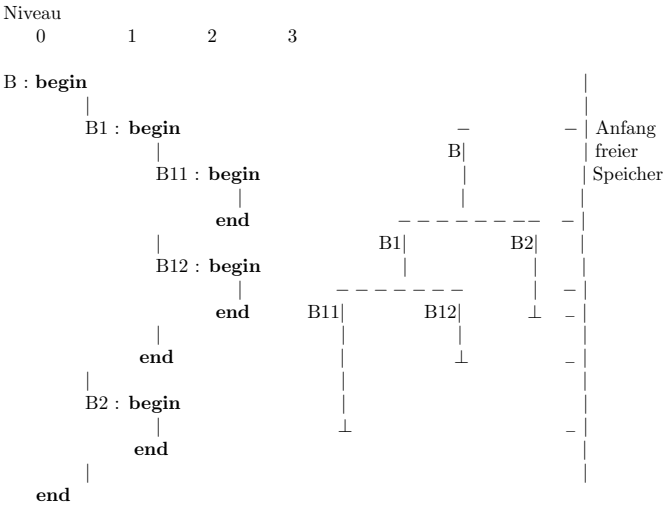
4 K. Samelsons und F. L. Bauers Artikel „Sequentielle Formelübersetzung“ 1959

Bauers und Samelsons Kellerprinzip und Kellerautomat sind 1957 noch nicht recht publik geworden, weil der Begriff „Keller“ eben nur in der Patentschrift stand und die Patentrichtlinien verlangten, dass über den Inhalt nur nach Auslegung der Schrift weiter publiziert werden dürfe. Daher ist das Bauer-Samelson'sche Kellerverfahren erst in dem 1959er Artikel [SaBa59] „Sequentielle Formelübersetzung“ in den „Elektronischen Rechenanlagen“ voll wahrgenommen worden. Wegen der Wichtigkeit für die Automatentheorie zitiere ich wörtlich: „In der Sprechweise der Theorie der Automaten kann das Prinzip so formuliert werden: Durch die gesamte Besetzung des Kellers wird ein Zustand (des Übersetzungsvorgangs) definiert, der effektiv in jedem Augenblick nur von dem obersten Kellerzeichen abhängt, und neu gelesene Information plus Zustand bestimmen die Aktionen [...]“. Die Autoren setzen nun ganz und gar auf das Übersetzen statt das Interpretieren. Vor ihrem geistigen Auge stand die international konzipierte höhere Programmiersprache ALGOL 58, 1958 definiert durch ein ACM/GAMM-Komitee, wofür Samelson zusammen mit A. J. Perlis den Bericht schrieb und herausgab [PeSa58/59].

In Samelsons Dresdener Vortrag hat es nicht bloß den einen deutlichen Hinweis auf kellerartiges Verhalten gegeben, nämlich das erwähnte Zahlkellerverhalten (ohne Erwähnung des Wortes „Keller“) beim Auswerten und Durchrechnen arithmetischer Formeln. Darüberhinaus lässt sich das Pulsieren des Zahlkellers zur Laufzeit auf den Datenspeicher für Teil- und Unterprogramme ausdehnen. Das tat auch Samelson und schrieb 1955: „Was in jedem Moment noch an für Zwischenrechnungen freien Speicherzellen zur Verfügung steht, hängt von der ganzen Vorgeschichte des Programms bis zum betrachteten Moment ab und läßt sich kaum vernünftig vom Compiler am Anfang festlegen. Es bleibt hier wohl nichts anderes übrig, als jeweils beim Übergang in ein neues abgeschlossenes Teilprogramm festzustellen, von wo ab der Speicher für Rechnungen frei ist, und diese Angabe dem Teilprogramm etwa in der Weise mit auf den Weg zu geben, daß man sie unter einem festen Variablennamen im ‚Telephonbuch‘ eintragen läßt, so daß also die zu der Variablen ‚Anfang freier Speicher‘ gehörige Rufnummer = Speicheradresse sich während der Rechnung laufend ändert“. Rechnungen eines neu aktivierten abgeschlossenen Unterprogramms brauchen insbesondere Speicherplätze für seine unterprogrammeigenen Variablen (einfache Variablen und Felder (arrays) wechselnder Größe), für Parameter (Argumente), Zwischenergebnisse, Unterprogrammresultat und Rückkehradresse.

Die wesentlichen Züge der Behandlung von Funktionen und Prozeduren werden in dem Artikel „Sequentielle Formelübersetzung“ [SaBa59] in Abschnitt „6. Vereinbarungen (declarations)“ diskutiert, wenn auch verhältnismäßig knapp; die Autoren sehen sich dazu gerechtfertigt, weil sie auf den Samelson'schen Dresdener Vortrag 1955/57 verweisen können. Sie schreiben 1959: „[. . .] Prozedur-Vereinbarungen [. . .] [führen] zu dynamischen [Unterprogrammen]. Dem aus der Auswertung der Vereinbarung resultierenden Unterprogramm ist also wie bei allen Unter- bzw. Bibliotheksprogrammen ein Anschlußteil voranzustellen, der die Übernahme der Rückkehradresse und der Programmparameter durchführt, im dynamischen Fall ist ein Adaptieren zur Berechnung des benötigten Hilfsspeichers und zur Adressierung der auf den Hilfsspeicher bezüglichen Befehle (mit Hilfe eines speziellen Parameters, der den Beginn des freien Speichers angibt) hinzuzufügen. Auch hier handelt es sich um bekannte Tatsachen, auf die nicht näher eingegangen zu werden braucht.“

Die Niveaustruktur von Blöcken und die damit einhergehende pulsierende Speicherverteilung gehören zu den wichtigsten Beiträgen Samelsons zur Programmiertechnik. Er brachte die Blockstruktur in ALGOL 60 [Nau+60] ein. Im ALGOL 58-Bericht [PeSa58/59] war noch keine Rede von Bindungs- und Gültigkeits-(Sichtbarkeits-)Bereichen von Identifikatoren, Phänomenen, die in Prädikatenlogik und λ-Kalkülen seit den 1930er-Jahren bekannt waren [Her61]. In einem ALGOL 58-Programm kann jeder Identifikator in nur einer Bedeutung auftreten. Daraus folgt für ALGOL 60, dass in parallelen Blöcken verschiedene einfache Variablen, Felder und Hilfsvariablen für Zwischenwerte gleiche Speicherplätze belegen dürfen. Das folgende Bild über Niveaustruktur von Blöcken und pulsierende Speicherverteilung stammt aus dem Buch von Bauer, Heinhold, Samelson, Sauer „Moderne Rechenanlagen“ 1965 [BHSS65, Lan02]:



Samelsons Vision über das Blockkonzept brachte die Mehrheit des ALGOL 60-Komitees dazu, dass sie sich von Samelson und Mitstreitern wie J. McCarthy, E. W. Dijkstra vom statischen Binden von Identifikatoren überzeugen ließen. Es wird dadurch zum Ausdruck

gebracht, dass bei Prozedurrumpf- und Parameterersetzungen (operationelle Kopierregelsemantik) Bindungsverfälschungen durch gebundene Identifikatorumbenennungen zu vermeiden waren. Sog. dynamisches Binden macht solche impliziten Umbenennungen nicht. Diese Überlegungen haben zur Folge, dass Programme mit Prozeduren sich als Programme mit i. Allg. unendlich tief geschachtelten generierten (modifiziert kopierten) Blöcken ohne Prozeduren und ohne Prozeduranweisungen auffassen lassen. Diese Sicht bietet eine Rechtfertigung für pulsierend arbeitende Laufzeitkeller. Rückblickend wären explizite Hinweise im ALGOL 60-Bericht auf α- und β-Reduktion in λ-Kalkülen für Programmierer und Übersetzerkonstrukteure sehr nützlich gewesen.

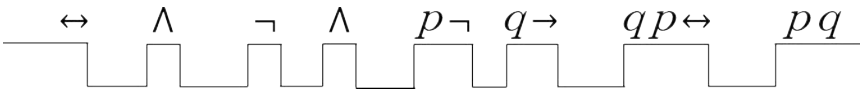
5 Vorläufer des Bauer-Samelson'schen Kellerprinzips

Wie steht es um Vorläufer des Bauer-Samelson'schen Kellerprinzips? 1950 entwickelt H. Angstl [Bau60] eine Mechanik, um klammerfreie aussagenlogische Formeln mit Symbolen der Stelligkeit ≥ 0 in polnischer Notation [Luk29] auf Wohlgeformtsein zu prüfen (vorgetragen 1950 im Logistikseminar von Prof. Britzelmayr an der LMU München). Angstl setzt die Symbole einer zu prüfenden Formel wie Spielsteine in eine Reihe, wobei je zwei aufeinanderfolgende Steine einen Zwischenraum der Größe der Stelligkeit des vorangehenden Symbols lassen. Der Boden jeden Zwischenraums wird abgesenkt um die Höhe der Spielsteine, sodass jeder Zwischenraum zur Fallgrube wird.

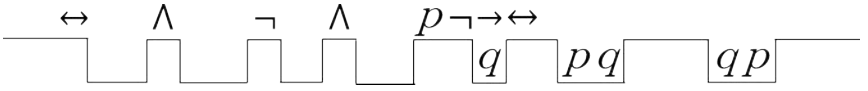
Beispiel: Die Tautologie

$$((\neg (p \wedge \neg q) \wedge (q \rightarrow p)) \leftrightarrow (p \leftrightarrow q))$$

in polnischer Notation



Die Spielsteine werden von hinten nach vorn geschoben. Trifft der vorderste geschobene Stein auf eine Fallgrube der Größe n, fallen die vorderen n geschobenen Steine hinein, sodass die übrigen darüber hinwegrutschen. Dabei schiebt man eine „kellerartig" pulsierende Serie von Spielsteinen vor sich her.



Wenn am Schluss alle Fallgruben vollgefüllt sind und ganz vorne nur noch exakt ein Stein im „Keller" übrig ist, genau dann ist die vorgelegte Formel wohlgeformt.

Dies ist im Grunde genau die Burks-Warren-Wright'sche mathematischer formulierte nichtrekursive Definition einer wohlgeformten Formel [BWW54] von 1954: Jedes Symbol bekommt ein

$$\textit{Gewicht} = 1 - \textit{Stelligkeit},$$

und eine Symbolreihe bekommt als *Gewicht* die Summe der Einzelgewichte. Eine nichtleere Symbolreihe heißt dann *wohlgeformte Formel*, wenn alle nichtleeren Postfixe positives Gewicht haben und das *Gesamtgewicht* 1 ist. Das positive Gewicht eines Postfix ist gerade die Länge des zugehörigen Angstl'schen Kellerinhalts, z. B. 4 ist die Länge von $p \neg \rightarrow \leftrightarrow$. Liest man eine Formel in polnischer Notation von hinten, dann reduziert sich der Bauer-Samelson'sche Operationskeller auf ein einziges Operationsregister.

Die rekursive Definition des Wohlgeformtseins einer Formel ist allerdings deutlich verständlicher und natürlicher als die obige nichtrekursive Definition. Die Äquivalenz beider Definitionen bedarf eigentlich eines Beweises, worauf weder Angstl noch Burks, Warren und Wright eingehen. Die rekursive Definition bietet aber kein unmittelbar deterministisches Entscheidungsverfahren.

1952 beschreibt W. L. van der Poel [vdP52] einen einfachen Rechner, dessen Befehle auch eine last-in-first-out-Organisation von Rückkehradressen bei wiederholt aufgerufenen Unterprogrammen erlauben. Das Charakterisierende eines Kellerautomaten ist aber nicht ganz einfach zu erkennen und wird nicht explizit formuliert.

S. Ginsburg unterscheidet in seinem Buch „The Mathematical Theory of Context-free Languages" 1966 [Gin66] zwischen bloßer Verwendung eines Kellerspeichers (pushdown tape) und dem Begriff Kellerautomat (pushdown automaton oder acceptor). Ein Kellerspeicher trete zuerst bei Burks, Warren und Wright 1954 auf und komme 1959/60 auch bei Bauer und Samelson vor. Der Kellerautomat sei dagegen zuerst von N.Chomsky [Cho62] formalisiert worden. Unser Darmstädter Kollege und anerkannter Automatentheoretiker Hermann Walter bestand aber in einer Diskussion der 1970er-Jahre [Wal70] darauf, dass F. L. Bauer und K. Samelson den Kellerautomaten 1957 erfunden hätten.

Dank: Ich danke Herrn Kollegen Friedrich L. Bauer für seinen Vorschlag, ich möge diesen Vortrag im Jenaer „Kellerkolloquium" halten. Ferner danke ich Frau Annemarie Langmaack für das Setzen dieses Artikels in LaTeX.

Literatur

[Bac+57] Backus, J. W., et al.: The FORTRAN Automatic Coding System. Proc. Western Joint Computing Conf. 11, 188-198 (1957)

[Bau60] Bauer, F. L.: The Formula-Controlled Logical Computer "Stanislaus". Math. Tab. a. Oth. Aids to Comp., Vol. XIV, 69, January (1960)

[Bau77] Bauer, F. L.: Angstl's Mechanism for Checking Wellformedness of Parenthesis-Free Formulae. Math. of Computation 31, 318–320 (1977)

[BaSa53] Bauer, F. L., Samelson, K.: Optimale Rechengenauigkeit bei Rechenanlagen mit gleitendem Komma. Z. Angew. Math. Phys. 4, 312–316 (1953)

[BaSa54] Bauer, F. L., Samelson, K.: Maßnahmen zur Erzielung kurzer und übersichtlicher Programme für Rechenautomaten. Z. Angew. Math. Mech. 34, 262–272 (1954)

[BaSa57] Bauer, F. L., Samelson, K.: Verfahren zur automatischen Verarbeitung von kodierten Daten und Rechenmaschinen zur Ausübung des Verfahrens. Patentanmeldung Deutsches Patentamt (1957)

[BHSS65] Bauer, F. L., Heinhold, J., Samelson, K., Sauer, R.: Moderne Rechenanlagen. Stuttgart: Teubner (1965)

[BWW54] Burks, A. W., Warren, D. W., Wright, J. B.: An Analysis of a logical machine using parenthesis-free notation. Mathematical Tables and Other Aids to Computation, vol. 8, pp. 53–57 (1954)

[Cho62] Chomsky, N.: Context-free grammars and pushdown storage. M.I.T. Res. Lab. Electron. Quart. Proc. Rept. 65 (1962)

[Gin66] Ginsburg, S.: The Mathematical Theory of Context Free Languages. McGraw-Hill (1966)

[GovN47] Goldstine, H., von Neumann, J.: Planning and Coding for an Elektronic Computing Instrument. Institute for Advanced Study: Princeton (1947)

[Her61] Hermes, H.: Aufzählbarkeit, Entscheidbarkeit, Berechenbarkeit. Berlin Göttingen Heidelberg: Springer (1961)

[Kaem60] Kämmerer, W.: Ziffernrechenautomaten. Akademie Verlag Berlin (1960)

[Lan02] Langmaack, H.: Klaus Samelsons frühe Beiträge zur Informatikentwicklung. Informatik Spektrum 18 (April 2002), 132–137 (2002)

[Luk29] Lukasiewicz, J.: Elementy Logiki Mathematycznej, Warszawa (1929)

[Nau$^+$60] Naur, P., (ed.) et al.: Report on the Algorithmic Language ALGOL 60. Num. Math. 2, 106–136 (1960)

[PeSa58/59] Perlis, A. J., Samelson, K. (eds.): ACM Committee on Programming Languages and GAMM Committee on Programming. Report on the Algorithmic Language ALGOL. Num. Math. 1, 41–60 (1959)

[Rut51] Rutishauser, H.: Über automatische Rechenplanfertigung bei programmgesteuerten Rechenanlagen. Z. Angw. Math. Mech. 31, 255 (1951)

[Sam55/57] Samelson, K.: Probleme der Programmierungstechnik. Intern. Koll. über Probleme der Rechentechnik, Dresden 1955. Berlin: VEB Deutscher Verlag der Wissenschaften, S. 61–68 (1957)

[SaBa59] Samelson, K., Bauer, F. L.: Sequentielle Formelübersetzung. Elektr. Rechenanl. 1 (4), 176–182 (1959)

[SaBa60] Samelson, K., Bauer, F. L.: Sequential Formula Translation. CACM, vol. 3, 76–83 (1960)

[vdP52] van der Poel, W. L.: Dead Programmes for a Magnetic Drum Automatic Computer. Appl. Sci. Res. B. 3: 190–198 (1952)

[Wal70] Walter, H.: Persönliche Mitteilung. Saarbrücken 1970er-Jahre

[Wil53] Wilkes, M. V.: The Use of a Floating Address System for Orders in an Automatic Digital Computer. Proc. Cambridge Philos. Soc. 49, 84–89 (1953)

Wilhelm Kämmerers Ideen vom automatischen Gedächtnis

Michael Fothe, Denny Steigmeier

Friedrich-Schiller-Universität Jena
Fakultät für Mathematik und Informatik
Ernst-Abbe-Platz 2
07743 Jena
michael.fothe@uni-jena.de
denny_steigmeier@gmx.de

Kurzfassung: In der Habilitationsschrift von Wilhelm Kämmerer aus dem Jahr 1958 finden sich Ideen zum Keller; er sprach vom automatischen Gedächtnis. Kämmerer setzte diese Struktur zum Aufbrechen von speziellen arithmetischen Formeln ein. Das entwickelte Verfahren eignet sich auch zur Verwaltung von Funktionsaufrufen und anderen Sprachkonstrukten.

Computerpioniere im deutschsprachigen Raum sind diejenigen, die von Konrad Zuse gemalt wurden.

Hartmut Wedekind

1 Zielstellung

Im September 2013 ehrte die Friedrich-Schiller-Universität Jena Wilhelm Kämmerer (1905–1994) mit einer Professorentafel an seinem langjährigen Wohnhaus, nachdem bereits im April 2013 eine „Oprema-Tafel“ am Gebäude der Fakultät für Mathematik und Informatik angebracht wurde [Webc, Webd, Webb]. Damit sollte einem möglichen Vergessen von Pionierleistungen der Informatik entgegengewirkt werden [Ker06]. Es entstand die Idee, sich genauer mit der Habilitationsschrift von Kämmerer zu befassen und diese einem breiteren Publikum bekannt zu machen [Käm58].

Kämmerer geht in seiner Habilitationsschrift einer praktisch bedeutsamen Fragestellung nach und entwickelt dabei Grundlagen für den Compilerbau und damit für den Einsatz höherer Programmiersprachen. Die Untersuchungen beruhen auf Erfahrungen, die er als Entwicklungsleiter im VEB Carl Zeiss Jena mit der Oprema, einem der ersten Computer Deutschlands, und dem Nachfolger ZRA 1 gewonnen hatte [Win10]. Kämmerer schreibt einschränkend: „Diese Abhandlung strebt nicht eine konstruktive Bauvorschrift an, wenn sie auch gelegentlich geschehene Hinweise in dieser Richtung für nicht unangebracht hält“ [Käm58, S. 69]. Im Rahmen einer studentischen Projektarbeit implementierte Denny Steigmeier das von Kämmerer entwickelte Verfahren als Java-Programm (siehe Abschnitt 5

und Quelle [Webe]). Auch hier trat das ein, was Raúl Rojas und andere bei ihren Implementierungsarbeiten zu einer Untermenge des Plankalküls von Konrad Zuse erlebten, dass nämlich mit solchen Projekten „Computergeschichte plötzlich und unerwartet lebendig wird“ [RGF+04, S. 228 f.].

2 Ausgangssituation

Kämmerer führt zu Beginn seiner Habilitationsschrift aus: „[D]ie maschinengerechte Programmierung [erfordert] erheblichen Aufwand an Zeit und routinemäßiger Konzentration und führt zu einer unübersichtlichen, weitschweifigen, schwer kontrollierbaren und derartig individuellen Form, daß Verständigungen über angewandte Verfahren zwischen verschiedenen Rechenzentren nur schwer möglich sind.“ Und weiter: „War man zunächst froh, überhaupt Rechenautomaten zu besitzen, so mußte man bald erkennen, daß man ihre Vorteile mit einer zeitraubenden, Unsicherheit tragenden Programmierungsarbeit bezahlte“ [Käm58, S. 2]. Dieser Standpunkt wird verständlich, wenn man sich Programme aus der damaligen Zeit anschaut, wie zum Beispiel das folgende; es berechnet $z = x^2 + y^2$ [Käm60, S. 214 f.]:

000	0 01 101
001	0 04 101
002	0 07 103
003	0 01 102
004	0 04 102
005	0 02 103
006	0 07 103
007	0 16 103
008	0 00 ...

Kämmerer geht in seiner Arbeit auf die Rechenplanberechnung nach Rutishauser näher ein und schreibt: „Das Charakteristische dieses Verfahrens ist ein fortgesetztes Durchmustern mit einem immer wieder von links nach rechts wechselnden Augenspiel“ [Käm58, S. 27]. Das Verfahren besitzt einen schlechten (quadratischen) Zeitaufwand mit der Eingabelänge. Friedrich L. Bauer bezeichnet es wohl in Anspielung an die Prozession in Echternach (Luxemburg) als „Springprozession Rutishausers“ [Bau04, S. 238]. Kämmerer führt aus: „In der vorliegenden Abhandlung werden in konsequenter Weiterführung der abgelaufenen Entwicklungstendenz auf einem neuen Weg Prinzipien für die Struktur eines Automaten erarbeitet, der in enger Anlehnung an das mathematische Formelbild eine bequeme, übersichtliche und adressenfreie Programmierung gestattet“ [Käm58, S. 3]. Und auch: „[D]as nach mathematischem Formelbild eingegebene Programm [ist] schon der Plan [...], den der Automat direkt verarbeiten kann“ [Käm58, S. 21]. Kämmerer nutzt als Gymnasiallehrer für Mathematik und Physik (promoviert in Mathematik) die Notation, mit der er bestens vertraut ist. Er stellt auch fest: „Das Programmieren würde mit der Aufstellung des algorithmischen Plans und der strukturellen Klärung der auftretenden Größen sein Ende finden“ [Käm58, S. 21].

3 Nutzung des automatischen Gedächtnisses

Das automatische Gedächtnis ist bei dem von Kämmerer entwickelten Verfahren zum Aufbrechen von Formeln von zentraler Bedeutung. Er schreibt: „[W]esentlich an der Methode ist, daß dem Automaten ein Gedächtnis eigen ist, über das er selbst verfügt“ [Käm58, S. 39]. Die Begriffe Gedächtnis und Speicher verwendet er synonym: „Eine nach diesen Prinzipien entwickelte Maschine wird über einen großen Teil des maschineneigenen Speicherraums selbst verfügen“ [Käm58, S. 25]. Die Arbeitsweise des Automaten kennzeichnet Kämmerer wie folgt: „[A]usführen wird er [...] jeweils nur, was sich als ausführungsmöglich bis dahin geklärt hat, das restliche wird er in seinem Gedächtnis zurückstellen [...]“ [Käm58, S. 67].

Wesentlich ist die Zelle vom Dienst (ZvD): „Der Schwerpunkt, die Stelle höchsten Bewußtseins, liegt dabei in der ZvD“ [Käm58, S. 39]. Kämmerer beschreibt die Erhöhung des Spannungsgrades: „Ein Aufsteigen in der Lage des ZvD [sic] ist mit Niederschrift von z. Z. hinsichtlich der Ausführungsbestimmungen unklar bleibenden Plananordnungen verbunden.“ Für eine Verringerung des Spannungsgrades gilt: „Ein Absteigen in der Lage des ZvD [sic] ist mit Lesen von Rückständen und entsprechenden Ausführungsmaßnahmen verbunden“ [Käm58, S. 30]. „Dabei sind jeweils alle Zellen mit Nummern über der der ZvD noch leer, oder ihr Inhalt ist schon wieder uninteressant geworden. Alle Zellen, deren Nummer unter der der ZvD liegen, haben dagegen wesentliche Informationen [...]“ [Käm58, S. 30]. „Während des Anstehens eines Befehls im Befehlsregister verbleibt der größte Teil dieser Gedächtniszellen im ‚Unterbewußtsein‘ des Automaten“ [Käm58, S. 39]. Wie Kämmerer auf diese psychoanalytische Begrifflichkeit gekommen ist, konnte nicht geklärt werden.

In der Habilitationsschrift sind zwei Beispiele beschrieben. In der Beschreibung des ersten Beispiels (siehe Abbildung 1) kann man das typische „Gebirge“ erkennen, das beim Aufbrechen von Formeln entsteht.

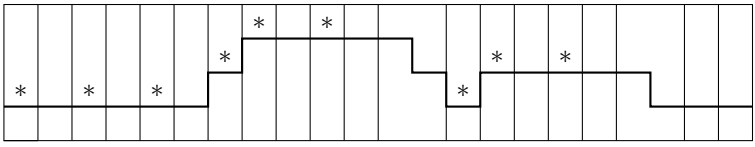


Abbildung 1: Lage der Zelle vom Dienst (ZvD) bei der Bearbeitung des Beispiels $a - b + c * (d + e) - (f + g) \Rightarrow h$. Der Doppelpfeil ist der Zuweisungsoperator. Der Stern bedeutet: Neubeschreibung der ZvD [Käm58, S. 49].

Nachfolgend sind drei Beispiele mit steigender Komplexität angegeben; diese sind nicht in der Habilitationsschrift enthalten. In der Spalte 1 sind unter der Überschrift „Rechenplan“ die eingelesenen Zeichen aufgeführt. Der Spalte 2 ist der aktuelle Wert des Resultatsregisters (RR) zu entnehmen. Die ZvD kann einen Wert sowie Anmerkungen (Zeichen) aufnehmen. In der Spalte 3 steht der Wert, in der Spalte 4 stehen die Anmerkungen Q_1 und Q_2, wenn sie gesetzt werden. Spalte 5 gibt die Lage der ZvD an. Zu Beginn und am Ende der Bearbeitung ist die Lage stets 1. In den Spalten 6 und 7 sind die Abfragen, ob die Anmerkungen Q_1 bzw. Q_2 gesetzt sind, protokolliert. In diesem Beitrag wird der Stern als Multiplikationsoperator verwendet (Kämmerer verwendete den Punkt).

	1 Rechenplan	2 RR	3 ZvD	4 Q_i	5 Lage ZvD	6 Q_1?	7 Q_2?
1	Anfang	0	0		1.		
2	a	a			1.		
3	$*$				1.		
4	b	$a * b$			1.		
5	$-$		$0 + a * b$	Q_1	1.	nein	
6	c	c			1.		
7	$*$				1.		
8	d	$c * d$			1.		
9	$\Rightarrow$	$a * b - c * d$			1.	ja	
10	r	$\langle \text{RR} \rangle \Rightarrow r$			1.		

Tabelle 1: Beispiel $a * b - c * d \Rightarrow r$

Schauen wir uns nun das Beispiel aus der Tabelle 1 genauer an. Am Anfang werden das Resultatsregister und die Zelle vom Dienst gelöscht. Ein Wert wird nach dem Einlesen in das Resultatsregister gebracht (Zeilen 2 und 6). Es gibt eine Besonderheit: Wird nach dem Einlesen eines Multiplikationsoperators ein Wert eingelesen, so wird der Inhalt des Resultatsregisters sofort mit diesem Wert multipliziert und das Produkt wird in das Resultatsregister gebracht (Zeilen 4 und 8). Nach dem Einlesen des Subtraktionsoperators (Zeile 5) wird geschaut, ob die Anmerkung Q_1 gesetzt ist (Spalte 6). Da sie in unserem Beispiel nicht gesetzt ist, wird nicht subtrahiert, sondern der Wert der ZvD und der Inhalt des Resultatsregisters werden addiert; die Summe wird in der ZvD gespeichert. Zusätzlich wird die Anmerkung Q_1 gesetzt. Die nachhängende Subtraktion wird in der Zeile 9 nach dem Einlesen des Zuweisungsoperators ausgeführt. Vom Wert der ZvD wird der Inhalt des Resultatsregisters subtrahiert; das Ergebnis wird in das Resultatsregister gebracht. In der Zeile 10 wird das Berechnungsergebnis der Variablen r zugewiesen. In dem Beispiel wird nur eine ZvD benötigt (siehe Spalte 5).

Das Beispiel aus der Tabelle 2 enthält ein Klammerpaar. Nach dem Einlesen des Multiplikationsoperators (Zeile 5) wird, da eine öffnende Klammer folgt, der Spannungsgrad um 1 erhöht. Die Anmerkung Q_2 wird gesetzt; sie steht für eine nachhängende Multiplikation. (Das Setzen der Anmerkungen Q_2 und Q_1 würde für eine nachhängende Division stehen.) Des Weiteren wird der Inhalt des Resultatsregisters in die ZvD kopiert. Nach dem Einlesen der öffnenden Klammer (Zeile 6) wird der Spannungsgrad um 1 erhöht und das Resultatsregister und die ZvD werden gelöscht. Nach dem Einlesen der schließenden Klammer (Zeile 10) folgen zwei Schritte: Zuerst wird eine nachhängende Subtraktion ausgeführt. Vom Wert der ZvD wird dazu der Inhalt des Resultatsregisters subtrahiert; das Ergebnis wird in das Resultatsregister gebracht. Der Spannungsgrad wird um 1 verringert. Gleich anschließend wird eine nachhängende Multiplikation ausgeführt (Zeile 11), und zwar werden der Wert der ZvD auf Stufe 2 (Wert von b) und der Inhalt des Resultatsregisters multipliziert; das Produkt kommt in das Resultatsregister. Der Spannungsgrad wird noch einmal um 1 verringert. Nach dem Einlesen des Subtraktionsoperators (Zeile 12) wird die nachhängende Addition ausgeführt und Q_1 wird gesetzt. Die nachhängende Subtraktion

wird nach dem Einlesen des Zuweisungsoperators ausgeführt (Zeile 14). Abschließend erfolgt die Zuweisung des Ergebnisses an die Variable s.

	1 Rechenplan	2 RR	3 ZvD	4 Q_i	5 Lage ZvD	6 Q_1?	7 Q_2?
1	Anfang	0	0		1.		
2	a	a			1.		
3	$+$		$0+a$		1.	nein	
4	b	b			1.		
5	$*$		b	Q_2	2.		
6	$($	0	0		3.		
7	c	c			3.		
8	$-$		$0+c$	Q_1	3.	nein	
9	d	d			3.		
10	$)$	$c-d$			3.	ja	
11		$b*(c-d)$			2.	nein	ja
12	$-$		$a+b*$ $(c-d)$	Q_1	1.	nein	
13	e	e			1.		
14	$\Rightarrow$	$a+b*(c-d)-e$			1.	ja	
15	s	$\langle \text{RR} \rangle \Rightarrow s$			1.		

Tabelle 2: Beispiel $a+b*(c-d)-e \Rightarrow s$

Das Beispiel aus der Tabelle 3 ist extrem, denn die erste Operation, die beim Durchlaufen der Zeichenfolge von links nach rechts ausgeführt wird, ist die am weitesten rechts stehende. Es wird also als erstes $i+k$ gerechnet. Bis dahin werden die Operationen, die verzögert ausgeführt werden sollen, und die benötigten Operanden im automatischen Gedächtnis gespeichert. Ab Zeile 25 werden die Operationen nachträglich ausgeführt, und zwar abwechselnd eine Addition und eine Multiplikation.

4 Funktionsaufrufe

Das in der Habilitationsschrift entwickelte Verfahren ermöglicht auch Funktionsaufrufe aus dem automatischen Gedächtnis heraus. Zulässig sind Funktionen von einem oder mehreren Parametern und in Verschachtelung [Käm58, S. 68]. Parameter können zusammengesetzte Ausdrücke sein [Käm58, S. 37 f.]. Das Vorgehen entspricht der Übergabeart *call by value* für Parameter. Auch wenn rekursive Aufrufe von Funktionen nicht vorgesehen sind, handelt es sich um einen insgesamt weitgehenden Leistungsumfang.

Das zweite Beispiel, das in der Habilitationsschrift beschrieben ist, lautet in der maschinengerechten Schreibweise der Plangleichung [Käm58, S. 50]:

$$\exp(-\text{pot}(x;2)) * \sin(c*x) \Rightarrow z$$

	1 Rechenplan	2 RR	3 ZvD	4 Q_i	5 Lage ZvD	6 Q_1?	7 Q_2?
1	Anfang	0	0		1.		
2	a	a			1.		
3	$+$		$0+a$		1.	nein	
4	b	b			1.		
5	$*$		b	Q_2	2.		
6	$($	0	0		3.		
7	c	c			3.		
8	$+$		$0+c$		3.	nein	
9	d	d			3.		
10	$*$		d	Q_2	4.		
11	$($	0	0		5.		
12	e	e			5.		
13	$+$		$0+e$		5.	nein	
14	f	f			5.		
15	$*$		f	Q_2	6.		
16	$($	0	0		7.		
17	g	g			7.		
18	$+$		$0+g$		7.	nein	
19	h	h			7.		
20	$*$		h	Q_2	8.		
21	$($	0	0		9.		
22	i	i			9.		
23	$+$		$0+i$		9.	nein	
24	k	k			9.		
25	$)$	$i+k$			9.	nein	
26		$h*(i+k)$			8.	nein	ja
27	$)$	$g+h*(i+k)$			7.	nein	
28		$f*(g+h*(i+k))$			6.	nein	ja
29	$)$	$e+f*(g+h*(i+k))$			5.	nein	
30		$d*(e+f*(g+h*(i+k)))$			4.	nein	ja
31	$)$	$c+d*(e+f*(g+h*(i+k)))$			3.	nein	
32		$b*(c+d*(e+f*(g+h*(i+k))))$			2.	nein	ja
33	$\Rightarrow$	$a+b*(c+d*(e+f*(g+h*(i+k))))$			1.	nein	
34	t	$\langle \text{RR} \rangle \Rightarrow t$			1.		

Tabelle 3: Beispiel $a+b*(c+d*(e+f*(g+h*(i+k)))) \Rightarrow t$

Nachfolgend wird der Funktionsaufruf $\mathrm{pot}(x; n)$ zum Berechnen einer Potenz erläutert. Der Wert des ersten Parameters wird (ggf. nach dessen Berechnung) der Speicherzelle am Anfang des Unterprogramms pot zugewiesen. Der Wert des zweiten Parameters wird (ggf. nach dessen Berechnung) der nachfolgenden Speicherzelle $\mathrm{pot} + 1$ zugewiesen. Für die Parameterübergabe gilt in unserem Beispiel: $x \Rightarrow \langle \mathrm{pot} \rangle, 2 \Rightarrow \langle \mathrm{pot} + 1 \rangle$ [Käm58, S. 51]. Danach erfolgt ein Sprung nach $\mathrm{pot} + 2$ und das Unterprogramm wird abgearbeitet. Anschließend enthält das Resultatsregister den Funktionswert. Bei Kämmerer heißt es: „Greift hinsichtlich der ZvD um eine Stufe zurück und erhält dort die Rückkehradresse“ [Käm58, S. 38]. In unserem Beispiel liefert das automatische Gedächtnis die Adresse $\mathrm{A}+10$. Ein Sprung dorthin im Hauptprogramm liefert den nächsten Befehl, der abzuarbeiten ist.

5 Implementierung des Verfahrens

Die von Kämmerer angegebene Bauvorschrift wurde als Java-Programm implementiert. Das Programm besitzt einen objektorientierten Aufbau; es besteht aus acht Klassen (siehe Abbildung 2).

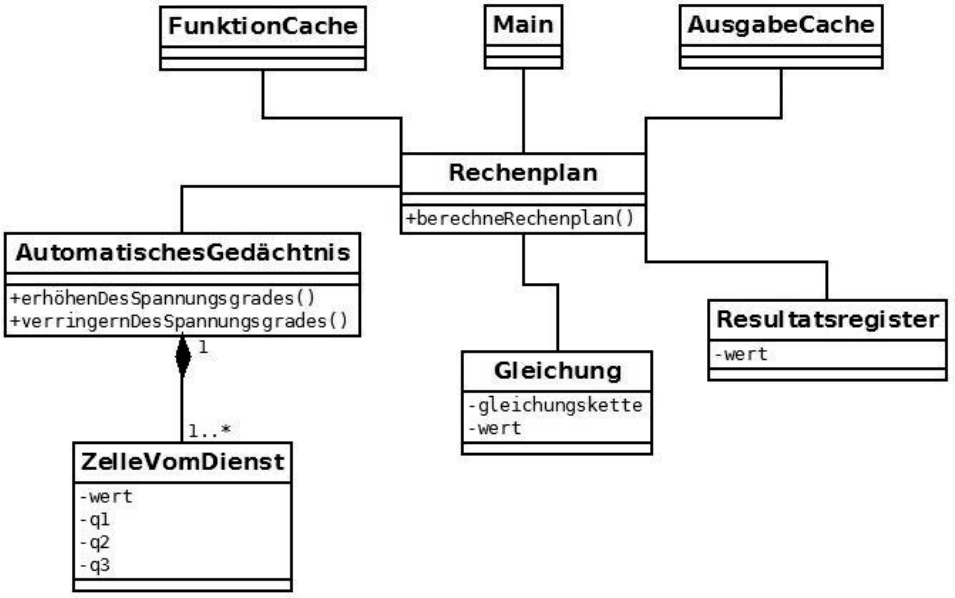


Abbildung 2: Aufbau der Implementierung

Die Klassen Main, FunktionCache und AusgabeCache stellen technische Klassen dar. Die Main-Klasse dient zum Initialisieren der benötigten Objekte und zum Starten des Programms. Im AusgabeCache werden die jeweiligen Ausgaben des Rechenplans zwischengespeichert und nach der Berechnung auf der Java-Oberfläche ausgegeben. Der

FunktionCache ist ein Zwischenspeicher zum Abarbeiten von Bibliotheksprogrammen bzw. Funktionen. Der Rechenplan ist die zentrale Klasse. Diese Klasse besitzt eine Gleichung, ein Resultatsregister und ein automatisches Gedächtnis. In der Gleichung ist die Gleichungskette hinterlegt und das Resultatsregister dient zum Abspeichern des Inhalts gemäß Rechenplan. Das automatische Gedächtnis ist als Stack implementiert und besitzt die Methoden `push()` (Erhöhung des Spannungsgrades) und `pop()` (Verringerung des Spannungsgrades). Die Elemente dieses Speichers sind die ZvD. Jede ZvD besitzt einen Zeiger auf ihren Vorgänger bzw. einen Null-Zeiger im Falle der untersten ZvD.

Nachdem der Programmbenutzer die Formel eingegeben hat, wird diese zunächst dem Rechenplan übergeben und in der Klasse Gleichung hinterlegt. Im nächsten Schritt wird die Methode `berechneRechenplan()` in der Klasse Rechenplan aufgerufen. Diese Methode liest die Zeichen der Formel schrittweise aus. Die Zeichen werden einer Switch-Struktur übergeben, in der sie erkannt und die entsprechenden Methoden aufgerufen werden. Mögliche Aufrufe sind:

- `symbol()`
- `addition()`, `subtraktion()`
- `multiplikationMitSymbol()`, `multiplikationOhneSymbol()`
- `divisionMitSymbol()`, `divisionOhneSymbol()`
- `klammerAuf()`, `klammerZu()`
- `semikolon()`
- `ergibt()`

Eine Ausnahme zur schrittweisen Abarbeitung der Zeichen besteht bei einem Multiplikationsoperator. Hierbei wird zusätzlich auf das folgende Zeichen der Gleichung zugegriffen. Nachdem der Rechenplan abgearbeitet wurde, kann das Ergebnis der Benutzungsoberfläche übergeben und ausgedruckt werden.

Die Darlegungen in der Habilitationsschrift erwiesen sich als konsistent und als geeignete Grundlage für die Implementierung des entwickelten Verfahrens.

6 Charakterisierung des Verfahrens

An der Verarbeitung von Ausdrücken kann man vieles studieren, was für kontextfreie Sprachen insgesamt relevant ist. Kämmerer entwickelt ein Verfahren speziell zum Aufbrechen von Formeln, die Additionsoperatoren $+$ und $-$, Multiplikationsoperatoren $*$ und $/$, Klammern sowie Funktionsaufrufe enthalten können. Eine Erweiterungsmöglichkeit auf weitere Stufen von Operatoren wird in der Habilitationsschrift nicht thematisiert; sie ist für das angestrebte Ziel nicht erforderlich. Eine Erweiterungsmöglichkeit ist auch nicht offensichtlich, da Additions- und Multiplikationsoperatoren uneinheitlich behandelt werden.

Man vergleiche zum Beispiel die Begrenzerpaarmethode mit Gewichtsvektor und einem Keller nach Edsger W. Dijkstra (1961), bei der alle Operatoren in gleicher Weise bearbeitet werden [Ker73, S. 455 ff.]. Das von Kämmerer entwickelte Verfahren kennt keine Unterscheidung von Zahlen- und Operatorenkeller. Sowohl Operatoren (kodiert über so genannte Anmerkungen) als auch Variablen sind stattdessen im automatischen Gedächtnis enthalten (in den Zellen vom Dienst).

Die gegebene Formel wird einmal von links nach rechts durchlaufen. Das aktuell eingelesene Zeichen bestimmt den Fortgang der Arbeiten. Mitunter ist eine Vorausschau um ein weiteres Zeichen erforderlich. Das Ausführen von Operationen wird, falls notwendig, verzögert. Die dafür erforderlichen Angaben werden in das automatische Gedächtnis gebracht und ihm später wieder entnommen. Dieser Speicher flexibler Größe wird ökonomisch verwaltet. Das Verfahren besitzt einen (wünschenswerten) linearen Zeitaufwand mit der Eingabelänge.

Friedrich L. Bauer geht auf Kämmerers Arbeit wie folgt ein: „Nachzutragen wäre, daß auch in der am 21.4.1958 eingereichten Habilitationsschrift von Wilhelm Kämmerer [...] gewisse Parallelen zum Kellerprinzip zu finden sind. Es ist zu vermuten, daß Kämmerer der Dresdner Vortrag von Samelson (1955) entgangen war. Von der am 30. März 1957 eingereichten Patentanmeldung von Bauer und Samelson für eine ‚formelgesteuerte Maschine' konnte Kämmerer natürlich noch nicht wissen" [Bau04, S. 238]. N. Joachim Lehmann sieht es folgendermaßen: „Erste Überlegungen dazu wurden von K. Samelson 1955 in Dresden angedeutet. Die endgültige Fassung des Kellerprinzips haben dann W. Kämmerer (1958, 1959) sowie K. Samelson und F. L. Bauer (1959) unabhängig voneinander entwickelt und publiziert" [Leh04, S. 257].

7 Weiterer Ausbau und Ausblick

Kämmerer beschreibt im §9 seiner Habilitationsschrift Ideen zu Sprachelementen höherer Programmiersprachen. Konkret geht er auf die folgenden Konstrukte ein:

- unbedingter Sprung,
- Alternative, Wahrheitswerte,
- Schleifen, Index-gebunden,
- Schleifen, ohne Bindung an einen Index,
- öffnendes und schließendes Symbol,
- Summenzeichen,
- Matrizen.

Er stellt fest, dass „für derartige strukturelle Fragen in der Schreibweise noch kein Analogon zu dem mathematischen Formelbild erwachsen ist" [Käm58, S. 55]. Für einige dieser

Konstrukte schlägt er eine Realisierung mithilfe des automatischen Gedächtnisses vor. Ein bedeutender Meilenstein in der Programmiersprachenentwicklung war dann ALGOL 60. An der Entwicklung dieser Programmiersprache war Kämmerer nicht beteiligt [Bau04, Leh04].

Kämmerer nahm eine Zusammenfassung seiner Habilitationsschrift in ein Lehrbuch auf [Käm60, S. 289 f.]. Im Berichtsband „Informatik“ der Jahresversammlung 1971 der Leopoldina gibt Kämmerer (als deren Mitglied) einen breit angelegten Überblick über das Thema „Algorithmische Sprachen“, in dem er u. a. den Zusammenhang zwischen Kellerautomaten und kontextfreien Sprachen thematisiert [Käm72]. Dabei erwähnt Kämmerer seine eigenen Untersuchungen von 1958 nicht einmal in einer Fußnote. Diese Form der Zurückhaltung ist beeindruckend.

Dem Computerpionier Wilhelm Kämmerer (gemalt von Konrad Zuse!) gelangen nicht nur bei der Computerentwicklung, sondern auch auf dem Gebiet der Programmiersprachen und zu einer wichtigen Struktur der Informatik bemerkenswerte Leistungen [JG95, Weba].

Literatur

[Bau04] F. L. Bauer. Die Algol-Verschwörung. In H. D. Hellige, Hrsg., *Geschichten der Informatik. Visionen, Paradigmen, Leitmotive*, Seiten 237–254. Springer-Verlag, Berlin, Heidelberg, New York, 2004.

[JG95] J. Jänike und F. Genser. *Die Vergangenheit der Zukunft. Deutsche Computerpioniere*. Düsseldorf, 2. Auflage, 1995.

[Käm58] W. Kämmerer. Ziffern-Rechenautomat mit Programmierung nach mathematischem Formelbild. Habilitationsschrift, Friedrich-Schiller-Universität Jena, http://www.db-thueringen.de/servlets/DocumentServlet?id=22616, 1958.

[Käm60] W. Kämmerer. *Ziffernrechenautomaten*. Band 1 von *Elektronisches Rechnen und Regeln*. Akademie-Verlag Berlin, 1960.

[Käm72] W. Kämmerer. Algorithmische Sprachen. In J.-H. Scharf, Hrsg., *Informatik*, Nova Acta Leopoldina 206, Bd. 37/1, Seiten 403–417. Johann Ambrosius Barth, Leipzig, 1972.

[Ker73] I. O. Kerner. *Numerische Mathematik und Rechentechnik. Teil 2/2*. Mathematisch-Naturwissenschaftliche Bibliothek, Bd. 47/2. Teubner, Leipzig, 1973.

[Ker06] I. O. Kerner. OPREMA und ZRA 1 – die Rechenmaschinen der Firma Carl Zeiss Jena. In F. Naumann und G. Schade, Hrsg., *Informatik in der DDR – eine Bilanz*, Lecture Notes in Informatics – Thematics, Seiten 147–177. Bonn, 2006.

[Leh04] N. J. Lehmann. ALGOL im Ostblock und der Weg zu Systemen von Programmiersprachen. In H. D. Hellige, Hrsg., *Geschichten der Informatik. Visionen, Paradigmen, Leitmotive*, Seiten 255–273. Springer-Verlag, Berlin, Heidelberg, New York, 2004.

[RGF+04] R. Rojas, C. Göktekin, G. Friedland, M. Krüger, L. Scharf, D. Kuniß und O. Langmack. Konrad Zuses Plankalkül. Seine Genese und eine moderne Implementierung. In H. D. Hellige, Hrsg., *Geschichten der Informatik. Visionen, Paradigmen, Leitmotive*, Seiten 215–235. Springer-Verlag, Berlin, Heidelberg, New York, 2004.

[Weba] Computerpioniere. http://www.superbrain.eu/Pioniere.htm.

[Webb] Der erste Informatiker in der Tafelrunde.
http://www.uni-jena.de/Mitteilungen/PM130913_Kämmerer_Tafel.html.

[Webc] Erster DDR-Computer hatte 500 Kilometer lange Leitungen.
http://www.heise.de/ix/meldung/Erster-DDR-Computer-hatte-500-Kilometer-lange-Leitungen-1834998.html.

[Webd] Erster DDR-Computer. Mit diesem Monstrum konnte man rechnen.
http://www.spiegel.de/einestages/erster-ddr-computer-oprema-a-951147.html.

[Webe] Java-Implementierung. https://cms.rz.uni-jena.de/minet_multimedia/autoGed.zip.

[Win10] J. F. H. Winkler. Konrad Zuse und die Optik-Rechenmaschine in Jena. *LOG IN*, Heft Nr. 166/167, Seiten 132–136, 2010.

Die angegebenen Internetquellen wurden zuletzt am 1. März 2015 geprüft.

Keller im Übersetzerbau

Reinhard Wilhelm

Fachrichtung Informatik
Universität des Saarlandes
66041 Saarbrücken
wilhelm@cs.uni-saarland.de

Kurzfassung: Keller sind im Übersetzerbau allgegenwärtig: *Deterministische Kellerautomaten* werden zur Syntaxanalyse verwendet. Jede Programmiersprache mit Rekursion benutzt einen *Laufzeitkeller*, um Inkarnationen von rekursiven Prozeduren oder Funktionen effizient zu verwalten. Sprachspezifische *virtuelle Kellermaschinen* wurden und werden entworfen, um die semantische Lücke zwischen höheren Programmiersprachen und Zielmaschinen zu überbrücken. Solche Kellermaschinen wurden sogar mehrfach in Hardware realisiert. Um das Anlegen und das Freigeben von Funktionsinkarnationen effizient zu unterstützen, werden in etlichen Architekturen *Registerkeller* angeboten.

1 Einleitung

Keller (Stapel, engl. stack) sind im Übersetzerbau allgegenwärtig: Sie sind auf natürliche Weise mit Rekursion verbunden, welche in der Syntax – Struktur von Programmen – und in der Semantik – rekursive Sprachkonstrukte wie Prozeduren, Funktionen und Klauseln – in Programmiersprachen vorkommt. Gemäß dem rekursiven Aufbau von Programmen werden (deterministische) *Kellerautomaten* zur Syntaxanalyse verwendet. Für die Implementierung von Rekursion benutzt man einen *Laufzeitkeller*, um Inkarnationen von rekursiven Prozeduren, Funktionen oder Klauseln effizient zu verwalten. Da die Organisation des Laufzeitkellers inklusive der Realisierung von Sichtbarkeitsregeln kompliziert sein kann und konventionelle Rechnerarchitekturen sie nicht unterstützen, wurden und werden sprachspezifische *virtuelle Kellermaschinen* entworfen, um die semantische Lücke zwischen höheren Programmiersprachen und Zielmaschinen zu überbrücken. Solche Kellermaschinen wurden sogar mehrfach in Hardware realisiert. Auch in konventionellen Architekturen werden Registerorganisationen angeboten, um das Anlegen und das Freigeben von Funktionsinkarnationen effizient zu unterstützen.

2 Kellerautomaten zur Syntaxanalyse

Die meisten Programmiersprachen haben syntaktische Strukturen, welche eine kontext-freie Beschreibung ohne Mehrdeutigkeiten und sogar eine deterministische Analyse erlauben.

LR-Syntaxanalyse [Knu65] ist das mächtigste deterministische Syntaxanalyseverfahren. Effiziente Verfahren und zugehörige Generierungsverfahren und -werkzeuge für leicht eingeschränkte Klassen von Grammatiken existieren, z. B. LALR-Analysatorgeneratoren [DeR71, Pen86]. Damit scheint die LR- oder LALR-Syntaxanalyse das Mittel der Wahl zu sein. Unglücklicherweise sind die Diagnose und die Behandlung von Syntaxfehlern durch LR-Analyseverfahren sehr kompliziert und nicht sehr effektiv [PD78, WSH13]. Deshalb werden in vielen Übersetzern Top-down-Analysatoren verwendet [RS70, SSS90].

3 Kellerartige Speicherverwaltung

Keller unterstützen eine besonders effiziente Speicherverwaltung. Meist ist in jeder Ausführungssituation nur genau so viel Speicher belegt, wie benötigt wird. Speicherbelegung und -freigabe sind sehr effizient; eine *push*-Operation reicht, um einen Speicherbereich zu belegen, und eine *pop*-Operation reicht, um einen Speicherbereich freizugeben.

3.1 Analyse der maximalen Kellertiefe

Ein interessantes Problem aus der Praxis ist der Konflikt zwischen der im Prinzip unbeschränkten Tiefe von Kellern in der Implementierung von Systemen und den tatsächlich vorhandenen Ressourcenbeschränkungen. Programmiersprachen kennen nur im Prinzip unbeschränkte oder gar keine Rekursion. Jede in der Realität verwendete Hardware hat aber beschränkte Ressourcen, insbesondere beschränkten Speicher. Ein beschränkt großer Speicher limitiert die Tiefe des Laufzeitkellers und damit die maximale Tiefe der Rekursion. Das ist bei PCs oder Hochleistungsrechnern kein Problem, da die Speicher heute genügend groß für die meisten Anwendungen sind. Viele eingebettete Systeme haben aus Kostengründen sehr eingeschränkte Ressourcen. Das relevante Problem ist dann, herauszufinden, wie die maximale Ausdehnung des Laufzeitkellers für ein Programm ist. Wenn für den Laufzeitkeller nicht genügend Speicherplatz zur Verfügung steht, können bei tiefen Rekursionen sich an den Laufzeitkeller anschließende Speicherbereiche überschrieben werden. Dies kann schreckliche Konsequenzen haben. Ein solcher Fehler hat erst kürzlich Toyota 1,2 Milliarden US Dollar gekostet [Dun13]. Wenn der Entwickler die maximale Ausdehnung des Laufzeitkellers kennt, kann er den Speicher für den Laufzeitkeller so auslegen, dass sowohl kein Speicher verschwendet wird als auch der Laufzeitkeller bei der Anlage von neuen Kellerrahmen keine anderen Speicherbereiche überschreibt.

Das Problem, die maximale Ausdehnung von System- und Nutzerkellern zu bestimmen, ist nicht besonders schwer und wird zuverläßig von geeigneten Werkzeugen erledigt [Abs, KF14, RRW05].

4 Laufzeitkeller

Höhere Programmiersprachen bieten meist Rekursion an, für deren Implementierung ein *Laufzeitkeller* (run-time stack) benötigt wird. Der Laufzeitkeller enthält zu jedem Zeitpunkt ein Folge von *Kellerrahmen* für die aktuell lebendigen Inkarnationen von rekursiven Objekten wie Prozeduren, Funktionen oder Klauseln. In den Kellerrahmen werden lokale Größen wie Parameter oder lokale Variablen in statisch relativ zu einem Register adressierten Zellen abgelegt. Außerdem wird ein *dynamischer Verweis* auf den Rahmen des dynamischen Vorgängers abgespeichert, der eine effiziente Realisierung der Speicherfreigabe (pop) erlaubt. Ein *statischer Verweis* erlaubt den effizienten Zugriff auf nichtlokale Größen in umfassenden Gültigkeitsbereichen.

Für die Implementierung von Rekursion mithilfe eines Laufzeitkellers braucht man Befehle oder Befehlsfolgen, die folgende Abfolge von Aktionen durchführen:

1. Anlegen eines Kellerrahmens,
2. Übergabe von Argumenten,
3. Reservierung von Platz für lokale Größen und
4. Sprung an die Anfangsadresse der aufgerufenen Funktion und Merken der Rücksprungadresse.

Nach der Terminierung der aufgerufenen Funktion muss

1. das Ergebnis zurückgegeben und
2. der Kellerrahmen aufgegeben werden.

Im Aufbau der Kellerrahmen müssen die Aufrufregeln (calling conventions) der Programmiersprache, des Betriebssystems und der Zielarchitektur beachtet werden [LR04].

Am oberen Ende solcher Kellerrahmen werden lokale rekursive Berechnungen durchgeführt, z. B. die Auswertung arithmetischer oder logischer Ausdrücke oder die Unifikation zwischen Termen. Am oberen Ende eines Kellerrahmens pulsiert also ein weiterer *kleiner* Keller, der *Auswertungskeller*. Insgesamt ergibt das die in Abbildung 1 dargestellte Struktur.

5 Kellermaschinen

Kellermaschinen (stack machines), d. h. Architekturen mit kellerartigen Speichern wurden und werden entworfen, um die *semantische Lücke* zwischen einer höheren Programmiersprache und der Maschinensprache des Zielrechners zu überbrücken. Eine Kellermaschine als Zielmaschine erleichtert die Codeerzeugung; das ist wichtig für die Implementierung der Programmiersprache durch einen Übersetzer. Deshalb gibt es typischerweise Instruktionen, welche die oben aufgelisteten Aktionen auf dem Laufzeittkeller unterstützen.

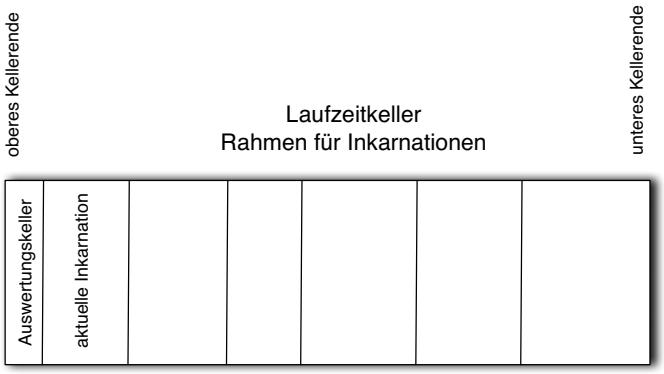


Abbildung 1: Ein Laufzeitkeller mit (kleinem) Auswertungskeller. Der Laufzeitkeller dient zur Unterstützung von Rekursion, der Auswertungskeller zur Auswertung oder zur Reduktion von Ausdrücken oder zur Unifikation von Termen.

Kellermaschinen sind sowohl in Software als auch in Hardware realisiert worden. Wir stellen im Folgenden einige vor.

5.1 Virtuelle Kellermaschinen – Motivation und Beispiele

Kellermaschinen, welche in Software realisiert wurden, werden meist *virtuelle* oder *abstrakte* Kellermaschinen genannt. Sie wurden häufig entworfen, um Implementierungen von Programmiersprachen leichter zu portieren. Gemeinsam ist den meisten dieser virtuellen Maschinen:

- Sie bieten Befehle zur Organisation eines Laufzeitkellers an, also für die in Abschnitt4 aufgelisteten Aktionen.
- Sie bieten Befehle zur Auswertung von Ausdrücken an, welche oben auf einem *Auswertungskeller* stattfindet. Operanden der arithmetischen und logischen Befehle und Vergleiche sind immer die obersten Zellen dieses Auswertungskellers oder Direktoperanden, also Konstanten. Deshalb enthalten die meisten dieser Befehle keine Operandenadressen. Die Programme für diese virtuellen Maschinen sind daher sehr kompakt, und dies ist günstig für den Speicherverbrauch und die notwendige Bandbreite beim Versenden und Herunterladen solcher Programme.
- Der Speicher ist in einen Laufzeitkeller und in eine Halde aufgeteilt.
- Der kellerorientierte Befehlssatz braucht nur wenige Register zur Implementierung der virtuellen Maschine. Deshalb sind i.A. alle Registermaschinen mögliche Zielarchitekturen, und die Emulation der virtuellen Kellermaschine auf einer Registermaschine ist meist leicht.

5.2 Virtuelle Kellermaschinen für imperative Programmiersprachen

Algol Object Code Schon bald nach der Definition von ALGOL 60 beschrieben Brian Randell und Lawford Russel in [RR64] den *Algol Object Code (AOC)*, einen Befehlssatz für eine virtuelle Kellermaschine. Dieser Code erlaubte eine einfache Übersetzung von ALGOL 60-Programmen. Er wurde im Whetstone KDF9 ALGOL 60-Übersetzer benutzt.

Die Pascal P-Maschine Die Motivation für den Entwurf der Pascal P-Maschine war die leichte Portierbarkeit des Pascal-Übersetzers. Urs Amman definierte die P-Maschine zur Übersetzung von Pascal [NAJ+81]. Ihr sauberer und einfacher Entwurf wurde zum Vorbild für viele spätere virtuelle Maschinen wie etwa die Java Virtual Machine (JVM).

Die erste Implementierung geschah auf einer CDC 6800. Mit Hilfe der P-Maschine wurde der Züricher Pascal-Übersetzer durch einen Bootstrap-Prozess auf viele andere Zielarchitekturen portiert.

5.3 Virtuelle Maschinen für vernetzte objekt-orientierte Programmiersprachen

Vernetzte, in objekt-orientierten Programmiersprachen realisierte Systeme tauschen neben Programmen (Klassen) auch Daten aus. Im Allgemeinen werden diese vernetzten Systeme auf verschiedenen Zielarchitekturen realisiert sein. Deshalb müssen Programme und Daten plattformunabhängig dargestellt sein. Bei der Pascal P-Maschine waren Maschinenbefehle auch schon plattformunabhängig, in einem festen Befehlsformat, dargestellt. Datenstrukturen waren durch die Speicherbelegung und den erzeugten Maschinencode realisiert. Wenn Daten zwischen vernetzten Systemen ausgetauscht werden sollen, müssen sie anders dargestellt werden, z. B., indem die Daten zusammen mit einer Beschreibung der Datenstruktur, *Metadaten*, verschickt werden.

Die beiden wichtigsten virtuellen Maschinen für objekt-orientierte Programmiersprachen und vernetzte Systeme sind die virtuelle Maschine für Java [Ora99] und die Microsoft common language infrastructure (CLI) [BP02].

Die virtuelle Java-Maschine (JVM) Die virtuelle Maschine für Java (JVM) [Gos95, LYBB13] besitzt einen Laufzeitkeller, der im Wesentlichen genauso aufgebaut ist wie oben beschrieben; zu jedem Zeitpunkt der Programmausführung enthält er einen Kellerrahmen für jede Methode, deren Ausführung begonnen, aber noch nicht abgeschlossen wurde. In diesem Rahmen liegen Werte der Funktionsparameter, Werte lokaler Variablen und Referenzen in die Halde. Am oberen Ende des Rahmens der aktuellen Methode befindet sich der lokale Auswertungskeller.

Zusätzliche Speicherbereiche sind die Halde, welche Felder (arrays) und Objekte enthält, einen Methodenbereich für Programmcode und einen Konstantenbereich.

Die Microsoft Intermediate Language Die Microsoft Intermediate Language (MSIL) [Lid02] für die Microsoft common language infrastructure (CLI) ist ebenfalls kellerorientiert, aber nur bezüglich der Auswertung von Ausdrücken. Die Aktivierung einer Methode führt zum Anlegen eines lokalen Speicherbereichs. Dieser Speicherbereich muss zugreifbar sein, wenn die Methode aktiv ist. Eine Implementierung der MSIL wird die lokalen Speicherbereiche wohl kellerartig verwalten. Die MSIL verfügt aber über keine Befehle zur Verwaltung eines Laufzeitkellers.

5.4 Virtuelle Kellermaschinen für funktionale Programmiersprachen

Höhere funktionale Programmiersprachen bieten einige Konzepte an, welche ihre Implementierung nichttrivial machen; *höhere Funktionen*, das sind Funktionen, welche wiederum Funktionen als Argumente und/oder Ergebnisse haben können, *verzögerte Auswertung* (lazy evaluation), *unterversorgte* und *überversorgte Funktionsaufrufe*. Die Mutter aller virtuellen Maschinen für funktionale Programmiersprachen ist Peter Landins SECD-Maschine [Lan64].

Eine Architektur zur Implementierung von höheren funktionalen Programmiersprachen mit verzögerter Ausführung wird in [WS10] vorgestellt. Sie enthält zwei Speicherbereiche: der *Laufzeitkeller* von der Struktur aus Abb. 1 dient wieder zur Unterstützung der Rekursion. Auf ihm werden Verweise auf Argumente und lokale Werte abgespeichert. Diese Verweise zeigen in die *Halde*, auf welcher die Werte abgespeichert werden. Für Funktionsargumente mit verzögerter Auswertung werden dort *Abschlüsse* gespeichert, das sind Ausdrücke zusammen mit den Umgebungen, in denen sie ausgewertet werden müssen.

Ein interessantes Problem ergibt sich durch die Möglichkeit, Funktionen auf weniger oder mehr Argumente anzuwenden als ihre Stelligkeit angibt, also Funktionen unter- oder überzuversorgen. In den Kellerrahmen des Laufzeitkellers möchte man alle (Verweise auf) Argumente und lokale Größen mit statischen Relativadressen adressieren. Die Zahl der lokalen Größen ist zur Übersetzungszeit bekannt. Sie liegen aber notwendigerweise oberhalb des Bereichs, in dem die Argumente liegen, weil diese schon vom Aufrufer der Funktion abgelegt werden. Die Zahl der Argumente kann aber eben von Aufruf zu Aufruf verschieden sein. Deshalb hätten lokale Größen keine statischen Relativadressen mehr. Dieses Problem wird dadurch gelöst, dass man die Reihenfolge der Argumente umdreht, den *Kellerzeiger* (framepointer) auf das erste Argument zeigen lässt und sowohl Argumente wie lokale Größen relativ zum Kellerzeiger adressiert. Der Kellerzeiger bewegt sich allerdings, sodass man keine festen Relativadressen mehr haben kann. Die Relativadressen verändern sich während der Auswertung einer Funktion. Deshalb simuliert der Codeerzeuger diese Bewegung des Kellerzeigers und vergibt Relativadressen, die von der Programmstelle abhängen.

5.5 Virtuelle Prolog-Maschinen

Virtuelle Maschinen für die logikbasierte Programmiersprache Prolog dienen der portablen Implementierung von Prolog [WS10]. Prolog ist weit entfernt von Zielarchitekturen, die im Allgemeinen für imperative Sprachen entworfen wurden und werden. Insbesondere Resolution, Unifikation und Backtracking werden durch diese virtuellen Maschinen implementiert.

Das rekursive Sprachkonstrukt, welches durch einen Laufzeitkeller unterstützt wird, ist die *Klausel*. Man kann sie als Prozedur auffassen, bei deren Aufruf die aktuellen und die formalen Parameter unifiziert werden. Für die rekursiven Aufrufe von Klauseln braucht man wieder einen Laufzeitkeller, ähnlich dem in Abb. 1. Allerdings wird der kleine Auswertungskeller jetzt zur Unifikation von Termen gebraucht.

Eine in imperativen oder funktionalen Sprachen nicht vorhandene Komplikation ist der *Nichtdeterminsmus* in der Logik, nämlich die Unbestimmtheit, in welcher Reihenfolge die Klauseln für ein Prädikatssymbol für die Resolution ausprobiert werden sollen. Dieser Nichtdeterminismus wird in Prolog durch Festlegung auf die *Tiefensuche* (depth-first search) aufgelöst; wenn sich eine ausgewählte Alternative irgendwann als falsch herausstellt, weil eine Unifikation bei einem Aufruf fehlschlägt, muss zurückgesetzt werden bis zu der ersten Klausel, welche noch eine weitere offene Alternative hat. Diese heißt der aktuelle *Rücksetzpunkt*. Um dieses Zurücksetzen effizient zu realisieren, wird in den Kellerrahmen des Laufzeitkellers ein zusätzlicher Verweis auf den Rahmen abgespeichert, der zu dieser Klausel gehört. Nach dem erfolgreichen Abarbeiten einer Klausel wird der oberste Kellerrahmen aufgegeben, beim Zurücksetzen eventuell eine ganze Folge von Rahmen am oberen Kellerende. Es gibt zwei zusätzliche Speicherbereiche in diesen virtuellen Prolog-Maschinen, eine *Halde*, auf welcher die Terme abgespeichert werden, und einen weiteren Keller, genannt die *Spur* (trail), auf welchem die seit Anlegen eines Rücksetzpunkts jeweils eingegangenen Bindungen vermerkt werden. Diese müssen im Falle des Rücksetzens wieder rückgängig gemacht werden.

David H.D. Warren hat zwei verschiedene virtuelle Maschinen für die Implementierung von Prolog entworfen, die sich in der Behandlung der Unifikation unterscheiden. Die Warren Abstract Machine WAM, [War83, Aït91], liegt den meisten Prolog-Implementierungen zugrunde.

6 Kellermaschinen in Hardware

Sprachorientierte Kellermaschinen wurden in Hardware realisiert, um höhere Leistung gegenüber Softwarerealisierungen zu erzielen. Diese in Hardware realisierten Kellermaschinen hatten allerdings Probleme, mit der Leistungssteigerung der traditionellen Registermaschinen mitzuhalten. Deshalb sind fast alle dieser Kellermaschinen bald vom Markt verschwunden.

Burroughs ALGOL 60-Maschinen, B5000, B6500 Die Burroughs Corporation entwarf in den 1960er- und 1970er-Jahren große Rechner mit Kellerarchitektur zur effizienten Übersetzung und Ausführung von ALGOL 60-Dialekten. Effiziente Übersetzung durch einen 1-Pass-Übersetzer wurde ermöglicht, indem, wie später auch in Pascal, obligate Vorwärtsdeklarationen in die Sprache aufgenommen wurden. Effiziente Ausführung, insbesondere von Prozedureintritt und -verlassen, wurde durch in Hardware realisierte Maschinenbefehle erreicht. Arithmetische und logische Operationen wurden, wie bei Kellermaschinen üblich, jeweils auf Operanden oben auf dem Keller ausgeführt. Sie wurden dadurch beschleunigt, dass die obersten Kellerzellen immer mit Registern identifiziert wurden.

Die Pascal Microengine von Western Digital Die Pascal MicroEngine [Wes79] war eine Serie kommerziell nicht sehr erfolgreicher Microcomputer, auf denen UCSD-Pascal effizient übersetzt und ausgeführt werden konnte. Die Befehle der P-Maschine wurde in Microcode interpretiert.

7 Registerkeller in Zielarchitekturen

In diesem Abschnitt möchten wir zeigen, wie Rekursion in konventionellen Architekturen unterstützt wird. Die erste Technik, ein großer *Registerkeller* aufgeteilt in *Registerfenster*, unterstützt Rekursion von Funktionen, die zweite, ein kleiner *Registerkeller*, unterstützt die Auswertung von Ausdrücken.

7.1 Registerfenster in der Intel IA-64–Architektur

Die Intel IA-64–Architektur (Itanium) [MS03] ist ein Beispiel, wie man die in Abschnitt 5 aufgelisteten Aktionen effizient in Hardware unterstützen kann. Die IA-64-CPU hat 128 allgemein verwendbare Register (general purpose registers) für ganze Zahlen, von denen die letzten 96 wie ein Keller für die Implementierung des Laufzeitkellers zur Verfügung stehen. Dieser Registerkeller ist, wie bei der SPARC-Architektur [WG94], in Fenster aufgeteilt, welche einzelnen Funktionsinkarnationen zugeteilt sind.

Ein spezielles Register, der *aktuelle Rahmenzeiger* (current frame marker, CFM) zeigt auf das *Registerfenster* der aktuellen Funktionsinkarnation. Dieses Registerfenster besteht aus, in dieser Reihenfolge, dem *Eingabe*-, dem *lokalen* und dem *Ausgabebereich*. Im Eingabebereich stehen die Argumente dieser Inkarnation, im lokalen Bereich seine lokalen Objekte. In den Ausgabebereich kopiert die Funktion ihrerseits die Argumente für Funktionen, welche sie aufruft. Der Übersetzer bestimmt die Größe dieser Teilbereiche und gibt sie der `alloc`-Instruktion als Parameter mit. Die Registerfenster der SPARC-Architektur haben konstante Größe, 24 Register, aufgeteilt in Bereiche von jeweils 8 Registern. Die `br.call`-Instruktion des Itaniums schaltet um auf das Registerfenster der aufgerufenen Funktion, setzt dabei das CFM-Register auf das neue Fenster und benennt die Register des Fensters der aufgerufenen Funktion um. Die alten Registerbenennungen werden gemerkt, damit man sie bei der Rückkehr wiederherstellen kann.

Das klingt alles zu schön oder zu einfach, um wahr zu sein. Tatsächlich muss man Lösungen für gewisse Beschränktheiten der Hardware finden: Beim Itanium sind *maximal* 8 Parameter für Funktionen erlaubt. Hat eine Funktion mehr als 8 Parameter, werden überzählige aktuelle Parameter auf dem Laufzeitkeller abgelegt. Gleitpunktargumente werden in anderen Registern übergeben. Auch die beschränkte Zahl der Register stellt ein Problem dar, wenn die Rekursion zu tief wird. Dann tritt ein Mechanismus in Kraft, der die ältesten belegten Register in einem *Registerhintergrundspeicher* ablegt und damit Register für den nächsten Aufruf frei macht. Die abgelegten Inhalte werden wieder zurück in Register geladen, wenn die Funktion verlassen wird.

7.2 Die x87-Instruktionen der Intel-Prozessoren

William Kahan, Turing-Preisträger von 1989, entwarf für Intel einen Gleitpunkt-Koprozessor für x86-CPUs. Dessen Befehlssatz wurde Teil des Befehlssatzes späterer x86-Prozessoren. Er wurde als der *x87-Befehlssatz* bezeichnet. Er umfasst Basisoperationen auf Gleitpunktzahlen wie Addition, Subtraktion und Vergleiche, aber z. B. auch die tangens-Funktion. Die x87-Register bilden einen 8 Stufen tiefen Keller von ST(0) bis ST(7). Monadische und dyadische Operationen arbeiten oben auf diesem Keller, d. h. auf den Registern ST(0) und ST(1). Es gibt Befehle, um Werte oben auf den Keller zu laden oder von ihm zu entfernen.

Der Registerkeller lässt allerdings auch nicht kellerartigen Gebrauch zu und wird deshalb als *nichtstrikter Keller* bezeichnet. Es gibt Befehle, welche das oberste Register ST(0) als Akkumulator benutzen. Der zweite Operand kann in einer Speicherzelle oder einem der anderen Register liegen. Um auch nichtkommutative Operationen zu unterstützen, kann man diese Rollen vertauschen. Außerdem kann man sehr effizient die Inhalte von Registern vertauschen.

8 Zusammenfassung

Wir haben die vielen verschiedenen Rollen, in denen Keller im Übersetzerbau auftreten, vorgestellt. Sie dienen zur syntaktischen Analyse von Programmen, als Laufzeitkeller zur Realisierung von Rekursion und zur Auswertung von Ausdrücken. Auf Registermaschinen treten sie in der Form von Registerkellern auf, um die Implementierung des Laufzeitkellers und die Ausdrucksauswertung effizient zu unterstützen.

Literatur

[Abs] AbsInt. StackAnalyzer: Stack Usage Analysis.
http://www.absint.com/stackanalyzer/index.htm.

[Aït91] Hassan Aït-Kaci. *Warren's Abstract Machine: A Tutorial Reconstruction*. MIT Press, 1991.

[BP02] Don Box und Ted Pattison. *Essential. NET: The common language runtime*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[DeR71] Frank DeRemer. Simple LR(k) Grammars. *Commun. ACM*, 14(7):453–460, 1971.

[Dun13] Michael Dunn. Toyota's killer firmware: Bad design and its consequences, 2013.

[Gos95] James Gosling. Java Intermediate Bytecode. In Michael D. Ernst, Hrsg., *Proceedings ACM SIGPLAN Workshop on Intermediate Representations (IR'95), San Francisco, CA, USA, January 22, 1995*, Seiten 111–118. ACM, 1995.

[KF14] Daniel Kästner und Christian Ferdinand. Proving the Absence of Stack Overflows. In A. Bondavalli und F. Di Giandomenico, Hrsg., *SAFECOMP 2014*, Jgg. 8666 of *LNCS*, Seiten 202–213. Springer, 2014.

[Knu65] D.E. Knuth. On the Translation of Languages from Left to Right. In *Information and Control* **8**, Seiten 607–639, 1965.

[Lan64] Peter J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308–320, 1964.

[Lid02] Serge Lidin. *Inside Microsoft. NET IL assembler*. Microsoft Press, 2002.

[LR04] Christian Lindig und Norman Ramsey. Declarative Composition of Stack Frames. In Evelyn Duesterwald, Hrsg., *Compiler Construction*, LNCS, Bd. 2985, Seiten 298–312, 2004.

[LYBB13] Tim Lindholm, Frank Yellin, Gilad Bracha und Alex Buckley. *The Java Virtual Machine Specification*. Addison-Wesley, 2013.

[MS03] Cameron McNairy und Don Soltis. Itanium 2 processor microarchitecture. *IEEE Micro*, 23(2):44–55, 2003.

[NAJ+81] K. V. Nori, Urs Ammann, Kathleen Jensen, H. H. Nageli und Christian Jacobi. Pascal-P Implementation Notes. In D. W. Barron, Hrsg., *Pascal - The Language and its Implementation*, Seiten 125–170. John Wiley, 1981.

[Ora99] Oracle Corp. *The Structure of the Java Virtual Machine*, 1999.

[PD78] Thomas J. Pennello und Frank DeRemer. A Forward Move Algorithm for LR Error Recovery. In Alfred V. Aho, Stephen N. Zilles und Thomas G. Szymanski, Hrsg., *ACM Symposium on Principles of Programming Languages*, Seiten 241–254. ACM Press, 1978.

[Pen86] Thomas J. Pennello. Very fast LR parsing. In Richard L. Wexelblat, Hrsg., *ACM SIGPLAN Symposium on Compiler Construction*, Seiten 145–151. ACM, 1986.

[RR64] Brian Randell und Lawford J. Russell. *ALGOL-60 Implementation*. Academic Press, 1964.

[RRW05] John Regehr, Alastair Reid und Kirk Webb. Eliminating stack overflow by abstract interpretation. *ACM Trans. Embedded Comput. Syst.*, 4(4):751–778, 2005.

[RS70] Daniel J. Rosenkrantz und Richard Edwin Stearns. Properties of Deterministic Top-Down Grammars. *Information and Control*, 17(3):226–256, 1970.

[SSS90] S. Sippu und E. Soisalon-Soininen. *Parsing Theory. Vol. 2: LR(k) and LL(k) Parsing*. Springer, 1990.

[War83] David H.D. Warren. An Abstract PROLOG Instruction Set. Technical Note 309, SRI International, 1983.

[Wes79] Western Digital. *WD 90 Pascal Microengine Reference Manual*, 1979.

[WG94] David L Weaver und Tom Gremond. *The SPARC architecture manual*. PTR Prentice Hall Englewood Cliffs, NJ 07632, 1994.

[WS10] Reinhard Wilhelm und Helmut Seidl. *Compiler Design—Virtual Machines*. Springer, 2010.

[WSH13] Reinhard Wilhelm, Helmut Seidl und Sebastian Hack. *Compiler Design - Syntactic and Semantic Analysis*. Springer, 2013.

Die Analyse von Kellerstrukturen: Eine Reise durch 50 Jahre Forschung

Wolfgang Thomas

Lehrstuhl Informatik 7
RWTH Aachen University
Ahornstr. 55, 52074 Aachen
thomas@informatik.rwth-aachen.de

Kurzfassung: Vor 50 Jahren bewies J. R. Büchi, dass die Menge der erreichbaren Kellerinhalte eines Kellerautomaten eine reguläre Sprache bildet. Nur 5 Jahre später eröffnete M. O. Rabin mit seiner Theorie endlicher Automaten auf unendlichen Bäumen eine weiter greifende Perspektive, die in neuester Zeit zu überraschend starken algorithmischen Ergebnissen geführt hat, unter anderem für Systeme mit geschachtelten Kellern. Wir geben eine informelle Darstellung dieser Entwicklung und skizzieren aktuelle Forschungsfragen.

1 Einleitung

Der Begriff des Kellerautomaten ist eng verbunden mit der Etablierung der Wissenschaft Informatik, mehr noch als der Begriff der Turingmaschine. Turing hat seine Maschinen als Beitrag zur Grundlagenforschung der Mathematik eingeführt; er arbeitete als Logiker und zeigte, dass ein zentrales Problem der mathematischen Logik (die Bestimmung der allgemeingültigen Formeln erster Stufe) keine algorithmische Lösung hat. Die Idee des Kellerspeichers und des Kellerautomaten ist dagegen im Zuge der Entwicklung der Programmiersprachen und ihrer Implementierung entstanden, insbesondere im Zusammenhang mit der Auswertung arithmetischer Ausdrücke und mit der Ausführung rekursiver Prozeduren.

Doch auch darüber hinaus hat sich der Begriff des Kellerautomaten und des Kellersystems als Schlüssel für starke algorithmische Ergebnisse der Informatik erwiesen. In diesen Forschungen geht es um die Analyse des Verhaltens von Systemen mit einem Keller als Speicher. Hier steht allerdings nicht die Frage im Vordergrund, welche Eingabewörter das System „akzeptiert“ , also welche Sprache es definiert, sondern eher die Frage, welche Kellerinhalte erreichbar sind. Da es i. Allg. unendlich viele Kellerinhalte gibt, ist dieses Erreichbarkeitsproblem schwieriger als im Fall der endlichen Automaten.

Dass dennoch eine algorithmische Lösung möglich ist, hat Büchi vor genau 50 Jahren gezeigt. Dieses Resultat ist der Beginn einer beeindruckenden Erfolgsgeschichte, die bis heute reicht und noch immer interessante Perspektiven bereithält. Wir versuchen, diese Geschichte ohne eine Überlast an formaler Darstellung nachzuzeichnen. Dabei verfolgen

wir zwei Zweige der Verallgemeinerung des Ergebnisses von Büchi. Zunächst tritt an die Stelle einer Erreichbarkeitsaussage eine beliebige Bedingung, die sich in einer recht mächtigen Logik-Sprache (der „monadischen Logik zweiter Stufe“) ausdrücken lässt, wobei die Erreichbarkeitsbedingung als Spezialfall erfasst ist. Zweitens werden weit komplexere Strukturen als nur Kellersysteme betrachtet.

Wir gehen in drei Etappen vor: Zunächst stellen wir Büchis Resultat von 1964 über seine „regulären kanonischen Systeme“ vor und erläutern den Zusammenhang mit Kellersystemen. Dann schildern wir, in welcher Weise Rabin 1969 mit seinem Theorem über die Entscheidbarkeit der monadischen Theorie des binären Baumes weiter gehende Forschungen eröffnet hat – insbesondere die Möglichkeit, über unendlichen Strukturen wie den „Kellergraphen“ algorithmische Programmverifikation (im Sinne des „Model-Checking“) zu etablieren. Im abschließenden Abschnitt wird die von Muchnik, Walukiewicz, Caucal und anderen in den 1990er-Jahren angestoßene und bis heute andauernde Entwicklung diskutiert, diese Theorie auf eine sehr reichhaltige Klasse weiterer unendlicher Modelle auszudehnen.

2 Kellersysteme und Büchis reguläre kanonische Systeme

Unter einem *Kellersystem* verstehen wir hier eine Variante des Kellerautomaten, bei der auf das Eingabealphabet verzichtet wird. Es geht in einer Transition also nur um die Veränderung von Kontrollzustand und Kellerinhalt. Wir schreiben Q für die Menge der Kontrollstände und Γ für das Kelleralphabet (zu dem auch das Kellerbodensymbol $\perp$ gehört). Ein Kellersystem $\mathcal{K}$ hat Transitionen der Form $pa \to qv$ wobei $p, q \in Q, a \in \Gamma, v \in \Gamma^*$, mit der Wirkung „in Zustand p mit oberstem Kellersymbol a ersetze a durch das Wort $v \in \Gamma^*$ und gehe in den Zustand q“ . Eine Konfiguration ist gegeben durch einen Kontrollzustand q und den Kellerinhalt, also ein Wort $w \in \Gamma^*\perp$; wir vereinbaren, dass die Kellerspitze am vorderen Ende von w liegt. Der Einfachheit halber notieren wir eine Konfiguration als Wort $qu \in Q\Gamma^*\perp$. Dann liefert die Transition $pa \to qv$ den Konfigurationsübergang $paw \vdash_{\mathcal{K}} qvw$ für beliebiges $w \in \Gamma^*\perp$. In diesem Sinne „sind“ Kellersysteme spezielle Präfix-Ersetzungssysteme, bestehend aus endlich vielen Regeln $u \to v$, die jeweils von einem Wort uw den Übergang nach vw erlauben.

Wir schreiben

$$p_1w_1 \vdash^*_{\mathcal{K}} p_2w_2$$

wenn man von der Konfiguration p_1w_1 mit Transitionen des Kellersystems $\mathcal{K}$ in endlich vielen Schritten zur Konfiguration p_2w_2 gelangen kann. Zu dieser Erreichbarkeitsrelation stellen sich zwei fundamentale Fragen:

- Kann man algorithmisch entscheiden, ob $p_1w_1 \vdash^*_{\mathcal{K}} p_2w_2$ gilt?
- Wie lässt sich die Menge der von p_1w_1 aus erreichbaren Konfigurationen beschreiben?

Der Satz von Büchi [Büc64] beantwortet beide Fragen, sogar allgemein für Präfix- Ersetzungssysteme:

1. *Für Präfix-Ersetzungssysteme ist das Erreichbarkeitsproblem entscheidbar.*
2. *Aus einer regulären Wortmenge A („Axiome“) liefert ein Präfix-Ersetzungssystem als Menge R der erreichbaren Wörter wieder eine reguläre Sprache (und man kann aus einem endlichen Automaten für A einen solchen für R konstruieren).*

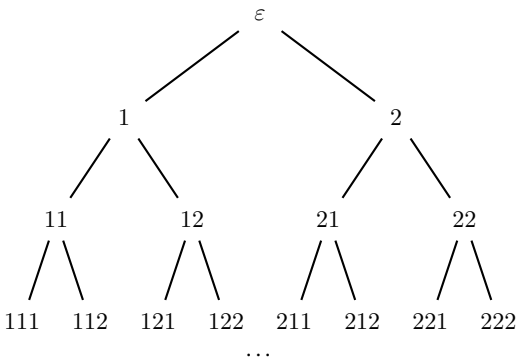
Für Kellerautomaten bedeutet das insbesondere, dass die Menge der vom Anfangsinhalt $\perp$ aus erreichbaren Kellerinhalte regulär ist.

Es gibt heute verschiedene Beweisansätze für den Satz von Büchi. Der Kerngedanke besteht darin, die Rechnungen, in denen sich Phasen der Wortexpansion mit solchen der Wortreduktion abwechseln, auf eine monotone Veränderung der Wortlänge (also des „Kellers“) zurückzuführen. Eine elegante Variante ist die „Saturierung von Automaten“ , worunter man das schrittweise Hinzufügen von Transitionen zu einem Automaten, der die Sprache A erkennt, versteht, bis ein Automat für die Sprache R erreicht ist.

Die besondere Bedeutung der Präfix-Ersetzungssysteme wird klar, wenn man sie den „Infix-Ersetzungssystemen“ gegenüberstellt, in denen eine Regel der Form $u \to v$ den Übergang von einem Wort xuy zu xvy erlaubt. Turingmaschinen [Tur37] induzieren Infix-Ersetzungssysteme (denn ein Turing-Konfigurationswort der Form xqy wird um das Symbol q herum modifiziert, um zur nächsten Konfiguration zu gelangen). Eine andere Form der Ersetzung, die Post in seinen „kanonischen Systemen“ [Pos43] verwendet hat, betrifft sowohl Präfix als auch Suffix eines Worts: Die Regel $u \to v$ erlaubt den Übergang von einem Wort uw zu wv. Sowohl Infix-Ersetzungsregeln als auch Posts kanonische Systeme erlauben die Simulation von Turingmaschinen. Das Erreichbarkeitsproblem ist hier also unentscheidbar.

3 Das Baumtheorem von Rabin und Kellergraphen

Nur fünf Jahre nach Büchi gelang es M. O. Rabin, viel komplexere Entscheidungsprobleme als nur das Erreichbarkeitsproblem zu lösen. Diese Perspektive wurde zunächst nur für eine einzige Struktur eröffnet (an Stelle aller Kellersysteme bzw. aller Präfixersetzungssysteme). Es handelt sich um den unendlichen binären Baum, dessen Knoten durch Wörter über dem Alphabet $\{1, 2\}$ darstellbar sind.



Als mathematische Struktur hat der binäre Baum die Form

$$T_2 = (\{1,2\}^*, \varepsilon, \cdot 1, \cdot 2)$$

mit dem ausgezeichneten Element ε (Wurzel des Baumes) und den Nachfolgerfunktionen $\cdot 1, \cdot 2$. Der Term $(((\varepsilon \cdot 1) \cdot 2) \cdot 2)$ bezeichnet den Knoten 122, der von der Wurzel mit den Nachfolgern $1, 2, 2$ erreicht wird.

Rabin bewies, dass man für jede Aussage einer bestimmten Logik-Sprache algorithmisch entscheiden kann, ob sie in T_2 gilt oder nicht. Diese Sprache der monadischen Logik zweiter Stufe (wir sagen im Folgenden einfach *monadische Logik*) hat Variablen $x, y, \ldots$ für Baumknoten, die Konstante ε, die beiden Nachfolgerfunktionen $\cdot 1, \cdot 2$, Variablen $X, Y, \ldots$ für Mengen von Baumknoten, Boole'sche Junktoren $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$ und schließlich die Quantoren $\exists, \forall$ (die man auf Knotenvariablen und auf Mengenvariablen anwenden kann).

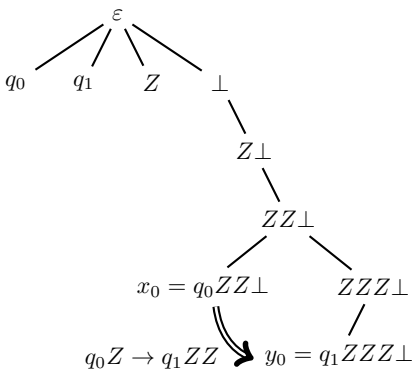
Das Rabin'sche Baumtheorem [Rab69] ist die folgende Aussage: *Man kann algorithmisch entscheiden, ob eine gegebene Aussage der monadischen Logik in der Struktur T_2 gilt oder nicht.* Fasst man die in T_2 geltenden Formeln zur „monadischen Theorie von T_2" zusammen, lässt sich der Satz kürzer formulieren: *Die monadische Theorie von T_2 ist entscheidbar.*

Wir werden im Folgenden auch von einer einfachen Verallgemeinerung des Satzes Gebrauch machen, in der an Stelle von T_2 der k-fach verzweigte Baum T_k betrachtet wird.

Der Beweis des Rabin'schen Baumtheorems ist so schwierig, dass es jahrzehntelanger Anstrengungen bedurfte, bis eine in Vorlesungen vortragbare Fassung erreicht war (vgl. [Tho97]). Rabin folgte einer Methode, die Büchi zuvor für die Struktur $N = (\mathbb{N}, 0, +1)$ entwickelt hatte (also für den „unären Baum"): Man zeigt, dass die Aussagen der monadischen Logik durch endliche Automaten (auf der Struktur N bzw. T_2) erfasst werden können und reduziert damit das logische Entscheidungsproblem auf die Frage, ob ein solcher „Büchi-Automat" (über N) bzw. „Rabin-Automat" (über T_2) einen erfolgreichen (unendlichen!) Lauf hat.

Was hat das Rabin'sche Resultat mit Kellersystemen zu tun? Wir stellen dazu Konfigurationen eines Kellersystems als Baumknoten dar. Betrachten wir ein Beispiel für ein System

mit zwei Zuständen q_0, q_1 und zwei Kellersymbolen $\bot, Z$. Wir arbeiten dann zur Darstellung der Konfiguationen im vierfach verzweigten Baum und stellen dort jede Konfiguration durch einen Baumknoten dar: Hierzu notieren wir die Konfigurationen vom Kellerbodensymbol $\bot$ aus nach unten, bis das Zustandssymbol erreicht ist. Zwei mögliche Konfigurationen, nämlich $x_0 = q_0ZZ\bot$ und $y_0 = q_1ZZZ\bot$, sind in der folgenden Figur notiert.



Die durch den Doppelpfeil angedeutete Anwendung der Transition $q_0Z \rightarrow q_1ZZ$, die von x_0 nach y_0 führt, ist allgemein auf Knoten x, y anwendbar, wenn $x = q_0Zw$ und $y = q_1ZZw$ für ein Wort $w \in Z^*\bot$; dieser Übergang entspricht im Baum jeweils einem Pfad der Länge 3 (ein Schritt zurück, zwei Schritte voraus). In der Tat ist die Wirkung einer Transition τ eines Kellersystems – Übergang von einem Baumknoten x zum Baumknoten y – beschreibbar durch eine Formel $\varphi_\tau(x, y)$ der monadischen Logik. Der Schritt durch irgendeine Transition des Kellersystems $\mathcal{K}$ ist dann durch die Disjunktion dieser Formeln $\varphi_\tau(x, y)$ definerbar; wir schreiben $\varphi_\mathcal{K}(x, y)$. Sind schließlich κ_1, κ_2 konkrete Konfigurationen (darstellbar durch Terme der monadischen Sprache, wie oben bereits erläutert), dann lässt sich die Erreichbarkeitsrelation $\kappa_1 \vdash_\mathcal{K}^* \kappa_2$ durch eine Formel beschreiben, die folgendes sagt: *Jede Knotenmenge, die κ_1 enthält und unter den Schritten gemäß $\varphi_\mathcal{K}$ abgeschlossen ist, enthält auch κ_2.*

Im Detail sieht diese Formel für die Beispielkonfigurationen $\kappa_1 = q_1ZZZ\bot$, $\kappa_2 = q_0Z\bot$ wie folgt aus:

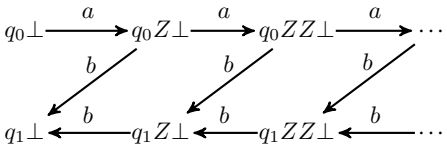
$$\forall X(\varepsilon \cdot \bot \cdot Z \cdot Z \cdot Z \cdot q_1 \in X \wedge \forall x \forall y(x \in X \wedge \varphi_\mathcal{K}(x, y) \rightarrow y \in X) \\ \rightarrow \varepsilon \cdot \bot \cdot Z \cdot q_0 \in X)$$

Das Zutreffen dieser Aussage kann man nach dem Satz von Rabin algorithmisch entscheiden, was einen neuen Beweis der ersten Aussage des Satzes von Büchi liefert. Doch ist dies nur eine sehr spezielle Anwendung des Satzes von Rabin (nämlich für Formeln einer sehr speziellen Form). Dass nun *jede* beliebige Formel, soweit sie nur in der Sprache der monadischen Logik formulierbar ist, entschieden werden kann, ist ein gewaltiger Schritt der Verallgemeinerung. (Auch der zweite Teil des Satzes von Büchi über die Regularität

der Erreichbarkeitsmenge ist über eine Erweiterung des Satzes von Rabin gewinnbar – wir gehen hierauf nicht näher ein.)

Wir haben gesehen, wie sich die Konfigurationen eines Kellersystems, die von einer gegebenen Konfiguration κ_1 aus erreichbar sind, durch eine Formel der monadischen Logik beschreiben lassen. Wir wenden diese Idee nun auf die Anfangskonfiguration $q_0\bot$ an. Dann erhalten wir eine Beschreibung $\psi_{\mathcal{K}}(x)$ aller erreichbaren Konfigurationen x von $\mathcal{K}$. Aus diesen Konfigurationen können wir $\mathcal{K}$ auch als Graphen präsentieren: Ein *Kellergraph* ist die Darstellung eines Kellersystems als Graph, indem man als Knoten alle (von $q_0\bot$ aus) erreichbaren Konfigurationen nimmt und als Kanten alle durch Transitionen realisierbaren Konfigurationsübergänge. Auch diese Kantenrelation ist in der Sprache der monadischen Logik definierbar, wie wir oben mit der Formel $\varphi_{\mathcal{K}}(x, y)$ gesehen haben. Man kann die Kantenrelation auch durch die Transitionen τ parametrisieren und hat dann die Beschreibung eines kantenbeschrifteten Graphen.

Als einfaches Beispiel eines Kellergraphen betrachten wir den Standard-Kellerautomaten, der die nichtreguläre Sprache $\{a^i b^i \mid i > 0\}$ erkennt. Er realisiert über dem Kellerbodensymbol $\bot$ einen Zähler durch Wörter in Z^*; als Anfangskonfiguration wählen wir $q_0\bot$, als Endkonfiguration $q_1\bot$.



Wir können eine Kopie des Kellergraphen in der Baumstruktur aller Konfigurationen herstellen, indem wir mit der Formel $\psi_{\mathcal{K}}(x)$ den Grundbereich beschreiben und mit den Formeln $\varphi_\tau(x, y)$ die Kantenrelationen. Diesen Vorgang der Beschreibung eines Graphen G im Konfigurationsbaum T nennt man auch eine „monadische Interpretation von G in T“ . Es ist einfach zu sehen, dass die monadische Theorie von G entscheidbar ist, sofern dies für T zutrifft.

Diese Fakten können wir nun zusammensetzen: Jeder Kellergraph G ist in einem Baum T_k (geeigneten Verzweigungsgrads, in unserem Beispiel mit $k = 4$) monadisch interpretierbar. Und da T_k eine entscheidbare Theorie hat, gilt dies auch für G. Dies ist die Aussage eines Satzes, den Muller und Schupp gezeigt haben [MS85]: *Die monadische Theorie eines Kellergraphen ist entscheidbar.*

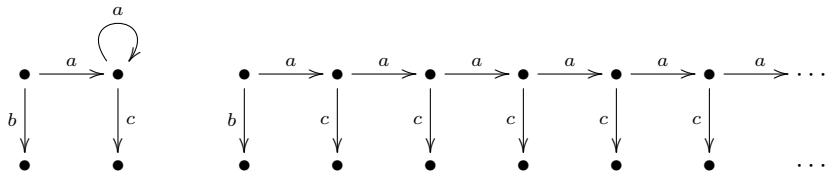
Es ist bemerkenswert, dass wir uns nun von sehr einfachen „Standardstrukturen“ (den Bäumen T_k) gelöst haben und eine riesige Klasse von Graphen, nämlich alle Kellergraphen, erschlossen haben. Die Kellergraphen sind damit das erste Beispiel für eine Klasse *unendlicher Strukturen*, über denen automatische Verifikation („Model-Checking“) möglich ist. Beim „Infinite-State Model-Checking“ fragt man:

- Gegeben ein unendlicher Graph G als Systemmodell (Zustände und Übergänge) und eine Anforderung φ an G, erfüllt G die Bedingung φ?

Betrachten wir Anforderungen φ der monadischen Logik, erhalten wir eine Lösung des Model-Checking-Problems für alle Kellergraphen.

4 Abwicklungen und die Kellerhierarchie

Im vorangehenden Abschnitt haben wir erläutert, wie man in einem Baum einen Graphen beschreiben kann. Eine zweite sehr natürliche Operation macht umgekehrt aus einem Graphen (mit ausgezeichnetem „Anfangsknoten“ v_0) einen Baum, nämlich die „Abwicklung von v_0 aus“ . Ein Beispiel illustriert diese Operation (hier ist v_0 der Knoten links oben im ersten Graphen):



Ein weiteres Beispiel: Aus dem Ein-Punkt-Graphen mit den beiden durch 1 bzw. 2 beschrifteten Kanten

wird durch Abwicklung der unendliche binäre Baum. Der Ein-Punkt-Graph hat trivialerweise eine entscheidbare monadische Theorie. Dass der sich durch Abwicklung ergebende binäre Baum wiederum eine entscheidbare monadische Theorie hat, ist ein schwieriges Resultat (Satz von Rabin). Auf A. Muchnik geht nun ein viel stärkeres Ergebnis zurück: An Stelle des Ein-Punkt-Graphen kann *jeder* (auch unendliche) Graph treten, wenn er nur selbst eine entscheidbare monadische Theorie hat (vgl. [Sem84, BB01, Wal02]).

Wir benutzen folgende Notation: Ist v_0 ein Knoten von G, so sei $T(G, v_0)$ der Abwicklungsbaum von G von v_0 aus. Dann besagt der Satz von Muchnik: *Ist die monadische Theorie des Graphen G entscheidbar und der G-Knoten v_0 in der monadischen Sprache definierbar, dann ist die monadische Theorie von $T(G, v_0)$ entscheidbar.*

D. Caucal regte in [Cau03] an, ausgehend von den endlichen Bäumen (die jeweils eine entscheidbare monadische Theorie haben), die beiden Prozesse „monadische Interpretation eines Graphen in einem Baum“ und „Abwicklung eines Graphen zu einem Baum“ abwechselnd anzuwenden. Damit erhält man die sog. *Kellerhierarchie* oder *Caucal-Hierarchie*. Die ersten Stufen sind gut bekannt:

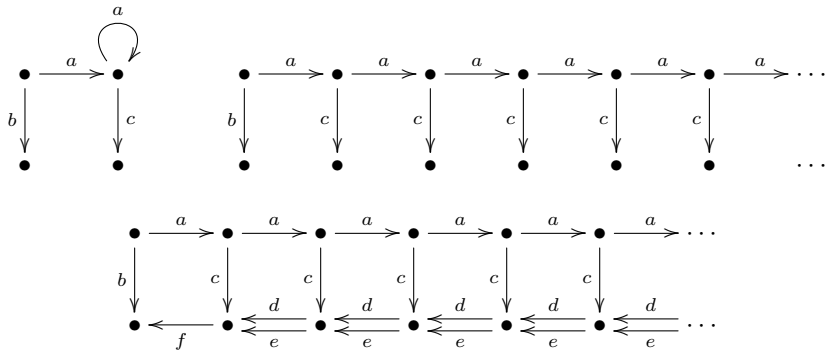
- $\mathcal{T}_0$ ist die Klasse der endlichen Bäume,
- $\mathcal{G}_0$ ist die Klasse der endlichen Graphen,

- $\mathcal{T}_1$ enthält die regulären Bäume (die Abwicklungen endlicher Graphen),
- $\mathcal{G}_1$ enthält u.a. die Kellergraphen.

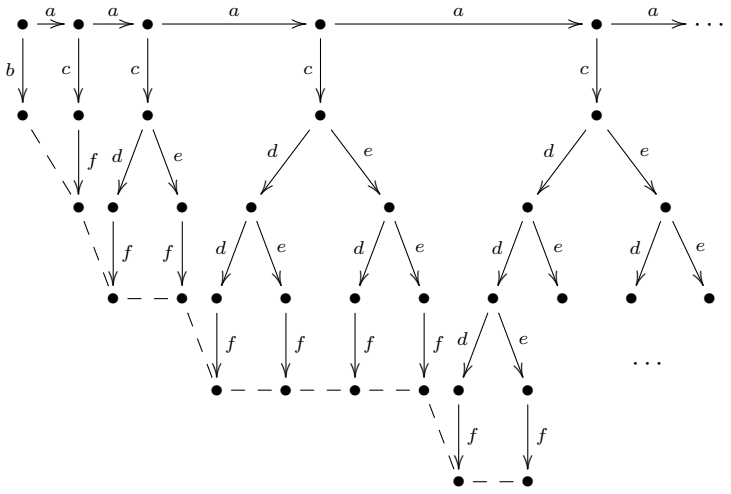
Da beide angewandten Prozesse die Entscheidbarkeit der monadischen Theorie erhalten, wissen wir: *Die monadische Theorie jeder Struktur der Kellerhierarchie ist entscheidbar.*

Die Landschaft der Strukturen, die man in der Kellerhierarchie erhält, ist so reichhaltig, dass man bisher nicht gut versteht, was man damit erschlossen hat. Ein kleines Beispiel mag die Mächtigkeit des Zusammenspiels von „monadischen Beschreibungen“ (Interpretationen) und „Abwicklungen“ erläutern.

Wir beginnen mit einem endlichen Graphen und seiner Abwicklung wie bereits oben dargestellt und erreichen dann mit einer Interpretation einen neuen Graphen, in dem noch Kanten mit Beschriftung d, e, f hinzugefügt sind.



Eine weitere Abwicklung liefert eine schon viel komplexere Baumstruktur:



In dieser Baumstruktur ist die Menge der Blätter, mit dem „ersten Blatt“ (als Ziel der b-Kante) und mit der Funktion „Übergang zum nächsten Blatt“ (angedeutet durch die gestrichelte Linie), monadisch definierbar: Dies ist eine Kopie der Nachfolgerstruktur $N = (\mathbb{N}, 0, +1)$ der natürlichen Zahlen. Doch ist noch mehr definierbar, nämlich die Menge der „linkesten Blätter“ der Unterbäume unterhalb der b- und c-Kanten. Diese Teilmenge in der Kopie von N ist die Menge Pot_2 der Zweierpotenzen $(0, 1, 2, 4, 8, \ldots)$. Mit zwei Abwicklungen und zwei Interpretationen ist also aus einem endlichen Graphen die Struktur $(\mathbb{N}, 0, +1, \mathrm{Pot}_2)$ entstanden, die nicht mehr „regulär“ oder „periodisch“ erscheint, aber immer noch eine entscheidbare monadische Theorie hat.

Die Kellerhierarchie hat eine Geschichte, die mit anderen Zugängen bis in die 1970er-Jahre zurückgeht. Maslov hat 1974 eine Hierarchie von Kellerautomaten eingeführt, deren Keller geschachtelt werden können. Hier betrachtet man auf Stufe 1 übliche Kellerautomaten, auf Stufe 2 Kellerautomaten, deren Kellereinträge wiederum Keller sind, auf Stufe3 Kellerautomaten, deren Kellereinträge Keller 2. Stufe sind, usw. Vor gut 10 Jahren bewiesen Carayol und Wöhrle [CW03]: *Die Transitionsgraphen von Kellerautomaten n-ter Stufe (mit ϵ-Transitionen) stimmen mit den Graphen der n-ten Stufe der Caucal-Hierarchie überein.* Damit ist die Bezeichnung „Kellerhierarchie“ gerechtfertigt. – Ein weiterer Zugang ist das Studium von rekursiven Programmschemata, bei denen über Rekursion höherer Stufe die Bäume der Caucal-Hierarchie spezifiziert werden (und sogar noch weitere Baumstrukturen). Dies genauer auszuführen ist hier aus Platzgründen nicht möglich – auch liegt der Schwerpunkt dort auf Bäumen, während im Model-Checking die unendlichen Transitionsgraphen im Vordergrund stehen. Ein erschöpfender und klarer Übersichtsartikel über den Zusammenhang mit der Theorie der Rekursionsschemata ist [Ong13].

5 Ausblick

Die Caucal-Hierarchie (oder: Kellerhierarchie) ist gegenwärtig der Gegenstand intensiver Forschungen. Die Hierarchie liefert eine reichhaltige Klasse von Strukturen, die sich durch modelltheoretische Operationen (Interpretation und Abwicklung), aber auch als Konfigurationsgraphen von geschachtelten Kellersystemen ergeben. All diese Strukturen erlauben die algorithmische Entscheidung, welche Aussagen der monadischen Logik darin gelten und welche nicht. Wir überblicken aber nur die ersten beiden Stufen dieser Hierarchie in befriedigender Weise; was darüber liegt, ist noch nicht gut erforscht. Auch jenseits der Kellerhierarchie sind Strukturen bekannt, die noch eine entscheidbare monadische Theorie haben. Das führt auf die Frage, ob man die beiden Operationen der monadischen Interpretation und der Abwicklung durch weitere Prozesse ergänzen kann, um weitere Strukturen mit entscheidbarer monadischer Theorie zu erhalten. Eine weitere offene Forschungsfrage zielt auf eine Abschwächung der monadischen Logik (etwa auf das aus praktischer Sicht interessante Fragment der „monadic transitive closure logic“), um die Entscheidbarkeit der zugehörigen Theorie auch für Strukturen außerhalb der Kellerhierarchie zu garantieren.

Literatur

[BB01] Dietmar Berwanger und Achim Blumensath. The monadic theory of tree-like structures. In *Automata, Logics, and Infinite Games*, Lecture Notes in Computer Science, Vol. 2500, Seiten 285–302. Springer, 2001.

[Büc64] J. Richard Büchi. Regular Canonical Systems. *Archiv für Mathematische Grundlagenforschung*, 6:91–111, 1964.

[Cau03] Didier Caucal. On infinite transition graphs having a decidable monadic theory. *Theoretical Computer Science*, 290(1):79–115, 2003.

[CW03] Arnaud Carayol und Stefan Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *Proceedings of the 23rd Conference on Foundations of Software Technology and Theoretical Computer Science, FST TCS 2003*, Lecture Notes in Computer Science, Vol. 2914, Seiten 112–123. Springer, 2003.

[MS85] David E. Muller und Paul E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37:51–75, 1985.

[Ong13] Luke Ong. Recursion schemes, collapsible pushdown automata and higher-order model checking. In *Language and Automata Theory and Applications - 7th International Conference, LATA 2013*, Lecture Notes in Computer Science, Vol. 7810, Seiten 13–41. Springer, 2013.

[Pos43] Emil Post. Formal Reductions of the General Combinatorial Decision Problem. *American Journal of Mathematics*, 65(2):197–215, 1943.

[Rab69] Michael O. Rabin. Decidability of Second-Order Theories and Automata on Infinite Trees. *Transactions of the American Mathematical Society*, 141:1–35, July 1969.

[Sem84] Alexei L. Semenov. Decidability of monadic theories. In *Proceedings of the 11th International Symposium on Mathematical Foundations of Computer Science, MFCS '84*, Lecture Notes in Computer Science, Vol. 176, Seiten 162–175. Springer, 1984.

[Tho97] Wolfgang Thomas. Languages, Automata, and Logic. In G. Rozenberg und A. Salomaa, Hrsg., *Handbook of Formal Language Theory, Vol. 3*, Seiten 389–455. Springer, 1997.

[Tur37] Alan Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, Series 2*, 42(1):230–265, 1937.

[Wal02] Igor Walukiewicz. Monadic second-order logic on tree-like structures. *Theoretical Computer Science*, 275(1-2):311–346, 2002.

GI-Edition Lecture Notes in Informatics

P-1 Gregor Engels, Andreas Oberweis, Albert Zündorf (Hrsg.): Modellierung 2001.

P-2 Mikhail Godlevsky, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications, ISTA'2001.

P-3 Ana M. Moreno, Reind P. van de Riet (Hrsg.): Applications of Natural Lan-guage to Information Systems, NLDB'2001.

P-4 H. Wörn, J. Mühling, C. Vahl, H.-P. Meinzer (Hrsg.): Rechner- und sensorgestützte Chirurgie; Workshop des SFB 414.

P-5 Andy Schürr (Hg.): OMER – Object-Oriented Modeling of Embedded Real-Time Systems.

P-6 Hans-Jürgen Appelrath, Rolf Beyer, Uwe Marquardt, Heinrich C. Mayr, Claudia Steinberger (Hrsg.): Unternehmen Hochschule, UH'2001.

P-7 Andy Evans, Robert France, Ana Moreira, Bernhard Rumpe (Hrsg.): Practical UML-Based Rigorous Development Methods – Countering or Integrating the extremists, pUML'2001.

P-8 Reinhard Keil-Slawik, Johannes Magenheim (Hrsg.): Informatikunterricht und Medienbildung, INFOS'2001.

P-9 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Innovative Anwendungen in Kommunikationsnetzen, 15. DFN Arbeitstagung.

P-10 Mirjam Minor, Steffen Staab (Hrsg.): 1st German Workshop on Experience Management: Sharing Experiences about the Sharing Experience.

P-11 Michael Weber, Frank Kargl (Hrsg.): Mobile Ad-Hoc Netzwerke, WMAN 2002.

P-12 Martin Glinz, Günther Müller-Luschnat (Hrsg.): Modellierung 2002.

P-13 Jan von Knop, Peter Schirmbacher and Viljan Mahni_ (Hrsg.): The Changing Universities – The Role of Technology.

P-14 Robert Tolksdorf, Rainer Eckstein (Hrsg.): XML-Technologien für das Semantic Web – XSW 2002.

P-15 Hans-Bernd Bludau, Andreas Koop (Hrsg.): Mobile Computing in Medicine.

P-16 J. Felix Hampe, Gerhard Schwabe (Hrsg.): Mobile and Collaborative Business 2002.

P-17 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Zukunft der Netze –Die Verletzbarkeit meistern, 16. DFN Arbeitstagung.

P-18 Elmar J. Sinz, Markus Plaha (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2002.

P-19 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3. Okt. 2002 in Dortmund.

P-20 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3. Okt. 2002 in Dortmund (Ergänzungsband).

P-21 Jörg Desel, Mathias Weske (Hrsg.): Promise 2002: Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen.

P-22 Sigrid Schubert, Johannes Magenheim, Peter Hubwieser, Torsten Brinda (Hrsg.): Forschungsbeiträge zur "Didaktik der Informatik" – Theorie, Praxis, Evaluation.

P-23 Thorsten Spitta, Jens Borchers, Harry M. Sneed (Hrsg.): Software Management 2002 – Fortschritt durch Beständigkeit

P-24 Rainer Eckstein, Robert Tolksdorf (Hrsg.): XMIDX 2003 – XML-Technologien für Middleware – Middleware für XML-Anwendungen

P-25 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Commerce – Anwendungen und Perspektiven – 3. Workshop Mobile Commerce, Universität Augsburg, 04.02.2003

P-26 Gerhard Weikum, Harald Schöning, Erhard Rahm (Hrsg.): BTW 2003: Datenbanksysteme für Business, Technologie und Web

P-27 Michael Kroll, Hans-Gerd Lipinski, Kay Melzer (Hrsg.): Mobiles Computing in der Medizin

P-28 Ulrich Reimer, Andreas Abecker, Steffen Staab, Gerd Stumme (Hrsg.): WM 2003: Professionelles Wissensmanagement – Er-fahrungen und Visionen

P-29 Antje Düsterhöft, Bernhard Thalheim (Eds.): NLDB'2003: Natural Language Processing and Information Systems

P-30 Mikhail Godlevsky, Stephen Liddle, Heinrich C. Mayr (Eds.): Information Systems Technology and its Applications

P-31 Arslan Brömme, Christoph Busch (Eds.): BIOSIG 2003: Biometrics and Electronic Signatures

P-32 Peter Hubwieser (Hrsg.): Informatische Fachkonzepte im Unterricht – INFOS 2003

P-33 Andreas Geyer-Schulz, Alfred Taudes (Hrsg.): Informationswirtschaft: Ein Sektor mit Zukunft

P-34 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 1)

P-35 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 2)

P-36 Rüdiger Grimm, Hubert B. Keller, Kai Rannenberg (Hrsg.): Informatik 2003 – Mit Sicherheit Informatik

P-37 Arndt Bode, Jörg Desel, Sabine Rathmayer, Martin Wessner (Hrsg.): DeLFI 2003: e-Learning Fachtagung Informatik

P-38 E.J. Sinz, M. Plaha, P. Neckel (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2003

P-39 Jens Nedon, Sandra Frings, Oliver Göbel (Hrsg.): IT-Incident Management & IT-Forensics – IMF 2003

P-40 Michael Rebstock (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2004

P-41 Uwe Brinkschulte, Jürgen Becker, Dietmar Fey, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle, Thomas Runkler (Edts.): ARCS 2004 – Organic and Pervasive Computing

P-42 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Economy – Transaktionen und Prozesse, Anwendungen und Dienste

P-43 Birgitta König-Ries, Michael Klein, Philipp Obreiter (Hrsg.): Persistance, Scalability, Transactions – Database Mechanisms for Mobile Applications

P-44 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): Security, E-Learning. E-Services

P-45 Bernhard Rumpe, Wofgang Hesse (Hrsg.): Modellierung 2004

P-46 Ulrich Flegel, Michael Meier (Hrsg.): Detection of Intrusions of Malware & Vulnerability Assessment

P-47 Alexander Prosser, Robert Krimmer (Hrsg.): Electronic Voting in Europe – Technology, Law, Politics and Society

P-48 Anatoly Doroshenko, Terry Halpin, Stephen W. Liddle, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications

P-49 G. Schiefer, P. Wagner, M. Morgenstern, U. Rickert (Hrsg.): Integration und Datensicherheit – Anforderungen, Konflikte und Perspektiven

P-50 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 1) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm

P-51 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 2) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm

P-52 Gregor Engels, Silke Seehusen (Hrsg.): DELFI 2004 – Tagungsband der 2. e-Learning Fachtagung Informatik

P-53 Robert Giegerich, Jens Stoye (Hrsg.): German Conference on Bioinformatics – GCB 2004

P-54 Jens Borchers, Ralf Kneuper (Hrsg.): Softwaremanagement 2004 – Outsourcing und Integration

P-55 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): E-Science und Grid Adhoc-Netze Medienintegration

P-56 Fernand Feltz, Andreas Oberweis, Benoit Otjacques (Hrsg.): EMISA 2004 – Informationssysteme im E-Business und E-Government

P-57 Klaus Turowski (Hrsg.): Architekturen, Komponenten, Anwendungen

P-58 Sami Beydeda, Volker Gruhn, Johannes Mayer, Ralf Reussner, Franz Schweiggert (Hrsg.): Testing of Component-Based Systems and Software Quality

P-59 J. Felix Hampe, Franz Lehner, Key Pousttchi, Kai Ranneberg, Klaus Turowski (Hrsg.): Mobile Business – Processes, Platforms, Payments

P-60 Steffen Friedrich (Hrsg.): Unterrichtskonzepte für inforrmatische Bildung

P-61 Paul Müller, Reinhard Gotzhein, Jens B. Schmitt (Hrsg.): Kommunikation in verteilten Systemen

P-62 Federrath, Hannes (Hrsg.): „Sicherheit 2005“ – Sicherheit – Schutz und Zuverlässigkeit

P-63 Roland Kaschek, Heinrich C. Mayr, Stephen Liddle (Hrsg.): Information Systems – Technology and ist Applications

P-64 Peter Liggesmeyer, Klaus Pohl, Michael Goedicke (Hrsg.): Software Engineering 2005

P-65 Gottfried Vossen, Frank Leymann, Peter Lockemann, Wolffried Stucky (Hrsg.): Datenbanksysteme in Business, Technologie und Web

P-66 Jörg M. Haake, Ulrike Lucke, Djamshid Tavangarian (Hrsg.): DeLFI 2005: 3. deutsche e-Learning Fachtagung Informatik

P-67 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 1)

P-68 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 2)

P-69 Robert Hirschfeld, Ryszard Kowalcyk, Andreas Polze, Matthias Weske (Hrsg.): NODe 2005, GSEM 2005

P-70 Klaus Turowski, Johannes-Maria Zaha (Hrsg.): Component-oriented Enterprise Application (COAE 2005)

P-71 Andrew Torda, Stefan Kurz, Matthias Rarey (Hrsg.): German Conference on Bioinformatics 2005

P-72 Klaus P. Jantke, Klaus-Peter Fähnrich, Wolfgang S. Wittig (Hrsg.): Marktplatz Internet: Von e-Learning bis e-Payment

P-73 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): “Heute schon das Morgen sehen“

P-74 Christopher Wolf, Stefan Lucks, Po-Wah Yau (Hrsg.): WEWoRC 2005 – Western European Workshop on Research in Cryptology

P-75 Jörg Desel, Ulrich Frank (Hrsg.): Enterprise Modelling and Information Systems Architecture

P-76 Thomas Kirste, Birgitta König-Riess, Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Informationssysteme – Potentiale, Hindernisse, Einsatz

P-77 Jana Dittmann (Hrsg.): SICHERHEIT 2006

P-78 K.-O. Wenkel, P. Wagner, M. Morgenstern, K. Luzi, P. Eisermann (Hrsg.): Land- und Ernährungswirtschaft im Wandel

P-79 Bettina Biel, Matthias Book, Volker Gruhn (Hrsg.): Softwareengineering 2006

P-80 Mareike Schoop, Christian Huemer, Michael Rebstock, Martin Bichler (Hrsg.): Service-Oriented Electronic Commerce

P-81 Wolfgang Karl, Jürgen Becker, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle (Hrsg.): ARCS´06

P-82 Heinrich C. Mayr, Ruth Breu (Hrsg.): Modellierung 2006

P-83 Daniel Huson, Oliver Kohlbacher, Andrei Lupas, Kay Nieselt and Andreas Zell (eds.): German Conference on Bioinformatics

P-84 Dimitris Karagiannis, Heinrich C. Mayr, (Hrsg.): Information Systems Technology and its Applications

P-85 Witold Abramowicz, Heinrich C. Mayr, (Hrsg.): Business Information Systems

P-86 Robert Krimmer (Ed.): Electronic Voting 2006

P-87 Max Mühlhäuser, Guido Rößling, Ralf Steinmetz (Hrsg.): DELFI 2006: 4. e-Learning Fachtagung Informatik

P-88 Robert Hirschfeld, Andreas Polze, Ryszard Kowalczyk (Hrsg.): NODe 2006, GSEM 2006

P-90 Joachim Schelp, Robert Winter, Ulrich Frank, Bodo Rieger, Klaus Turowski (Hrsg.): Integration, Informationslogistik und Architektur

P-91 Henrik Stormer, Andreas Meier, Michael Schumacher (Eds.): European Conference on eHealth 2006

P-92 Fernand Feltz, Benoît Otjacques, Andreas Oberweis, Nicolas Poussing (Eds.): AIM 2006

P-93 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 1

P-94 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 2

P-95 Matthias Weske, Markus Nüttgens (Eds.): EMISA 2005: Methoden, Konzepte und Technologien für die Entwicklung von dienstbasierten Informationssystemen

P-96 Saartje Brockmans, Jürgen Jung, York Sure (Eds.): Meta-Modelling and Ontologies

P-97 Oliver Göbel, Dirk Schadt, Sandra Frings, Hardo Hase, Detlef Günther, Jens Nedon (Eds.): IT-Incident Mangament & IT-Forensics – IMF 2006

P-98 Hans Brandt-Pook, Werner Simonsmeier und Thorsten Spitta (Hrsg.): Beratung in der Softwareentwicklung – Modelle, Methoden, Best Practices

P-99 Andreas Schwill, Carsten Schulte, Marco Thomas (Hrsg.): Didaktik der Informatik

P-100 Peter Forbrig, Günter Siegel, Markus Schneider (Hrsg.): HDI 2006: Hochschuldidaktik der Informatik

P-101 Stefan Böttinger, Ludwig Theuvsen, Susanne Rank, Marlies Morgenstern (Hrsg.): Agrarinformatik im Spannungsfeld zwischen Regionalisierung und globalen Wertschöpfungsketten

P-102 Otto Spaniol (Eds.): Mobile Services and Personalized Environments

P-103 Alfons Kemper, Harald Schöning, Thomas Rose, Matthias Jarke, Thomas Seidl, Christoph Quix, Christoph Brochhaus (Hrsg.): Datenbanksysteme in Business, Technologie und Web (BTW 2007)

P-104 Birgitta König-Ries, Franz Lehner, Rainer Malaka, Can Türker (Hrsg.) MMS 2007: Mobilität und mobile Informationssysteme

P-105 Wolf-Gideon Bleek, Jörg Raasch, Heinz Züllighoven (Hrsg.) Software Engineering 2007

P-106 Wolf-Gideon Bleek, Henning Schwentner, Heinz Züllighoven (Hrsg.) Software Engineering 2007 – Beiträge zu den Workshops

P-107 Heinrich C. Mayr, Dimitris Karagiannis (eds.) Information Systems Technology and its Applications

P-108 Arslan Brömme, Christoph Busch, Detlef Hühnlein (eds.) BIOSIG 2007: Biometrics and Electronic Signatures

P-109 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 1

P-110 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 2

P-111 Christian Eibl, Johannes Magenheim, Sigrid Schubert, Martin Wessner (Hrsg.) DeLFI 2007: 5. e-Learning Fachtagung Informatik

P-112 Sigrid Schubert (Hrsg.) Didaktik der Informatik in Theorie und Praxis

P-113 Sören Auer, Christian Bizer, Claudia Müller, Anna V. Zhdanova (Eds.) The Social Semantic Web 2007 Proceedings of the 1st Conference on Social Semantic Web (CSSW)

P-114 Sandra Frings, Oliver Göbel, Detlef Günther, Hardo G. Hase, Jens Nedon, Dirk Schadt, Arslan Brömme (Eds.) IMF2007 IT-incident management & IT-forensics Proceedings of the 3rd International Conference on IT-Incident Management & IT-Forensics

P-115 Claudia Falter, Alexander Schliep, Joachim Selbig, Martin Vingron and Dirk Walther (Eds.) German conference on bioinformatics GCB 2007

P-116 Witold Abramowicz, Leszek Maciszek (Eds.) Business Process and Services Computing 1st International Working Conference on Business Process and Services Computing BPSC 2007

P-117 Ryszard Kowalczyk (Ed.) Grid service engineering and manegement The 4th International Conference on Grid Service Engineering and Management GSEM 2007

P-118 Andreas Hein, Wilfried Thoben, Hans-Jürgen Appelrath, Peter Jensch (Eds.) European Conference on ehealth 2007

P-119 Manfred Reichert, Stefan Strecker, Klaus Turowski (Eds.) Enterprise Modelling and Information Systems Architectures Concepts and Applications

P-120 Adam Pawlak, Kurt Sandkuhl, Wojciech Cholewa, Leandro Soares Indrusiak (Eds.) Coordination of Collaborative Engineering - State of the Art and Future Challenges

P-121 Korbinian Herrmann, Bernd Bruegge (Hrsg.) Software Engineering 2008 Fachtagung des GI-Fachbereichs Softwaretechnik

P-122 Walid Maalej, Bernd Bruegge (Hrsg.)
Software Engineering 2008 - Workshopband
Fachtagung des GI-Fachbereichs Softwaretechnik

P-123 Michael H. Breitner, Martin Breunig, Elgar Fleisch, Ley Pousttchi, Klaus Turowski (Hrsg.)
Mobile und Ubiquitäre Informationssysteme – Technologien, Prozesse, Marktfähigkeit
Proceedings zur 3. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2008)

P-124 Wolfgang E. Nagel, Rolf Hoffmann, Andreas Koch (Eds.)
9th Workshop on Parallel Systems and Algorithms (PASA)
Workshop of the GI/ITG Speciel Interest Groups PARS and PARVA

P-125 Rolf A.E. Müller, Hans-H. Sundermeier, Ludwig Theuvsen, Stephanie Schütze, Marlies Morgenstern (Hrsg.)
Unternehmens-IT: Führungsinstrument oder Verwaltungsbürde
Referate der 28. GIL Jahrestagung

P-126 Rainer Gimnich, Uwe Kaiser, Jochen Quante, Andreas Winter (Hrsg.)
10th Workshop Software Reengineering (WSR 2008)

P-127 Thomas Kühne, Wolfgang Reisig, Friedrich Steimann (Hrsg.)
Modellierung 2008

P-128 Ammar Alkassar, Jörg Siekmann (Hrsg.)
Sicherheit 2008
Sicherheit, Schutz und Zuverlässigkeit
Beiträge der 4. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)
2.-4. April 2008
Saarbrücken, Germany

P-129 Wolfgang Hesse, Andreas Oberweis (Eds.)
Sigsand-Europe 2008
Proceedings of the Third AIS SIGSAND European Symposium on Analysis, Design, Use and Societal Impact of Information Systems

P-130 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
1. DFN-Forum Kommunikations-technologien Beiträge der Fachtagung

P-131 Robert Krimmer, Rüdiger Grimm (Eds.)
3rd International Conference on Electronic Voting 2008
Co-organized by Council of Europe, Gesellschaft für Informatik and E-Voting. CC

P-132 Silke Seehusen, Ulrike Lucke, Stefan Fischer (Hrsg.)
DeLFI 2008:
Die 6. e-Learning Fachtagung Informatik

P-133 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik
Band 1

P-134 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik
Band 2

P-135 Torsten Brinda, Michael Fothe, Peter Hubwieser, Kirsten Schlüter (Hrsg.)
Didaktik der Informatik – Aktuelle Forschungsergebnisse

P-136 Andreas Beyer, Michael Schroeder (Eds.)
German Conference on Bioinformatics GCB 2008

P-137 Arslan Brömme, Christoph Busch, Detlef Hühnlein (Eds.)
BIOSIG 2008: Biometrics and Electronic Signatures

P-138 Barbara Dinter, Robert Winter, Peter Chamoni, Norbert Gronau, Klaus Turowski (Hrsg.)
Synergien durch Integration und Informationslogistik
Proceedings zur DW2008

P-139 Georg Herzwurm, Martin Mikusz (Hrsg.)
Industrialisierung des Software-Managements
Fachtagung des GI-Fachausschusses Management der Anwendungsentwick-lung und -wartung im Fachbereich Wirtschaftsinformatik

P-140 Oliver Göbel, Sandra Frings, Detlef Günther, Jens Nedon, Dirk Schadt (Eds.)
IMF 2008 - IT Incident Management & IT Forensics

P-141 Peter Loos, Markus Nüttgens, Klaus Turowski, Dirk Werth (Hrsg.)
Modellierung betrieblicher Informations-systeme (MobIS 2008)
Modellierung zwischen SOA und Compliance Management

P-142 R. Bill, P. Korduan, L. Theuvsen, M. Morgenstern (Hrsg.)
Anforderungen an die Agrarinformatik durch Globalisierung und Klimaveränderung

P-143 Peter Liggesmeyer, Gregor Engels, Jürgen Münch, Jörg Dörr, Norman Riegel (Hrsg.)
Software Engineering 2009
Fachtagung des GI-Fachbereichs Softwaretechnik

P-144 Johann-Christoph Freytag, Thomas Ruf, Wolfgang Lehner, Gottfried Vossen (Hrsg.)
Datenbanksysteme in Business, Technologie und Web (BTW)

P-145 Knut Hinkelmann, Holger Wache (Eds.)
WM2009: 5th Conference on Professional Knowledge Management

P-146 Markus Bick, Martin Breunig, Hagen Höpfner (Hrsg.)
Mobile und Ubiquitäre Informationssysteme – Entwicklung, Implementierung und Anwendung
4. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2009)

P-147 Witold Abramowicz, Leszek Maciaszek, Ryszard Kowalczyk, Andreas Speck (Eds.)
Business Process, Services Computing and Intelligent Service Management
BPSC 2009 · ISM 2009 · YRW-MBP 2009

P-148 Christian Erfurth, Gerald Eichler, Volkmar Schau (Eds.)
9th International Conference on Innovative Internet Community Systems
I²CS 2009

P-149 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
2. DFN-Forum Kommunikationstechnologien
Beiträge der Fachtagung

P-150 Jürgen Münch, Peter Liggesmeyer (Hrsg.)
Software Engineering
2009 - Workshopband

P-151 Armin Heinzl, Peter Dadam, Stefan Kirn, Peter Lockemann (Eds.)
PRIMIUM
Process Innovation for Enterprise Software

P-152 Jan Mendling, Stefanie Rinderle-Ma, Werner Esswein (Eds.)
Enterprise Modelling and Information Systems Architectures
Proceedings of the 3rd Int'l Workshop EMISA 2009

P-153 Andreas Schwill, Nicolas Apostolopoulos (Hrsg.)
Lernen im Digitalen Zeitalter
DeLFI 2009 – Die 7. E-Learning Fachtagung Informatik

P-154 Stefan Fischer, Erik Maehle Rüdiger Reischuk (Hrsg.)
INFORMATIK 2009
Im Focus das Leben

P-155 Arslan Brömme, Christoph Busch, Detlef Hühnlein (Eds.)
BIOSIG 2009:
Biometrics and Electronic Signatures
Proceedings of the Special Interest Group on Biometrics and Electronic Signatures

P-156 Bernhard Koerber (Hrsg.)
Zukunft braucht Herkunft
25 Jahre »INFOS – Informatik und Schule«

P-157 Ivo Grosse, Steffen Neumann, Stefan Posch, Falk Schreiber, Peter Stadler (Eds.)
German Conference on Bioinformatics 2009

P-158 W. Claupein, L. Theuvsen, A. Kämpf, M. Morgenstern (Hrsg.)
Precision Agriculture Reloaded – Informationsgestützte Landwirtschaft

P-159 Gregor Engels, Markus Luckey, Wilhelm Schäfer (Hrsg.)
Software Engineering 2010

P-160 Gregor Engels, Markus Luckey, Alexander Pretschner, Ralf Reussner (Hrsg.)
Software Engineering 2010 – Workshopband
(inkl. Doktorandensymposium)

P-161 Gregor Engels, Dimitris Karagiannis Heinrich C. Mayr (Hrsg.)
Modellierung 2010

P-162 Maria A. Wimmer, Uwe Brinkhoff, Siegfried Kaiser, Dagmar Lück-Schneider, Erich Schweighofer, Andreas Wiebe (Hrsg.)
Vernetzte IT für einen effektiven Staat
Gemeinsame Fachtagung Verwaltungsinformatik (FTVI) und Fachtagung Rechtsinformatik (FTRI) 2010

P-163 Markus Bick, Stefan Eulgem, Elgar Fleisch, J. Felix Hampe, Birgitta König-Ries, Franz Lehner, Key Pousttchi, Kai Rannenberg (Hrsg.)
Mobile und Ubiquitäre Informationssysteme
Technologien, Anwendungen und Dienste zur Unterstützung von mobiler Kollaboration

P-164 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2010: Biometrics and Electronic Signatures Proceedings of the Special Interest Group on Biometrics and Electronic Signatures

P-165 Gerald Eichler, Peter Kropf, Ulrike Lechner, Phayung Meesad, Herwig Unger (Eds.)
10th International Conference on Innovative Internet Community Systems (I^2CS) – Jubilee Edition 2010 –

P-166 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
3. DFN-Forum Kommunikationstechnologien
Beiträge der Fachtagung

P-167 Robert Krimmer, Rüdiger Grimm (Eds.)
4th International Conference on Electronic Voting 2010
co-organized by the Council of Europe, Gesellschaft für Informatik and E-Voting.CC

P-168 Ira Diethelm, Christina Dörge, Claudia Hildebrandt, Carsten Schulte (Hrsg.)
Didaktik der Informatik
Möglichkeiten empirischer Forschungsmethoden und Perspektiven der Fachdidaktik

P-169 Michael Kerres, Nadine Ojstersek Ulrik Schroeder, Ulrich Hoppe (Hrsg.)
DeLFI 2010 - 8. Tagung der Fachgruppe E-Learning der Gesellschaft für Informatik e.V.

P-170 Felix C. Freiling (Hrsg.)
Sicherheit 2010
Sicherheit, Schutz und Zuverlässigkeit

P-171 Werner Esswein, Klaus Turowski, Martin Juhrisch (Hrsg.)
Modellierung betrieblicher Informationssysteme (MobIS 2010)
Modellgestütztes Management

P-172 Stefan Klink, Agnes Koschmider Marco Mevius, Andreas Oberweis (Hrsg.)
EMISA 2010
Einflussfaktoren auf die Entwicklung flexibler, integrierter Informationssysteme
Beiträge des Workshops der GI-Fachgruppe EMISA (Entwicklungsmethoden für Informationssysteme und deren Anwendung)

P-173 Dietmar Schomburg, Andreas Grote (Eds.)
German Conference on Bioinformatics 2010

P-174 Arslan Brömme, Torsten Eymann, Detlef Hühnlein, Heiko Roßnagel, Paul Schmücker (Hrsg.)
perspeGKtive 2010
Workshop „Innovative und sichere Informationstechnologie für das Gesundheitswesen von morgen“

P-175 Klaus-Peter Fähnrich, Bogdan Franczyk (Hrsg.)
INFORMATIK 2010
Service Science – Neue Perspektiven für die Informatik
Band 1

P-176 Klaus-Peter Fähnrich, Bogdan Franczyk (Hrsg.)
INFORMATIK 2010
Service Science – Neue Perspektiven für die Informatik
Band 2

P-177 Witold Abramowicz, Rainer Alt, Klaus-Peter Fähnrich, Bogdan Franczyk, Leszek A. Maciaszek (Eds.)
INFORMATIK 2010
Business Process and Service Science – Proceedings of ISSS and BPSC

P-178 Wolfram Pietsch, Benedikt Krams (Hrsg.)
Vom Projekt zum Produkt
Fachtagung des GI-Fachausschusses Management der Anwendungsentwicklung und -wartung im Fachbereich Wirtschafts-informatik (WI-MAW), Aachen, 2010

P-179 Stefan Gruner, Bernhard Rumpe (Eds.)
FM+AM`2010
Second International Workshop on Formal Methods and Agile Methods

P-180 Theo Härder, Wolfgang Lehner, Bernhard Mitschang, Harald Schöning, Holger Schwarz (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW)
14. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme“ (DBIS)

P-181 Michael Clasen, Otto Schätzel, Brigitte Theuvsen (Hrsg.)
Qualität und Effizienz durch informationsgestützte Landwirtschaft, Fokus: Moderne Weinwirtschaft

P-182 Ronald Maier (Hrsg.)
6th Conference on Professional Knowledge Management
From Knowledge to Action

P-183 Ralf Reussner, Matthias Grund, Andreas Oberweis, Walter Tichy (Hrsg.)
Software Engineering 2011
Fachtagung des GI-Fachbereichs Softwaretechnik

P-184 Ralf Reussner, Alexander Pretschner, Stefan Jähnichen (Hrsg.)
Software Engineering 2011
Workshopband
(inkl. Doktorandensymposium)

P-185 Hagen Höpfner, Günther Specht, Thomas Ritz, Christian Bunse (Hrsg.)
MMS 2011: Mobile und ubiquitäre Informationssysteme Proceedings zur 6. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2011)

P-186 Gerald Eichler, Axel Küpper, Volkmar Schau, Hacène Fouchal, Herwig Unger (Eds.)
11th International Conference on Innovative Internet Community Systems (I^2CS)

P-187 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
4. DFN-Forum Kommunikations-technologien, Beiträge der Fachtagung 20. Juni bis 21. Juni 2011 Bonn

P-188 Holger Rohland, Andrea Kienle, Steffen Friedrich (Hrsg.)
DeLFI 2011 – Die 9. e-Learning Fachtagung Informatik
der Gesellschaft für Informatik e.V.
5.–8. September 2011, Dresden

P-189 Thomas, Marco (Hrsg.)
Informatik in Bildung und Beruf
INFOS 2011
14. GI-Fachtagung Informatik und Schule

P-190 Markus Nüttgens, Oliver Thomas, Barbara Weber (Eds.)
Enterprise Modelling and Information Systems Architectures (EMISA 2011)

P-191 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2011
International Conference of the Biometrics Special Interest Group

P-192 Hans-Ulrich Heiß, Peter Pepper, Holger Schlingloff, Jörg Schneider (Hrsg.)
INFORMATIK 2011
Informatik schafft Communities

P-193 Wolfgang Lehner, Gunther Piller (Hrsg.)
IMDM 2011

P-194 M. Clasen, G. Fröhlich, H. Bernhardt, K. Hildebrand, B. Theuvsen (Hrsg.)
Informationstechnologie für eine nachhaltige Landbewirtschaftung
Fokus Forstwirtschaft

P-195 Neeraj Suri, Michael Waidner (Hrsg.)
Sicherheit 2012
Sicherheit, Schutz und Zuverlässigkeit
Beiträge der 6. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)

P-196 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2012
Proceedings of the 11th International Conference of the Biometrics Special Interest Group

P-197 Jörn von Lucke, Christian P. Geiger, Siegfried Kaiser, Erich Schweighofer, Maria A. Wimmer (Hrsg.)
Auf dem Weg zu einer offenen, smarten und vernetzten Verwaltungskultur
Gemeinsame Fachtagung Verwaltungsinformatik (FTVI) und Fachtagung Rechtsinformatik (FTRI) 2012

P-198 Stefan Jähnichen, Axel Küpper, Sahin Albayrak (Hrsg.)
Software Engineering 2012
Fachtagung des GI-Fachbereichs Softwaretechnik

P-199 Stefan Jähnichen, Bernhard Rumpe, Holger Schlingloff (Hrsg.)
Software Engineering 2012
Workshopband

P-200 Gero Mühl, Jan Richling, Andreas Herkersdorf (Hrsg.)
ARCS 2012 Workshops

P-201 Elmar J. Sinz Andy Schürr (Hrsg.)
Modellierung 2012

P-202 Andrea Back, Markus Bick, Martin Breunig, Key Pousttchi, Frédéric Thiesse (Hrsg.)
MMS 2012:Mobile und Ubiquitäre Informationssysteme

P-203 Paul Müller, Bernhard Neumair, Helmut Reiser, Gabi Dreo Rodosek (Hrsg.)
5. DFN-Forum Kommunikations-technologien
Beiträge der Fachtagung

P-204 Gerald Eichler, Leendert W. M. Wienhofen, Anders Kofod-Petersen, Herwig Unger (Eds.)
12th International Conference on Innovative Internet Community Systems (I2CS 2012)

P-205 Manuel J. Kripp, Melanie Volkamer, Rüdiger Grimm (Eds.)
5th International Conference on Electronic Voting 2012 (EVOTE2012)
Co-organized by the Council of Europe, Gesellschaft für Informatik and E-Voting.CC

P-206 Stefanie Rinderle-Ma,
Mathias Weske (Hrsg.)
EMISA 2012
Der Mensch im Zentrum der Modellierung

P-207 Jörg Desel, Jörg M. Haake,
Christian Spannagel (Hrsg.)
DeLFI 2012: Die 10. e-Learning
Fachtagung Informatik der Gesellschaft
für Informatik e.V.
24.–26. September 2012

P-208 Ursula Goltz, Marcus Magnor,
Hans-Jürgen Appelrath, Herbert Matthies,
Wolf-Tilo Balke, Lars Wolf (Hrsg.)
INFORMATIK 2012

P-209 Hans Brandt-Pook, André Fleer, Thorsten
Spitta, Malte Wattenberg (Hrsg.)
Nachhaltiges Software Management

P-210 Erhard Plödereder, Peter Dencker,
Herbert Klenk, Hubert B. Keller,
Silke Spitzer (Hrsg.)
Automotive – Safety & Security 2012
Sicherheit und Zuverlässigkeit für
automobile Informationstechnik

P-211 M. Clasen, K. C. Kersebaum, A.
Meyer-Aurich, B. Theuvsen (Hrsg.)
Massendatenmanagement in der
Agrar- und Ernährungswirtschaft
Erhebung - Verarbeitung - Nutzung
Referate der 33. GIL-Jahrestagung
20. – 21. Februar 2013, Potsdam

P-212 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2013
Proceedings of the 12th International
Conference of the Biometrics
Special Interest Group
04.–06. September 2013
Darmstadt, Germany

P-213 Stefan Kowalewski,
Bernhard Rumpe (Hrsg.)
Software Engineering 2013
Fachtagung des GI-Fachbereichs
Softwaretechnik

P-214 Volker Markl, Gunter Saake, Kai-Uwe
Sattler, Gregor Hackenbroich, Bernhard Mit
schang, Theo Härder, Veit Köppen (Hrsg.)
Datenbanksysteme für Business,
Technologie und Web (BTW) 2013
13. – 15. März 2013, Magdeburg

P-215 Stefan Wagner, Horst Lichter (Hrsg.)
Software Engineering 2013
Workshopband
(inkl. Doktorandensymposium)
26. Februar – 1. März 2013, Aachen

P-216 Gunter Saake, Andreas Henrich,
Wolfgang Lehner, Thomas Neumann,
Veit Köppen (Hrsg.)
Datenbanksysteme für Business,
Technologie und Web (BTW) 2013 –
Workshopband
11. – 12. März 2013, Magdeburg

P-217 Paul Müller, Bernhard Neumair, Helmut
Reiser, Gabi Dreo Rodosek (Hrsg.)
6. DFN-Forum Kommunikations-
technologien
Beiträge der Fachtagung
03.–04. Juni 2013, Erlangen

P-218 Andreas Breiter, Christoph Rensing (Hrsg.)
DeLFI 2013: Die 11 e-Learning
Fachtagung Informatik der Gesellschaft
für Informatik e.V. (GI)
8. – 11. September 2013, Bremen

P-219 Norbert Breier, Peer Stechert,
Thomas Wilke (Hrsg.)
Informatik erweitert Horizonte
INFOS 2013
15. GI-Fachtagung Informatik und Schule
26. – 28. September 2013

P-220 Matthias Horbach (Hrsg.)
INFORMATIK 2013
Informatik angepasst an Mensch,
Organisation und Umwelt
16. – 20. September 2013, Koblenz

P-221 Maria A. Wimmer, Marijn Janssen,
Ann Macintosh, Hans Jochen Scholl,
Efthimios Tambouris (Eds.)
Electronic Government and
Electronic Participation
Joint Proceedings of Ongoing Research of
IFIP EGOV and IFIP ePart 2013
16. – 19. September 2013, Koblenz

P-222 Reinhard Jung, Manfred Reichert (Eds.)
Enterprise Modelling
and Information Systems Architectures
(EMISA 2013)
St. Gallen, Switzerland
September 5. – 6. 2013

P-223 Detlef Hühnlein, Heiko Roßnagel (Hrsg.)
Open Identity Summit 2013
10. – 11. September 2013
Kloster Banz, Germany

P-224 Eckhart Hanser, Martin Mikusz, Masud
Fazal-Baqaie (Hrsg.)
Vorgehensmodelle 2013
Vorgehensmodelle – Anspruch und
Wirklichkeit
20. Tagung der Fachgruppe
Vorgehensmodelle im Fachgebiet
Wirtschaftsinformatik (WI-VM) der
Gesellschaft für Informatik e.V.
Lörrach, 2013

P-225 Hans-Georg Fill, Dimitris Karagiannis, Ulrich Reimer (Hrsg.)
Modellierung 2014
19. – 21. März 2014, Wien

P-226 M. Clasen, M. Hamer, S. Lehnert, B. Petersen, B. Theuvsen (Hrsg.)
IT-Standards in der Agrar- und Ernährungswirtschaft Fokus: Risiko- und Krisenmanagement
Referate der 34. GIL-Jahrestagung
24. – 25. Februar 2014, Bonn

P-227 Wilhelm Hasselbring, Nils Christian Ehmke (Hrsg.)
Software Engineering 2014
Fachtagung des GI-Fachbereichs Softwaretechnik
25. – 28. Februar 2014
Kiel, Deutschland

P-228 Stefan Katzenbeisser, Volkmar Lotz, Edgar Weippl (Hrsg.)
Sicherheit 2014
Sicherheit, Schutz und Zuverlässigkeit
Beiträge der 7. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)
19. – 21. März 2014, Wien

P-230 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2014
Proceedings of the 13th International Conference of the Biometrics Special Interest Group
10. – 12. September 2014 in Darmstadt, Germany

P-231 Paul Müller, Bernhard Neumair, Helmut Reiser, Gabi Dreo Rodosek (Hrsg.)
7. DFN-Forum Kommunikationstechnologien
16. – 17. Juni 2014
Fulda

P-232 E. Plödereder, L. Grunske, E. Schneider, D. Ull (Hrsg.)
INFORMATIK 2014
Big Data – Komplexität meistern
22. – 26. September 2014
Stuttgart

P-233 Stephan Trahasch, Rolf Plötzner, Gerhard Schneider, Claudia Gayer, Daniel Sassiat, Nicole Wöhrle (Hrsg.)
DeLFI 2014 – Die 12. e-Learning Fachtagung Informatik
der Gesellschaft für Informatik e.V.
15. – 17. September 2014
Freiburg

P-234 Fernand Feltz, Bela Mutschler, Benoît Otjacques (Eds.)
Enterprise Modelling and Information Systems Architectures
(EMISA 2014)
Luxembourg, September 25-26, 2014

P-235 Robert Giegerich, Ralf Hofestädt, Tim W. Nattkemper (Eds.)
German Conference on Bioinformatics 2014
September 28 – October 1
Bielefeld, Germany

P-236 Martin Engstler, Eckhart Hanser, Martin Mikusz, Georg Herzwurm (Hrsg.)
Projektmanagement und Vorgehensmodelle 2014
Soziale Aspekte und Standardisierung
Gemeinsame Tagung der Fachgruppen Projektmanagement (WI-PM) und Vorgehensmodelle (WI-VM) im Fachgebiet Wirtschaftsinformatik der Gesellschaft für Informatik e.V., Stuttgart 2014

P-237 Detlef Hühnlein, Heiko Roßnagel (Hrsg.)
Open Identity Summit 2014
4.–6. November 2014
Stuttgart, Germany

P-238 Arno Ruckelshausen, Hans-Peter Schwarz, Brigitte Theuvsen (Hrsg.)
Informatik in der Land-, Forst- und Ernährungswirtschaft
Referate der 35. GIL-Jahrestagung
23. – 24. Februar 2015, Geisenheim

P-239 Uwe Aßmann, Birgit Demuth, Thorsten Spitta, Georg Püschel, Ronny Kaiser (Hrsg.)
Software Engineering & Management 2015
17.-20. März 2015, Dresden

P-240 Herbert Klenk, Hubert B. Keller, Erhard Plödereder, Peter Dencker (Hrsg.)
Automotive – Safety & Security 2015
Sicherheit und Zuverlässigkeit für automobile Informationstechnik
21.–22. April 2015, Stuttgart

P-241 Thomas Seidl, Norbert Ritter, Harald Schöning, Kai-Uwe Sattler, Theo Härder, Steffen Friedrich, Wolfram Wingerath (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW 2015)
04. – 06. März 2015, Hamburg

P-242 Norbert Ritter, Andreas Henrich, Wolfgang Lehner, Andreas Thor, Steffen Friedrich, Wolfram Wingerath (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW 2015) – Workshopband
02. – 03. März 2015, Hamburg

GI-Edition Lecture Notes in Informatics – Thematics

T-1 Naumann, Schade (Hrsg.): Informatik in der DDR – eine Bilanz

T-2 Werner Altmann, Albert Mas y Parareda (Hrsg.)
IT-Aus- und Weiterbildung: Chance und Herausforderung für Wirtschaft und Hochschule

T-3 Birgit Demuth (Hrsg.)
Informatik in der DDR - Grundlagen und Anwendungen

T-4 Werner Altmann (Hrsg.)
Lebenslanges Lernen in der Informatik
Beiträge der Hochschulen und Erwartungen der Wirtschaft

T-5 Roland Traunmüller, Maria A. Wimmer (Hrsg.)
Informatik in Recht und Verwaltung: Gestern - Heute - Morgen

T-6 Hans-Dieter Ehrich (Herausgeber) Klaus Alber, Günter Stiege, Roland Vollmar, Dietmar Wätjen (Mitherausgeber)
40 Jahre Informatik an der Technischen Universität Braunschweig 1972 - 2012

T-7 Michael Fothe, Thomas Wilke (Hrsg.)
Keller, Stack und automatisches Gedächtnis – eine Struktur mit Potenzial

The titles can be purchased at:

Köllen Druck + Verlag GmbH
Ernst-Robert-Curtius-Str. 14 · D-53117 Bonn
Fax: +49 (0)228/9898222
E-Mail: druckverlag@koellen.de

Gesellschaft für Informatik e.V. (GI)

publishes this series in order to make available to a broad public recent findings in informatics (i.e. computer science and information systems), to document conferences that are organized in cooperation with GI and to publish the annual GI Award dissertation.

Broken down into
• seminars
• proceedings
• dissertations
• thematics
current topics are dealt with from the vantage point of research and development, teaching and further training in theory and practice. The Editorial Committee uses an intensive review process in order to ensure high quality contributions.

The volumes are published in German or English.

Information: http://www.gi.de/service/publikationen/lni/

ISBN 978-3-88579-426-4

The stack data structure (Keller) is one of the pillars of computer science. To celebrate it, a series of talks was organized at Friedrich Schiller University Jena on 14 November 2014. This volume contains the text of all talks, which cover all aspects of the stack, from historic, to practical, and to scientific. All papers are written in German.