Beispiele zur Rekursion von Reinhold Franck und anderen Autoren

Jürgen F.H. Winkler
Siemens AG, Zentrale Forschung
und Entwicklung, München

"Theoria cum praxi"
G.W. Leibniz

Die Fakultätsfunktion wird häufig verwendet, um das Konzept der Rekursion bei der Programmierung zu erläutern. Die dabei typisch verwendete Formulierung liefert bei den typischen Ganzzahlbereichen auf Rechnern für fast alle Argumentwerte einen Programmfehler. In einem Beitrag im Informatik Spektrum Aug. 1989 wurden Versionen der Fakultätsfunktion vorgestellt, die "richtiger" und gleichzeitig effizienter sind. Da diese Versionen nicht rekursiv sind, wurde gleichzeitig nach guten Lehrbeispielen für die Rekursion gefragt.

Auf diesen kurzen Aufsatz hat Reinhold Franck mit einer ausführlichen Stellungnahme geantwortet, deren wesentlicher Inhalt hier dargestellt wird. Außerdem werden Beispiele aus anderen Zuschriften präsentiert.

Schlüsselwörter: Rekursion, Fakultätsfunktion, binäres Suchen, Reinhold Franck, endliche Zahlenbereiche, von Koch'sche Kurve

1 Einführung

Reinhold Franck äußerte sich in einer ausführlichen Stellungnahme im Herbst 1989 zu dem kleinen Beitrag im Informatik Spektrum "Wie soll die Fakultätsfunktion programmiert werden?" [NW 89]. Als Hochschullehrer ging es ihm dabei vor allem um didaktische Gesichtspunkte. Im vorliegenden Aufsatz werden der Anlaß zu [NW 89] erläutert und die Stellungnahmen von Reinhold Franck und weiteren Autoren, die sich ebenfalls geäußert haben, dargestellt.

Zwei Beobachtungen waren seitens des Autors Anlaß zu [NW 89]:

a) in einem Programmier-Kurs stellten A.Schleiermacher und der Autor Anfang der 80-er Jahre im Zusammenhang mit Rekursion die dafür typische Aufgabe, die Fakultätsfunktion zu implementieren, und erläuterten damit das Prinzip der Rekursion. Da in diesem Kurs auch darauf geachtet werden sollte, daß die Teilnehmer lernen, effizienten Code zu erstellen, behandelten wir auch die Schleifenlösung und ließen den Zeit-Speicher-Aufwand ermitteln. Während der Übungen stellten wir dann noch die Zusatzfrage, ob jemand noch eine effizientere Lösung wüßte (wir dachten an die Reihungslösung). Keiner der Teilnehmer erwähnte sie. Während der 80-er Jahre konnte der Autor noch mehrmals die Beobachtung machen, daß die Reihungslösung nicht genannt wurde.

b) eine genauere Betrachtung der Reihungslösung, bei welcher man ja überhaupt nicht auf die Idee kommt, den vordefinierten Typ INTEGER als Argumenttyp, d.h. hier als Indextyp, zu verwenden, zeigte, daß die Reihungslösung nicht nur effizienter ist, sondern auch "korrekter" oder zumindest wohldefinierter.

Betrachten wir die übliche Formulierung:

```
FUNCTION Fak_1 (N: Integer) Return Integer IS
BEGIN IF N = 0
      THEN Return 1
      ELSE Return N * Fak_1 (N-1);
      END IF;
END Fak_1 ;
```

Man kann diese Lösung die Algol 60-Lösung nennen, da Algol 60 für ganze Zahlen nur INTEGER kannte, oder auch die naive Lösung, da so getan wird, als seien die Typen im Programm gleichwertig zu den entsprechenden mathematischen Zahlenmengen (für ein anderes Beispiel s. z.B. [Win 90]).

Fak_1 hat an zwei Stellen wesentliche Schwächen: (1) in der Spezifikation beim Argumentyp Integer und (2) im Rumpf bei der Abfrage "IF N = 0". Diese Abfrage führt dazu, daß Fak_1 für negative Werte in eine nichtabbrechende Rekursion geht, die in der Regel zu einem Programmabbruch führt.
Betrachtet man die Spezifikation

```
FUNCTION Fak_1 (N: Integer) Return Integer
```

von Fak_1 genauer und berücksichtigt dabei das eben geschilderte Verhalten, dann stellt man folgende Schwächen fest:

- Fak_1 liefert fast immer einen Fehler und keinen Wert vom Typ Integer. Wenn für die interne Darstellung 32 Bit verwendet werden, dann liefert Fak_1 nur für 13 von 4_294_967_296 (d.h. über 4 Milliarden) möglichen Argumentwerten ein richtiges Ergebnis, d.h. es arbeitet für fast alle Argumentwerte falsch.
- die Beschreibung der möglichen Ergebnisse durch den Typ Integer ist unvollständig, da das Ergebnis "Programmabbruch" nicht in dieser Menge liegt. Wenn das Wort "FUNCTION" suggerieren soll, daß Fak_1 ähnliche Eigenschaften wie eine math. Funktion haben sollte, dann sollten alle möglichen Ergebnisse durch die Ergebnisbeschreibung erfaßt werden. Außer den Werten des Typs Integer können aber noch die Fehler "Constraint_Error" und "Storage_Error" (in Ada-Terminologie) auftreten. Die meisten Sprachen bieten nun keine Möglichkeit, dies in der Spezifikation zu beschreiben. Eine der wenigen Sprachen, die das erlauben, ist CHILL [ITU 88]:

  ```
  Fak_1a: PROC (N Integer) RETURNS(Integer) EXCEPTIONS(OVERFLOW, SPACEFAIL);
  ```

  Hier ist klar dokumentiert, daß außer einem zahlenmäßigen Funktionsergebnis, auch andere Ergebnisse auftreten können. Mit den Mitteln der Ausnahmebehandlung könnte der Rumpf von Fak_1 anders formuliert werden; das soll hier aber nicht weiter verfolgt werden.

Diese nicht zu übersehenden Schwächen legen doch die Frage nahe, ob Fak_1 überhaupt jemals in der Lehre gezeigt werden sollte (oder nur als "Negativbeispiel"). Insbesondere wenn man bedenkt, daß Zuverlässigkeit das wohl drängenste Problem der Softwaretechnik ist [Neu 89].

Vom Standpunkt des SW-Ingenieurs sind vor allem technische Eigenschaften eines Programmes wichtig, wie z.B. Fehlerfreiheit [Red 89] und Wirtschaftlichkeit. Gibt es für eine Aufgabe verschiedene Realisierungsmöglichkeiten (wie z.B. Rekursion, Iteration und Reihung für die Fakultät), dann wird er die Möglichkeit wählen, welche sein Ziel, fehlerfreie und effiziente Programme, am besten unterstützt. Auch das, was oft als (mathematische)

Eleganz bezeichnet wird, wird im Zweifelsfalle hinter diese Ziele zurücktreten müssen [Mee 90: 83].

Rekursion ist eine von meist mehreren möglichen Implementierungstechniken, und sie wird generell als eine bedeutende Implementierungstechnik angesehen. Der vorliegende Aufsatz und [Nie 90] zeigen, daß es eine Reihe von weniger problematischen Beispielen zur Anwendung der Rekursion gibt.

2 Beispiele zur Rekursion von Reinhold Franck

Am 31. Oktober 1989 antwortete Reinhold Franck mit einer ausführlichen Stellungnahme unter dem Titel "Rekursion in der Wissenschaft oder die Ausbildung in rekusivem Denken" auf [NW 89]. Entsprechend seinem Engagement als Hochschullehrer beschäftigte sie sich wesentlich mit didaktischen Fragen, und zwar unter folgenden drei Gesichtspunkten:

- Die Programmierung der Fakultätsfunktion
- Didaktisch geeignetere Beispiele für rekursive Funktionen
- Das allgemeine Problem der Ausbildung in Rekursion

Im ersten Punkt verweist er kurz auf die Implementierung der Fakultätsfunktion in [KV 74: 77]. Diese Formulierung verwendet für große Argumente die Stirling'sche Näherungsformel, weist darüberhinaus aber auch einige Schwächen auf.

2.1 Didaktisch geeignetere Beispiele für rekursive Funktionen

Reinhold Franck gibt hier zwei Beispiele an, die er auch selbst in der Ausbildung verwendet hat. "Auswahlkriterium dafür war, daß es sich um möglichst einfach verstehbare Funktionen handelt, deren rekursive Fassung jedoch nicht so offensichtlich fragwürdig ist wie bei der Fakultätsfunktion." Das erste Beispiel betrifft die Spiegelung einer Folge von Zeichen:

Aufgabe: Spiegelung einer Folge von Eingabezeichen bis zu einem Stoppsymbol 'Φ'

```
proc   mirror
       int a;
       read(a);
       if a =/= 'Φ'
       then mirror
       fi;
       print(a)
```

Er hat dieses Beispiel aus den Lehrunterlagen von K.-P. Löhr (jetzt FU Berlin) übernommen. "Dieses Beispiel ist offensichtlich ein wenig künstlich und auch nur zur Erläuterung der Rekursion entstanden. ... Die Prozedur *mirror* ist sehr einfach und übersichtlich; sie verfügt über eine lokale Variable, so daß daran die Übersetzung rekursiver Funktionen sowie deren dynamische Abarbeitung und insbesondere die Funktionsweise eines Variablenkellers gut erklärbar sind."

Das zweite Beispiel ist die binäre Suche. Dafür gibt er zwei Formulierungen an, eine abstrakte und eine auf Reihungen bezogene:

```
proc binsearch = (Eingabe: list, key; Ausgabe: pos);
begin
    int median;
    if Liste leer
    then pos : = nichtgefunden
    else median : = Position in der Mitte der Liste;
         if list[median] = key
         then pos : = median
         else if key < list[median]
              then binsearch (unterer Teil, key, pos)
              else binsearch (oberer Teil, key, pos)
    fi   fi   fi
end;
```

```
proc binsearch = (int lwb, upb; item key; var int pos);
begin
    int median;
    if lwb > upb
    then pos : = nichtgefunden
    else median : = (lwb + upb) div 2;
         if list[median] = key
         then pos : = median
         else if key < list[median]
              then binsearch(lwb, median-1, key, pos)
              else binsearch(median + 1, upb, key, pos)
    fi   fi   fi
end;
```

2.2 Das allgemeine Problem der Ausbildung in Rekursion

Um die Rekursion, die er für einen Schwerpunkt der "Algorithmen"-Säule in der Informatik-Grundausbildung hält, zu lehren und zu lernen, findet Reinhold Franck Lisp (und auch Prolog) sehr gut geeignet; insbesondere wenn man an nichtnumerische Übungsbeispiele denkt. Viele nichtnumerische Beispiele stammen aus der Listenverarbeitung. Wenn in prozeduralen Sprachen Listen durch Reihungen realisiert werden, dann stellt die richtige Bestimmung der Reihungsindizes oft ein Problem dar, welches dann von dem eigentlichen Lehrgegenstand, der Rekursion, ablenkt. Daher gab er auch für die binäre Suche zwei Fassungen an, eine mehr abstrakte und eine, bei welcher eine konkrete Repräsentation gewählt wurde (s. Abschnitt 2.1). Außerdem würde die Verwendung von Reihungen eher den Einsatz von iterativen Techniken und weniger den der Rekursion nahelegen.
Als Beispiele für die Eignung von Lisp gibt er zwei kurze Beispiele an:

Aufgabe: Invertierung einer Liste

```
(LABEL Invert (LAMBDA (P) (COND ((ATOM P) P)
                                (T CONS ( Invert (CDR P) (CAR P)))
)              )              )
```

Aufgabe: Verkettung zweier Listen

```
(LABEL Append (LAMBDA (L1 L2)
                (COND ((NULL L1) L2)
                      (T (CONS (CAR L1) (Append (CDR L1) L2)))
)               ))
```

3 Beispiele zur Rekursion von anderen Autoren

3.1 Berechnung von Polynomen

G.Lamprecht (Universität Bremen) verweist auf Kapitel 10 seines Buches [Lam 88], in welchem mehrere Beispiele für rekursive Programme angegeben sind. In seinem Brief meint er, "daß es kein Beispiel gibt, bei dem sich die Rekursion als "natürlich und effektiv" erweist". Dies ist natürlich eine weitere Herausforderung an die Leser.

Das erste Beispiel von Herrn Lamprecht ist die Berechnung des Tschebyschew-Polynoms

$$T_0(x) = 1$$
$$T_1(x) = x$$
$$T_n(x) = 2 \cdot x \cdot T_{n-1}(x) - T_{n-2}(x) \quad \text{für } n > 1.$$

Dazu gibt er folgendes Programm an:

```
begin
    real x1, y;

    real procedure t(n,x);
        integer n; real x;
        t: = if n>1 then 2 * x * t(n-1, x) - t(n-2, x)
                    else  (if n = 1 then x else 1);

    for x1 : = -1 step 0.1 until 1 do
    begin
        y : = t(5, x1);
        outfix(x1, 2, 6); outfix(y, 3, 10); outimage;
    end;

end
```

Dies ist eine direkte Umsetzung der mathematischen Formel. Wegen der Verwendung von "integer" weicht sie etwas von der mathematischen "Spezifikation" ab, da z.B. auch der Aufruf t(-10, x) zulässig ist und dasselbe Ergebnis wie t(0, x) liefert.
Herr Lamprecht weist dann weiter darauf hin, daß bei dieser direkten Umsetzung viele Doppelberechnungen vorgenommen werden (ähnlich wie bei der rekursiven Berechnung der Fibonnacci-Zahlen; JW) und verweist auf eine iterative Lösung zur Berechnung der Tschebyschew-Polynome.

Weiter wird als Beispiel die Berechnung des Laguerrschen Polynoms

$$L_n(x) = 1/n\,((n-1-x) \cdot L_n\text{-}1(x) - (n-1) \cdot L_{n-2}(x)) \quad \text{für } n > 1$$
$$L_0(x) = 1$$
$$L_1(x) = -x + 1$$

angegeben.

An diesem Beispiel zeigt sich also wieder der Einfluß von Effizienzüberlegungen, der für die Informatik charakteristisch ist ("The fundamental question underlying all of computing is, "What can be (efficiently) automated?" " [CDG 89: 12]).

3.2 Berechnung des Pascal'schen Dreiecks

Dieses Beispiel stammt ebenfalls aus dem Buch von G.Lamprecht. Das Pascal'sche Dreieck enthält bekanntermaßen die Binomialkoeffizienten, d.h. die Koeffizienten der Terme von $(a+b)^n$. Es hat die folgende Gestalt:

```
                1
             1     1
          1     2     1
       1     3     3     1
             usw.
```

Es wird durch folgendes Programm berechnet:

```
begin
    integer array a(0:15);

    procedure pascal(n, a); integer n; integer array a;
    begin
        integer k; integer array b(0:n);
        a(0) := a(n) := 1;
        if n>0 then pascal (n-1, b);
        for k := 1 step 1 until n-1 do
            a(k) := b(k) + b(k-1);
        sysout.setpos((65 - n*4);
        for k := 0 step 1 until n do
        begin
            outint(a(k), 5); sysout.setpos(sysout.pos + 3);
        end;
        outimage;
    end;

    spacing(3);
    pascal(15,a);
end
```

3.3 Die von Koch'sche Kurve

Herr Lamprecht weist bei diesem Beispiel daruf hin, daß hier eine iterative Lösung "recht unübersichtlich werden dürfte". Er schreibt weiter in seinem Buch :
"Die von Koch'sche Kurve wurde Anfang dieses Jahrhunderts als Beispiel für eine stetige Kurve angegeben, die nirgends eine Tangente besitzt. Sie wird folgendermaßen konstruiert:

> Man geht von einem Dreieck ABC aus und drittelt die dem Punkt C gegenüberliegende Seite. Die Teilungspunkte seien mit C_1 und C_2 bezeichnet. Die Strecke C_1C_2 wird entfernt. Mit den Dreiecken ACC_1 und BCC_2 verfährt man in derselben Weise bezüglich der Seiten AC und BC. Indem man die Dreiecke immer weiter unterteilt, ergibt sich im Grenzübergang die von Koch'sche Kurve."
> (Dieses Prinzip ist in Bild 1 dargestellt; JW)

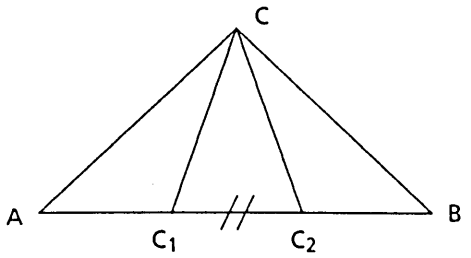

Bild 1. Konstruktionsprinzip der von Koch'schen Kurve

Das Programm zur grafischen Ausgabe der von Koch'schen Kurve 7.Ordnung sieht folgendermaßen aus:

```
begin
    ! Die Klasse graphic ist an dieser Stelle anzugeben
    ! wie sie oben (im Buch; JW) angelistet ist.

    graphic(0, 1, 0, 1);

        procedure koch(n, a, b, c);
            short integer n; ref(point) a, b, c;
        if n>0 then
        begin
            ref(point) c1, c2;
            c1 : = new point(a.x + (b.x-a.x)/3, a.y + (b.y-a.y)/3);
            c2 : = new point(b.x + (a.x-b.x)/3, b.y + (a.y-b.y)/3);
            koch(n-1, a, c, c1); koch(n-1, b, c, c2);
        end
        else
        begin
            new line(a, b).display;
            new line(b, c).display;
            new line(c, a).display;
        end;

        new legend(new point(0, 0.7),
                        "von Koch'sche Kurve der Ordnung 7").display;

        koch(7, new point(0,0), new point(1,0), new point(0.5, 0.6));
    end;
end
```

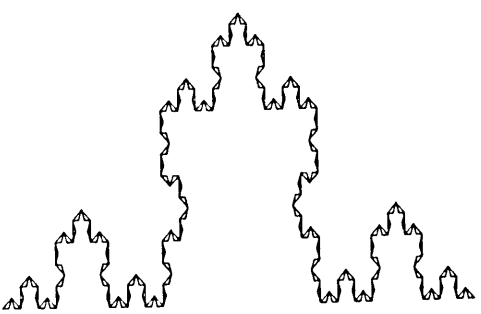

Bild 2. Von Koch'sche Kurve der Ordnung 7

3.4 Realisierung der Fakultätsfunktion in Lisp

Lothar Hotz aus Hamburg beantwortet die im August gestellte Frage mit "Mit Lisp" und meint, daß sie auch nur damit beantwortet werden könne. Er bezieht sich darauf, daß in Lisp-Systemen in der Regel mit "beliebig" langen Zahlen gerechnet werden kann und nicht nur mit den Zahlen, die durch die Wortlänge des Rechners gegeben sind. Die Grenze wird bei einem solchen Lisp-System dann erreicht, wenn der gesamte Speicher mit dem Funktionsergebnis gefüllt wird. Er gibt an, daß " 100 000! wirklich etwa 450 000 Stellen, etwa 86 Druckseiten und ca. 5 Seiten Nullen am Ende hat (berechnet auf RT 6150)".

Dem Problem, daß Argumentbereich und Resultatbereich nicht aneinander angepaßt sind, entgeht auch diese Implementierung nicht. Wegen n!/n = (n-1)! verschärft sich das Problem sogar noch.

Die Arithmetik von Lisp wird auch von Thomas Hemman (Infovation GmbH Bonn) erwähnt. Er teilt mit, daß er mit Allegro Common Lisp auf einem Macintosh SE/30 mit 5 MB Speicher 8429! berechnen kann.

3.5 Realisierung von Rekursion in Hardware

Prof. Liebig und Michael Sperling (beide TU Berlin) untersuchen in [LS 89] die Realisierung der Rekursion direkt durch geeignet konfigurierte Hardwarezellen. Unter den Beispielen, die sie heranziehen, befindet sich auch die Fakultätsfunktion, wobei sowohl die rekursive als auch die iterative Lösung untersucht werden. Es wird auch beobachtet, daß im Falle der Fakultät die Rekursion nur eine Art Zählfunktion hat und daher sehr leicht durch Iteration ersetzt werden kann. Für die Fibonnaccizahlen wird eine Anordnung angegeben, welche das mehrfache Berechnen von Zwischenergebnissen vermeidet.

Danksagung

Herrn Nievergelt bin ich zu Dank verpflichtet, dafür daß er das kleine Beispiel der Programmierung der Fakultätsfunktion in die Rubrik "Overflow" des Informatik-Spektrums aufgenommen hat. Reinhold Franck hat das darin steckende Problem auch erkannt und sich in seiner Stellungnahme engagiert damit auseinandergesetzt, wofür ich ihm posthum herzlich danke. Weiterhin danke ich den Autoren der anderen hier vorgestellten Beispiele für ihr Interesse an dieser Frage.

Literatur

CDG 89 Denning, Peter J; Comer, Douglas E; Gries, David; Mulder Michael C; Tucker, Allen, Turner, Joe A; Young, Paul R: Computing as a Discipline. CACM 32,1(1989) 9..23.

ITU 88 International Telecommunication Union (ed.): CCITT High Level Language (CHILL). Recommendation Z.200, Geneva 1988.

KV 74 van der Meulen, S. G.; Kühling, P.: Programmieren in ALGOL68 - Teil I, Einführung in die Sprache. Walter de Gruyter, Berlin usw., 1974.

Lam 88 Lamprecht, G.: Einführung in die Programmiersprache Simula, 3.Aufl. Vieweg 1988.

LS 89 Liebig, Hans; Sperling, Michael: Darstellung und Verwirklichung rekursiver Algorithmen durch Hardware. Techn. Universtität Berlin, FB 20, Informatik, Bericht 1989/1.

Mee 90 Meek, Brian: Failure is not just one value. SIGPLAN Notices 25,8 (1990) 80..83.

Neu 89 Neumann, P.G.: Risks to the Public in Computers and Related Systems. Software Engineering Notes 14,1 (1989) 6..26.

Nie 90 Nievergelt, J.: Schulbeispiele zur Rekursion. Informatik Spektrum 13,2 (1990) 106..108.

NW 89 Nievergelt, J.; Winkler, J.F.H.: Wie soll die Fakultätsfunktion programmiert werden? Informatik Spektrum 12,4 (1989) 220..221.

Red 89 Redmill, F. J.: Considering Quality in the Management of Software-Based Development Projects. In: Zalewski, Janusz; Ehrenberger, Wolfgang (Eds.): Hardware and Software for Real Time Process Control. North Holland, Amsterdam etc., 1989, 293..304.

Win 90 Winkler, J. F. H.: Functions not equivalent. Letter to the Editor. IEEE Software 7,3 (1990) 10.

H.-J. Kreowski (Hrsg.)

Informatik zwischen Wissenschaft und Gesellschaft

Zur Erinnerung an Reinhold Franck

Proceedings

Springer-Verlag
Berlin Heidelberg New York London Paris
Tokyo Hong Kong Barcelona Budapest

Herausgeber
Hans-Jörg Kreowski
Universität Bremen, Fachbereich Mathematik und Informatik
Bibliothekstraße, Postfach 330 440, W-2800 Bremen 33

CR Subject Classification (1991): K.4

ISBN 3-540-55389-4 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-55389-4 Springer-Verlag New York Berlin Heidelberg

Satz: Reproduktionsfertige Vorlage vom Autor
Druck- u. Bindearbeiten: Weihert-Druck GmbH, Darmstadt
33/3140-543210 – Gedruckt auf säurefreiem Papier