Die prinzipielle Ausschaltbarkeit der rekursiven Prozeduren aus der Programmierungsprache Algol 60

Von R. PÉTER

1. Da das Wirken eines Komputers immer darin besteht, daß in ihm gewisse Daten eingehen, und von diesen abhängig entweder gewisse Ergebnisse von ihm herauskommen, oder sein Ruhezustand nimmermehr wiederkehrt; da ferner sowohl die eingehenden Daten als auch die Folge der herauskommenden Ergebnisse durch natürliche Zahlen kodiert werden können: kann das Wirken eines Komputers immer als das Berechnen der Werte einer zahlentheoretischen Funktion betrachtet werden, die an gewissen Stellen auch undefiniert bleiben kann.

Bei der Idealisierung, daß die Speicherinhälte der Komputer unbeschränkt sind, kann man beweisen,[1] daß die durch Komputer berechenbaren zahlentheoretischen Funktionen mit den „partiell-rekursiven" Funktionen identisch sind. Die genannte Idealisierung gehört dabei zu jenen, die immer auftauchen, wenn für praktische Fragen umfassende mathematische Theorien gebildet werden; dies wurde auch derart formuliert: „Das Unendliche ist eine bequeme Annäherung für das sehr große Endliche".

Wird also untersucht, wie die Berechnung der Werte der partiell-rekursiven Funktionen programmiert werden kann, so wird damit eigentlich die Frage der Programmierung sämtlicher durch Komputer zu lösenden Probleme untersucht.

Die verschiedenen bekannten Rekursionsarten können in der Programmierungsprache Algol 60 (in den weiteren kurz „Algol" genannt) nur durch „rekursive" bzw. durch „simultane Prozeduren" wiedergegeben werden, welche sich nicht unmittelbar auf die Maschinensprache übersetzen lassen. (Unter „nicht unmittelbar" verstehe ich: nicht ohne einer — in den Späteren geschilderten — „Entwirrung", die dem „Kellerungsprinzip" der üblichen „Algol-Compiler" entspricht.) Es gibt eine Tendenz, derartige Prozeduren von den Programmen auszuschalten. In vorliegender Arbeit zeige ich, daß dies prinzipiell immer möglich ist.

2. Betreffs der Kenntnisse über rekursive Funktionen berufe ich mich auf mein Buch.[2] Jedoch gebe ich eine kurze Schilderung der Bedeutung der hier gebrauchten Begriffe.

---

[1] Siehe: J. C. SHEPHERDSON and H. E. STURGIS, *Computability of Recursive Functions*, Journ. of the ACM **10** (1963) S. 217—255, und R. PÉTER, *Programmierung und partiell-rekursive Funktionen*, Acta Math. Ac. Sci. Hung. **14** (1963) S. 373—401.

[2] R. PÉTER, *Recursive Functions* (Budapest—New York—London, 1967; früher deutsch, russisch und chinesisch erschienen).

Wenn nicht anderes gesagt wird, wird im Folgenden „Zahl" immer natürliche Zahl (0 inbegriffen) und „Funktion" zahlentheoretische Funktion bedeuten.

Die verschiedenartigen rekursiven Funktionen kommen aus gewissen Ausgangsfunktionen durch endlichmal verwendete Substitutionen und verschiedenartige Rekursionen zustande.

Das Schema der „primitiven Rekursion", wodurch der Wert der zu definierenden Funktion f an einer Stelle mit Hilfe des Wertes von f an der unmittelbar vorangehenden Stelle angegeben wird, lautet:

$$(D_1) \qquad \begin{cases} f(0, a_1, \dots, a_r) = g_0(a_1, \dots, a_r) \\ f(n+1, a_1, \dots, a_r) = g(n, a_1, \dots, a_r, f(n, a_1, \dots, a_r)), \end{cases}$$

wobei g_0 und g bereits definierte primitiv-rekursive Funktionen sind.

Dies kann auch in folgender „Fallunterscheidungsform" geschrieben werden:

$$(D_1') \qquad f(n, a_1, \dots, a_r) = \begin{cases} g_0(a_1, \dots, a_r), & \text{falls} \quad n=0 \\ g(n \dot{-} 1, a_1, \dots, a_r, f(n \dot{-} 1, a_1, \dots, a_r)) & \text{sonst,} \end{cases}$$

wobei der Wert von $a \dot{-} b$ für $a \geqq b$ die Differenz $a-b$, und sonst 0 bedeutet. Es ist erlaubt, jede Funktion auch als eine Funktion ihrer Argumente und beliebig (endlich) vieler „hinzugenommenen" Argumente, von denen sie nicht tatsächlich abhängt, zu betrachten.

Ohne genaue Angabe zähle ich auch einige kompliziertere Rekursionsarten auf: 1) die „simultane Rekursion" für mehrere Funktionen; 2) die „Wertverlaufsrekursion", wobei zur Berechnung des Funktionswertes an einer Stelle der ganze bisherige Wertverlauf der Funktion herangezogen werden darf; 3) jene Rekursion, wobei auch die (in (D_1) mit $a_1, \dots, a_r$ bezeichneten) Parameter nicht unverändert bleiben; sie können sich sogar von der zu definierenden Funktion f abhängig verändern, wodurch die verwickeltesten Einschachtelungen der f-Werte im Definitionschema auftreten können; 4) die „mehrfache Rekursion", die zugleich nach mehreren Variablen verläuft; 5) die „transfinite Rekursion", wobei eine Stelle nicht im Sinne der ihrer Größe nach geordneten natürlichen Zahlen als Vorgänger einer anderen Stelle betrachtet wird, sondern im Sinne einer anderen Wohlordnung; 6) Rekursionen, wobei zur eindeutigen Berechnung der Funktionswerte auch an späteren Stellen angenommene Werte der Funktion herangezogen werden; usw.

All diese Rekursionen stimmen darin überein, daß sie aus Definitionsgleichungen bestehen, deren Seiten Ausdrücke sind, die formal aus Zahlzeichen, Variablenzeichen, Zeichen für die zu definierende Funktion und für die Hilfsfunktionen (wie f bzw. g_0 und g in (D_1)), Kommata und Klammern aufgebaut werden; und an jeder Stelle ergibt sich der Wert der zu definierenden Funktion durch endlichmalige Anwendung von Schritten folgender Art:

a) Das Einsetzen einer Zahl für eine Variable, überall, wo diese Variable in einer Gleichung vorkommt.

b) Das Ersetzen in einer Gleichung eines Teilausdrucks, der auch als eine Seite einer anderen Gleichung auftritt, durch die andere Seite dieser anderen Gleichung.

Die aus einem Definitionsgleichungsystem an jeder Stelle auf diese Weise eindeutig berechenbaren Funktionen werden „allgemein-rekursiv" genannt; und es ergab

sich bis heute keine überall effektiv berechenbare zahlentheoretische Funktion, die nicht unter diesen Begriff fallen würde.

Wird hier die Bedingung „an jeder Stelle" fallen gelassen, also zugelassen, daß die Funktion nicht überall definiert sei, so gelangt man zur alles bisherige umfassende Klasse der „partiell-rekursiven" Funktionen.

KLEENE hat bewiesen, daß von primitiv-rekursiven Funktionen ausgehend jede partiell-rekursive Funktion durch endlichmalige Anwendung von Substitutionen und von folgender Operation erhalten werden kann:

(Op) $$\mu_i[g(i, n_1, \ldots, n_r)=0],$$

die für gegebene $n_1, \ldots, n_r$ die kleinste Zahl i liefert, für welche $g(i, n_1, \ldots, n_r)$ verschwindet, falls es ein solches i überhaupt gibt, und sonst undefiniert ist.

3. Von einer primitiven Rekursion (D_1), oder von ihrer dazu besser passenden Form (D_1') kann die folgende Algol-Prozedur für die Berechnung der Werte der Funktion f abgelesen wereden:[3]

integer procedure $f(n, a_1, \ldots, a_r)$; **value** $n, a_1, \ldots; a_r$; **integer** $n, a_1, \ldots, a_r$;

f:=**if** $n=0$ **then** $g_0(a_1, \ldots, a_r)$ **else** $g(n-1, a_1, \ldots, a_r, f(n-1, a_1, \ldots, a_r))$;

(angenommen, daß Prozeduren für Berechnung der Werte von g_0 und g bereits zur Verfügung stehen).

Eine derartige Prozedur, welche (in noch unfertigem Zustand) sich selber (hier zur Berechnung von f an einer anderen Stelle) aufruft, wird im Algol eine „rekursive Prozedur" genannt. („Simultane Prozeduren" rufen sich gegenseitig auf.)

Um diese Prozedur für den Komputer durchführbar zu machen, muß sie erst entwirrt werden. Es ergibt sich daraus schrittweise:

Ist $n=0$, so ist

$$f(n, a_1, \ldots, a_r)=g_0(a_1, \ldots, a_r),$$

sonst ist

$$g(n \dot{-} 1, a_1, \ldots, a_r, f(n \dot{-} 1, a_1, \ldots, a_r))$$

zu berechnen.

Darin: ist $n \dot{-} 1 = 0$, so ist

$$f(n \dot{-} 1, a_1, \ldots, a_r) = g_0(a_1, \ldots, a_r),$$

also

$$f(n, a_1, \ldots, a_r) = g(n \dot{-} 1, a_1, \ldots, a_r, g_0(a_1, \ldots, a_r)),$$

sonst ist

$$g(n \dot{-} 2, a_1, \ldots, a_r, f(n \dot{-} 2, a_1, \ldots, a_r))$$

zu berechnen (zufolge $(n \dot{-} 1) \dot{-} 1 = n \dot{-} 2$).

Darin: ist $n \dot{-} 2 = 0$, so ist

$$f(n \dot{-} 2, a_1, \ldots, a_r) = g_0(a_1, \ldots, a_r),$$

also

$$f(n, a_1, \ldots, a_r) = g(n \dot{-} 1, a_1, \ldots, a_r, g(n \dot{-} 2, a_1, \ldots, a_r, g_0(a_1, \ldots, a_r))),$$

[3] Übersichtlichkeitshalber werde ich immer auch Buchstaben mit unteren Indizes gebrauchen, die aber leicht durch im Algol zugelassene Bezeichnungen ersetzt werden können.

usw. Man sieht, daß sich nach n Schritten

$$f(n, a_1, \ldots, a_r) = g\big(n \dot{-} 1, a_1, \ldots, a_r, g(n \dot{-} 2, a_1, \ldots, a_r, \ldots$$

$$\ldots, g(2, a_1, \ldots, a_r, g(1, a_1, \ldots, a_r, g(0, a_1, \ldots, a_r, g_0(a_1, \ldots, a_r))))\ldots)\big)$$

ergibt.

Erst nach dieser (dem bereits erwähnten „Kellerungsprinzip entsprechenden) Entwirrung von (D_1') kann die Berechnung durch den Komputer Schritt für Schritt durchgeführt werden (falls bereits Subrutinen zur Berechnung der Werte von g_0 und g vorhanden sind: zuerst (im „0-ten Schritt") die Berechnung von $g_0(a_1, \ldots, a_r)$, dann mit dem erhaltenen Wert w die Berechnung von $g(0, a_1, \ldots, a_r, w)$, dann mit dem erhaltenen neuen Wert w die Berechnung von $g(1, a_1, \ldots, a_r, w)$, usw.; wurde im i-ten Schritt der Wert w erhalten, dann folgt im $i+1$-ten Schritt die Berechnung von $g(i, a_1, \ldots, a_r, w)$; und der Wert von $f(n, a_1, \ldots, a_r)$ ist jener Wert w, der sich im n-ten Schritt ergibt.

Dies spiegelt sich in folgender Definition einer Hilfsfunktion $h(n, a_1, \ldots, a_r, i, w)$:

$$h(n, a_1, \ldots, a_r, i, w) = \begin{cases} w, & \text{falls} \quad i = n \\ h\big(n, a_1, \ldots, a_r, i+1, g(i, a_1, \ldots, a_r, w)\big) & \text{sonst,} \end{cases}$$

von der ich behaupte, daß sie für die „Ausgangsargumente"

$$n, a_1, \ldots, a_r, 0, g_0(a_1, \ldots, a_r)$$

den Wert $f(n, a_1, \ldots, a_r)$ annimmt.

4. Ich beweise also den folgenden Satz:

$$(S) \qquad h\big(n, a_1, \ldots, a_r, 0, g_0(a_1, \ldots, a_r)\big) = f(n, a_1, \ldots, a_r).$$

Es genügt dazu den folgenden Hilfsatz zu beweisen (mit Rücksicht auf

$$g_0(a_1, \ldots, a_r) = f(0, a_1, \ldots, a_r)\big):$$

Für $i \leqq n$ gilt

$$(H) \qquad h\big(n, a_1, \ldots, a_r, 0, f(0, a_1, \ldots, a_r)\big) = h\big(n, a_1, \ldots, a_r, i, f(i, a_1, \ldots, a_r)\big).$$

Denn für $i=n$ ergibt sich daraus nach der Definition von h:

$$h\big(n, a_1, \ldots, a_r, 0, g_0(a_1, \ldots, a_r)\big) = h\big(n, a_1, \ldots, a_r, n, f(n, a_1, \ldots, a_r)\big) = f(n, a_1, \ldots, a_r),$$

also der Satz (S).

Beweis des Hilfsatzes:

Für $n=0$ kann $i \leqq n$ auch nur 0 sein, und so sind die beiden Seiten von (H) identisch.

Für $n \neq 0$ läßt sich (H) durch Induktion nach i beweisen.

Für $i=0$ sind die beiden Seiten von (H) identisch. Angenommen, daß (H) bereits für ein $i<n$ gilt, vererbt sich dies auch auf $i+1$, denn dann läßt sich die

rechte Seite von (H) nach der Definition von h und nach (D_1) derart weiterentwikkeln:

$$h(n, a_1, \ldots, a_r, i, f(i, a_1, \ldots, a_r)) =$$
$$= h(n, a_1, \ldots, a_r, i+1, g(i, a_1, \ldots, a_r, f(i, a_1, \ldots, a_r))) =$$
$$= h(n, a_1, \ldots, a_r, i+1, f(i+1, a_1, \ldots, a_r)).$$

Somit ist der Hilfsatz, und damit auch (S) bewiesen.

5. So ergibt sich zur Berechnung der Werte von $f(n, a_1, \ldots, a_r)$ die folgende neue Definition:

$$(D_2') \qquad \begin{cases} f(n, a_1, \ldots, a_r) = h(n, a_1, \ldots, a_r, 0, g_0(a_1, \ldots, a_r)) \\ h(n, a_1, \ldots, a_r, i, w) = \begin{cases} w, & \text{falls } i=n \\ h(n, a_1, \ldots, a_r, i+1, g(i, a_1, \ldots, a_r, w)) & \text{sonst.} \end{cases} \end{cases}$$

Davon läßt sich die folgende Algol-Prozedur ablesen:

integer procedure $f(n, a_1, \ldots, a_r)$; **value** $n, a_1, \ldots, a_r$; **integer** $n, a_1, \ldots, a_r$;

begin integer i, w; $i:=0$; $w:=g_0(a_1, \ldots, a_r)$; Z: **if** $i=n$ **then go to** A

else begin $w:=g(i, a_1, \ldots, a_r, w)$; $i:=i+1$; **go to** Z **end**; A: $f:=w$ **end**;

wobei die Marke A auf den Ausgang der Prozedur hinweist, und die Marke Z auf den Beginn eines Zyklus (das besser durch eine Spirallinie veranschaulicht werden könnte, da man bei jedem Rückkehr zu Z auch weiterkommt: w verändert sich; i wächst um 1, bis es zu n heranwächst, wo man durch einen Sprung zu A aus dem Zyklus herauskommt).

Diese ist schon keine „rekursive Prozedur": durch sie wird unterwegs sie selber nie zur Berechnung eines f-Wertes aufgerufen.

6. Man sieht unmittelbar, daß die Berechnung der Werte der einfachen Ausgangsfunktionen, die zur Bildung der primitiv-rekursiven Funktionen verwendet werden, im Algol ohne rekursive und simultane Prozeduren programmiert werden kann, und daß sich diese Eigenschaft (NRP) von Funktionen auch auf solche Funktionen vererbt, die aus ihnen durch Substitutionen entstehen. Nun sahen wir, daß sich die Eigenschaft (NRP) auch bei Verwendung von primitiven Rekursionen vererbt. Daher kann nach Nr. **2** die Berechnung der Werte jeder primitiv-rekursiven Funktion im Algol ohne rekursive und simultane Prozeduren programmiert werden.

7. Nun zeige ich, daß sich die Eigenschaft (NRP) auch bei Verwendung der Operation (Op) der Nr. **2** vererbt.

Nehmen wir also an, daß die Funktion g bereits der Eigenschaft (NRP) ist, und betrachten wir die Funktion

$$(M) \qquad f(n_1, \ldots, n_r) = \mu_i [g(i, n_1, \ldots, n_r) = 0].$$

Bekanntlich kann f durch

$$(F) \qquad f(n_1, \ldots, n_r) = h(n_1, \ldots, n_r, 0, g(0, n_1, \ldots, n_r))$$

definiert werden, wobei die Hilfsfunktion h durch

$$h(n_1, \ldots, n_r, i, a) = \begin{cases} i, & \text{falls } a = 0 \\ h(n_1, \ldots, n_r, i+1, g(i+1, n_1, \ldots, n_r)) & \text{sonst} \end{cases}$$

definiert wird.

(Bezeichne h^* die rechte Seite von (F), und betrachten wir die Werte von h^*, die sich aus der Definition von h ergeben. Ist bereits $g(0, n_1, \ldots, n_r) = 0$, so ist $h^* = 0$. Ist bei $g(0, n_1, \ldots, n_r) \neq 0$ bereits $g(1, n_1, \ldots, n_r) = 0$, so ist $h^* = 1$. Ist bei $g(0, n_1, \ldots, n_r) \neq 0$ und $g(1, n_1, \ldots, n_r) \neq 0$ bereits $g(2, n_1, \ldots, n_r) = 0$, so ist $h^* = 2$. Usw. Man sieht, daß h^* das kleinste i ist, wofür $g(i, n_1, \ldots, n_r) = 0$ gilt, falls ein solches i zu finden ist, und sonst undefiniert bleibt; daß also (F) tatsächlich besteht.)

Von der Definition von h kann folgende Algol-Prozedur zur Berechnung der Werte von h abgelesen werden:

integer procedure $h(n_1, \ldots, n_r, i, a)$; **value** $n_1, \ldots, n_r, i, a$; **integer** $n_1, \ldots, n_r, i, a$;

begin Z: **if** $a = 0$ **then go to** A **else begin** $i := i + 1$; $a := g(i, n_1, \ldots, n_r)$;

go to Z **end**; A: $h := i$ **end**;

wobei man aus dem Zyklus nie herauskommt, wenn es kein i mit $g(i, n_1, \ldots, n_r) = 0$ gibt.

Dies ist keine rekursive Prozedur, weder eine Zugehörige simultaner Prozeduren, und nach (F) entseht f mittels Substitution aus dem dadurch berechenbaren h; demnach ist auch f der Eigenschaft (NRP).

Nach dem zum Schluß der Nr. **2** hervorgehobenen Ergebnis von KLEENE folgt daraus, daß die Berechnung der (existierenden) Werte jeder partiell-rekursiven Funktion im Algol ohne rekursive und simultane Prozeduren programmiert werden kann.

Nach den Erörterungen der Nr. **1** kann man aus diesen Ergebnissen darauf schließen, daß *die Ausschaltung der rekursiven und der simultanen Prozeduren aus den Algol-Programmen prinzipiell immer möglich ist.*

8. Zum selben Ergebnis führt auch ein anderer Weg. Bei der Programmierung ist es Brauch den Gedankengang erst mit Skizzen („flow-diagram") zu veranschaulichen. KALUŽNIN hat für diese Skizzen mit der Benennung „Graphschema" eine exakte Definition eingeführt; und ich habe bewiesen[4] (auf diesen Beweis werde ich mich als auf (N) berufen), daß sich jede partiell-rekursive Funktion auch durch ein Graphschema definieren läßt. Ich zeige nun, daß von jedem der dabei verwendeten Graphschemata zur Berechnung der Werte der dadurch definierten Funktion eine nicht-rekursive Algol-Prozedur abgelesen werden kann, die auch keine Zugehörige simultaner Prozeduren ist.

Ein solches Graphschema ⑤, welches in der Berechnung einer zahlentheoretischen Funktion eine Rolle spielt, ist ein endlicher, zusammenhängender, gerichteter Graph, mit zwei ausgezeichneten Knotenpunkten, kürzer: „Ecken": in die Ecke E („Eingang") läuft keine Kante hinein, und aus der Ecke A („Ausgang") läuft keine Kante hinaus. Die Ecken werden teils „mathematische" teils „logische" Ecken genannt;

[4] R. PÉTER, *Graphschemata und rekursive Funktionen*, Dialectica **12** (1958) S. 373—393.

A gehört unbedingt zu den mathematischen Ecken. Aus jeder mathematischen Ecke außer A läuft genau eine Kante hinaus; aus jeder logischen Ecke laufen zwei Kanten hinaus, eine mit $\uparrow$ und eine mit $\downarrow$ bezeichnete Kante ($\uparrow$ und $\downarrow$ sind Zeichen für die logischen Werte „wahr" bzw. „falsch"). Hineinlaufende Kanten führen zu jeder Ecke außer E.

Den mathematischen Ecken werden „mathematische" Funktionen zugeordnet, die für endliche Zahlenfolgen gewisser Länge definiert sind, und auch als Werte endliche Zahlenfolgen gewisser (nicht unbedingt derselben) Länge annehmen. Den logischen Ecken werden für endliche Zahlenfolgen gewisser Länge definierte logische Funktionen zugeordnet.

Seien die Ecken von $\mathfrak{G}$ in einer festgelegten Reihenfolge:

$$B_1=E,\ B_2,\ \ldots,\ B_s=A.$$

Die zur mathematischen Ecke B_i gehörige Funktion, die k_i-tupeln $(v_{i,1}, \ldots, v_{i,k_i})$ in l_i-tupeln $(w_{i,1}, \ldots, w_{i,l_i})$ überführt, werde ich durch b_i bezeichnen; so gilt:

$$b_i(v_{i,1}, \ldots, v_{i,k_i})=(w_{i,1}, \ldots, w_{i,l_i}).$$

(k_1 werde ich kürzer mit k bezeichnen.)

Ferner ist jedes Graphschema $\mathfrak{G}$ auch „wirksam", und berechnet durch sein Wirken folgenderweise (partiell) die Werte einer Funktion:

Das Wirken von $\mathfrak{G}$ wird dadurch in Gang gesetzt, daß als Argument der zu E gehörigen, für k-tupeln definierten Funktion eine konkrete Zahlenfolge $(n_1, \ldots, n_k)$ angegeben wird. Diese wird, falls E eine mathematische Ecke ist, durch die dazu gehörige Funktion in eine bestimmte Zahlenfolge überführt, die dann (für $E\neq A$) längs der einzigen von E hinauslaufenden Kante in die nächste Ecke weiterläuft, und zum Argument der dieser Ecke zugeordneten Funktion wird. Ist E eine logische Ecke, so läuft $(n_1, \ldots, n_k)$ entlang der mit $\uparrow$ oder der mit $\downarrow$ bezeichneten Kante bis zur nächsten Ecke, je nachdem die zu E gehörige logische Funktion für $(n_1, \ldots, n_k)$ den Wert $\uparrow$ oder den Wert $\downarrow$ annimmt. In beiden Fällen wiederholt sich alles gesagte für jene Ecke statt E, wohin man angelangt ist. Usw. Es kann vorkommen, daß im Lauf ein Kreisweg unendlich oft beschrieben wird. Falls aber der Lauf in endlich vielen Schritten zur Ecke A führt (hier muß er stocken, da aus A keine Kante hinausführt), so ergibt sich hier eindeutig eine Zahlenfolge $(w_{s,1}, \ldots, w_{s,l_s})$ als Wert der zur A gehörigen Funktion; und wir sagen, daß für die durch $\mathfrak{G}$ partiell definierte Funktion f

$$f(n_1, \ldots, n_k)=a(n_1, \ldots, n_k)=(w_{s,1}, \ldots, w_{s,l_s})$$

gilt. Ist hier $l_s=1$, so werden auf diese Weise die Werte einer zahlentheoretischen Funktion f berechnet.

9. Der Wirkungslauf von $\mathfrak{G}$ könnte stocken, auch bevor A erreicht wird, wenn z. B. eine k_i-tupel in eine Ecke einlaufen würde, zu der eine nicht für k_i-tupeln definierte Funktion gehört. Wir beschränken uns hier auf solche Graphschemata, in welchen das nicht vorkommen kann, da falls aus einer Ecke B_i eine Kante in eine Ecke B_j führt, dann für logisches B_i immer $k_j=k_i$, für mathematisches B_i immer $k_j=l_i$ gilt. Diese Eigenschaft ist für die Graphschemata, die im Beweis (N) verwendet wurden, leicht nachzuweisen.

Ferner gehören die in (*N*) verwendeten Graphschemata unter die „Normalschemata", in welchen jeder logischen Ecke B_i eine Relation der Form

$$L_i(v_{i,1}, \ldots, v_{i,k_i}) = \begin{cases} \text{t}, & \text{falls} \quad w_{i,1} = w_{i,2} \\ \text{f} & \text{sonst} \end{cases}$$

zugeordnet wird, wobei sowohl $w_{i,1}$ als auch $w_{i,2}$ eines der $v_{i,1}, \ldots, v_{i,k_i}$ ist; und jeder mathematischen Ecke B_i eine der gewählten sehr einfachen „Ausgangsfunktionen", d.h. eine Funktion der Form

$$b_i(v_{i,1}, \ldots, v_{i,k_i}) = (w_{i,1}, \ldots, w_{i,l_i}),$$

wobei jedes der $w_{i,j}$ $(1 \leqq j \leqq l_i)$ nur 0 oder eines der

$$v_{i,1}, \ldots, v_{i,k_i},$$

oder einer der „Nachfolger"

$$v_{i,k}, v_{i,1}+1, \ldots_{,}+1$$

von diesen sein kann.

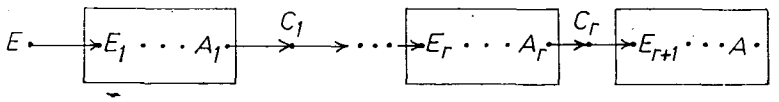
Die Werte $w_{i,j}$ werde ich in ihrer gegebenen Abhängigkeit von $v_{i,1}, \ldots, v_{i,k_i}$ auch als

$$w_{i,j} = b_{i,j}(v_{i,1}, \ldots, v_{i,k_i})$$

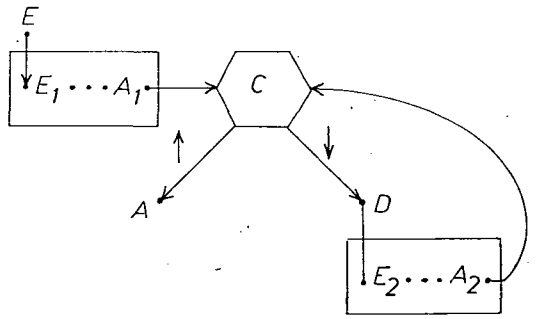
bezeichnen.

Eine Ausgangsfunktion kann durch ein „Ausgangs-Normalschema" definiert werden, in welchem die Ecken *E* und *A* zusammenfallen, und dieser einzigen (natürlich mathematischen) Ecke sie selber zugeordnet wird. Auch das leere Graph wird zu den Ausgangs-Normalschemata gerechnet.

Außer den Ausgangs-Normalschemata war jedes der im Beweis (*N*) verwendeten Normalschemata $\mathfrak{G}$ der Struktur:



oder



wobei durch jedes der $\boxed{\cdot E_i \ldots A_i \cdot}$ — die ich die „Blöcke" von $\mathfrak{G}$ nennen werde — ein Normalschema ebenfalls einer dieser beiden Strukturen oder ein Ausgangsnormalschema angedeutet wurde. (War dieses leer, so wurde die hinein- und die hinauslaufende Kante zu einer einzigen Kante zusammengefügt.)

10. Von einem im Beweis (N) benutzten Normalschema $\mathfrak{G}$ mit Ecken

$$E=B_1, B_2, \ldots, B_s=A$$

und $l_s=1$ kann nun eine Algol-Prozedur zur Berechnung der Werte der dadurch definierten $k=k_1$-stelligen Funktion abgelesen werden. Jedes Zeichen B_i dient auch als Marke der ersten jener Anweisungen, die zur Ecke B_i gehören.

Der Prozedurkopf lautet:

integer procedure $f(n_1, \ldots, n_k)$; **value** $n_1, \ldots, n_k$; **integer** $n_1, \ldots, n_k$;

Im Prozedurrumpf werden die weiteren nötigen Variablen durch

begin integer $v_{1,1}, \ldots, v_{1,k}, \ldots, \ldots, v_{s,1}, \ldots, v_{s,k_s}$;

deklariert. Dann folgen, nach

$$v_{1,1}:=n_1; \ldots; v_{1,k}:=n_k;$$

bei B_1 beginnend die zu den Ecken gehörigen Anweisungen.

Ist B_i für $i\neq s$ eine mathematische Ecke, woraus eine Kante nach B_j läuft, so lauten die zu ihr gehörigen Anweisungen (in Betracht genommen, daß nach Annahme $k_j=l_i$ sein muß):

B_i: $v_{j,1}:=b_{i,1}(v_{i,1}, \ldots, v_{i,k_i}); \ldots; v_{j,k_j}:=b_{i,l_i}(v_{i,1}, \ldots, v_{i,k_i})$; **go to** B_j;

und ist B_i eine logische Ecke, woraus die mit ↑ bezeichnete Kante zur Ecke B_{j_1} führt, die mit ↓ bezeichnete Kante zu B_{j_2}, so gehört zu ihr die folgende Anweisung (in Betracht genommen, daß nach Annahme $k_{j_1}=k_{j_2}=k_i$ sein muß):

B_i: **if** $b_{i,1}(v_{i,1}, \ldots, v_{i,k_i})=b_{i,2}(v_{i,1}, \ldots, v_{i,k_i})$ **then begin** $v_{j_1,1}:=v_{i,1}; \ldots; v_{j_1,k_{j_1}}:=v_{i,k_i}$;

go to B_{j_1} **end else begin** $v_{j_2,1}:=v_{i,1}; \ldots; v_{j_2,k_{j_2}}:=v_{i,k_i}$; **go to** B_{j_2} **end**;

endlich gehört zur Ecke $B_s=A$ die einzige Anweisung:

$$f:=b_{s,1}(v_{s,1}, \ldots, v_{s,k_s}) \quad \textbf{end};$$

(Manche „**go to**" Anweisungen können dabei überflüssig sein und weggelassen werden; sind aber jedenfalls unschädlich.)

Durch diese an sich (nicht als eine Zugehörige gewisser simultanen Prozeduren) vollkommene, nicht-rekursive Prozedur werden die Werte der durch $\mathfrak{G}$ definierten Funktion f berechnet.

11. Als Beispiel dafür, wie in der Praxis ein Graphschema aus einer rekursiven Definition abgelesen wird, betrachten wir den parameterlosen Fall für $g_0=0$ der Definition (D_2') in Nr. **4**:

$$(D_2'') \qquad \begin{cases} f(n)=h(n, 0, 0) \\ h(n, i, w)=\begin{cases} w, & \text{falls} \quad i=n \\ h(n, i+1, g(i, w)) & \text{sonst.} \end{cases} \end{cases}$$

Aus der ersten Zeile der Definition sieht man, daß E eine mathematische Ecke

sein muß, der die Funktion

$$e(n)=(n, 0, 0)$$

zuzuordnen ist. Auf den gewonnenen Tripel ist ein Verfahren zu verwenden, wodurch die Werte der dreistelligen Funktion h berechnet werden können. Das beginnt mit einer Fallunterscheidung, je nachdem $i=n$ oder $i\neq n$ gilt. Daher muß die aus E hinauslaufende Kante in eine logische Ecke B führen, welcher die Relation

$$L(n, i, w)=\begin{cases}\uparrow, & \text{falls} \quad i=n\\ \downarrow & \text{sonst}\end{cases}$$

zugeordnet wird. Für $i=n$ ergibt die Definition für $h(n, i, w)$ den Wert w, also führt aus B die mit $\uparrow$ bezeichnete Kante zur Ecke A, der die Funktion

$$a(n, i, w)=w$$

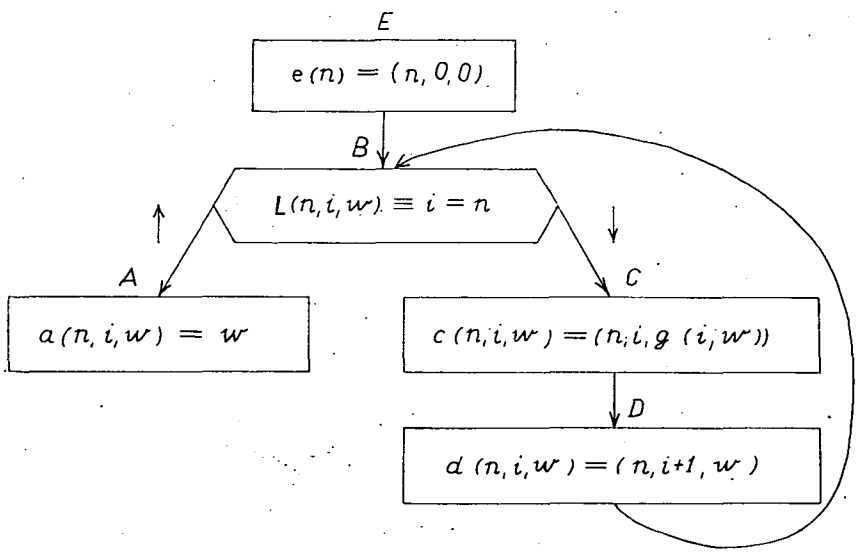
zugeordnet wird. Im Fall $i\neq n$ hat man erst im Tripel (n, i, w) von w auf $g(i, w)$, dann von i auf $i+1$ zu übergehen, daher führt aus B die mit $\downarrow$ bezeichnete Kante zu einer mathematischen Ecke C, und die aus C hinauslaufende Kante zu eine mathematische Ecke D, wobei man der Ecke C die Funktion

$$c(n, i, w)=(n, i, g(i, w))$$

und der Ecke D die Funktion

$$d(n, i, w)=(n, i+1, w)$$

zuordnet. Auf den gewonnenen Tripel ist noch das h-Werte berechnende Verfahren anzuwenden, das in der Ecke B beginnt, daher muß die von C hinauslaufende Kante zu B zurückführen. Aus all diesen ergibt sich das Graphschema:



Das ist aber wegen das Auftreten von $g(i, w)$ allgemein kein Normalschema. Kann $g(i, w)$ bereits durch ein Normalschema $\mathfrak{G}_g$ definiert werden, so kann man aus unserem Graphschema durch „Einschalten" eines „Modifizierten" von $\mathfrak{G}_g$ für die Ecke C ein — ebenfalls die f-Werte berechnendes — Normalschema $\mathfrak{G}$ erhalten. Dies bedeutet folgendes:

Wir wollen nicht den Wert $g(i, w)$, sondern den Tripel

$$(n, i, g(i, w))$$

erhalten. Dieses ergibt sich aus dem derart modifizierten — mit $\mathfrak{G}_g^*$ bezeichneten — $\mathfrak{G}_g$, daß erstens vor die Ecke E_g von $\mathfrak{G}_g$ ein neuer — mathematischer — Eingang E_g^* gesetzt, und dieser Ecke die Funktion

$$e_g^*(n, i, w) = (n, i, i, w)$$

zugeordnet wird, dann jede Folge, die als Argument oder Wert einer der den Ecken von $\mathfrak{G}_g$ zugeordneten Funktionen auftritt, durch Davorsetzen von n, i, verlängert wird, demzufolge falls der Wert der zum Ausgang A_g von $\mathfrak{G}_g$ gehörigen Funktion z war, der Wert der zu A_g gehörigen modifizierten Funktion sich als (n, i, z) ergibt; endlich aus unserem Graphschema die Ecke C samt den zu ihr inzidenten Kanten gelöscht, und dafür eine aus B zu E_g^* und eine aus A_g zu D führende Kante, ferner zwischen E_g^* und A_g das ganze Normalschema $\mathfrak{G}_g^*$ aufgenommen wird. So erhält man zur Berechnung der Werte von f ein Normalschema, woraus eine Algol-Prozedur wie in Nr. **10** abgelesen werden kann.

12. Könnte das nicht ebenso vom entsprechenden Spezialfall der Definition (D_1') in Nr. **2** ausgehend erreicht werden?

Dieser lautet:

$$(D_1'') \qquad f(n) = \begin{cases} 0, & \text{falls} \quad n = 0 \\ g(n \dot{-} 1, f(n \dot{-} 1)) & \text{sonst,} \end{cases}$$

das mit einer Fallunterscheidung beginnt, demnach die Ecke E des dazu gehörigen Graphschemas eine logische Ecke sein muß, welcher die Relation

$$L(n) = \begin{cases} \uparrow, & \text{falls} \quad n = 0 \\ \downarrow & \text{sonst} \end{cases}$$

zugeordnet wird. Für $n=0$ ergibt die Definition für $f(n)$ den konkreten Wert 0, also führt aus B die mit $\uparrow$ bezeichnete Kante zur Ecke A, der die Funktion

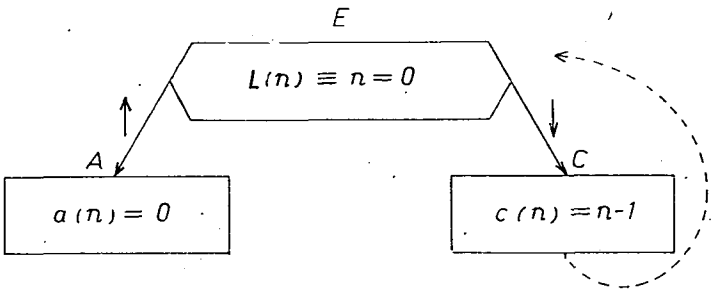
$$a(n) = 0$$

zugeordnet wird. Im Fall $n \neq 0$ hat man erst von n auf $n-1$ zu übergehen, (das für $n \neq 0$ mit $n \dot{-} 1$ übereinstimmt), daher muß aus B die mit $\downarrow$ bezeichnete Kante zu einer mathematischen Ecke C führen, der die Funktion

$$c(n) = n - 1$$

zugeordnet wird.

Nach den Bisherigen muß das Graphschema so beginnen:



Hier scheitert aber unser Vorhaben. Denn nach (D_1'') wäre zunächst auf das gewonnene $n-1$ selbst das f-Werte berechnende Verfahren anzuwenden, das in der Ecke E beginnt. Würde aber die von C hinauslaufende Kante zu E zurückführen, dann würde mit $n-1$ dasselbe geschehen, was vorhin mit n; so würde sich $n-2$ in C, und nachher in E ergeben, dann ebenso $n-3$, usw., wobei die Funktion g garnicht zu Worte kommen könnte, als ob f von g garnicht abhängig wäre. Nach n Schritten würde 0 in E eintreffen, wovon wir schon sahen, daß dies als Wert der durch das Graphschema berechneten Funktion 0 liefert. Diese Funktion wäre also nicht unser $f(n)$, sondern die Konstante 0.

So ist in Hinsicht der Programmierung die Verwendung des Definitionschemas (D_2') viel einfacher als des Definitionschemas (D_1'); obwohl das letztere nur eine andere Aufzeichnung der primitiven Rekursion (D_1) ist.

13. Woran liegt das?

Jedenfalls ist auch (D_2') ein Spezialfall der partiellen Rekursion, da es auch auf Definitionsglechungsystem-Form gebracht werden kann: wird die charakteristische Funktion der Relation $i=n$, die für $i=n$ den Wert 1, und für $i\neq n$ den Wert 0 annimmt, durch $eq(i, n)$ bezeichnet, so kann (D_2') in folgender Form aufgezeichnet werden:

$$(D_2)\qquad \begin{cases} f(n, a_1, \ldots, a_r) = h\big(n, a_1, \ldots, a_r, 0, g_0(a_1, \ldots, a_r)\big) \\ h(n, a_1, \ldots, a_r, i, w) = eq(i, n)\cdot w + |1 - eq(i, n)| \cdot \\ \qquad\qquad \cdot h\big(n, a_1, \ldots, a_r, i+1, g(i, a_1, \ldots, a_r, w)\big). \end{cases}$$

(Dabei sind $a+b$, $a\cdot b$, $|a-b|$ und $eq(a, b)$ primitiv-rekursive Funktionen.)

Es ist in Augen springend, daß diese Definition ein komplizierterer Spezialfall der allgemeinen Rekursion ist, als die primitive Rekursion. (Eine allgemeine Rekursion ist es, obwohl h für $i>n$ nicht definiert ist; denn solche h-Werte werden zur Berechnung von f-Werten nicht gebraucht. Man hätte aber h auch für $i>n$ als w definieren können.) Es ist aber auch prinzipiell komplizierter. Es wird dadurch der Wert von h an einer Stelle

$$(n, a_1, \ldots, a_r, i, w)$$

mit Verwendung eines h-Wertes an solcher Stelle berechnet, die hinsichtlich keiner der Argumente als vorherig betrachtet werden kann: $n, a_1, \ldots, a_r$ sind die selben geblieben, i hat sich um 1 erhöht, an Stelle von w ist

$$g(i, a_1, \ldots, a_r, w)$$

getreten, daß allgemein nicht kleiner als w ist. Dennoch hätte man in Hinsicht der Programmierung diese verquickte Definition primitiv zu nennen. — Hier scheint sich die zu Beginn der Verbreitung der Komputer erstandene Vermutung von L. KALMÁR zu bestätigen, daß dies eine änderung unserer Vorstellung darüber, was in der Mathematik „einfach" ist, mit sich bringen kann. (Er vermutete sogar, daß in den zukünftigen untersten Schulklassen die Mathematikunterricht nicht mit den 4 Spezies, sondern mit solchen Operationen beginnen wird, die hinsichtlich der Maschinen den Vorzug haben.) —

Es kann aber nicht sein, daß sich unabhängig von der Programmierung garnichts an den Definitionschemata zeige, worin (D_2) einfacher als (D_1) ist. Betrachte man ihre Fallunterscheidungsformen (D_2') bzw. (D_1'). In beiden kommt es vor, daß der Wert von f oder h an einer Stelle mit Hilfe eines f- bzw. h-Wertes an einer anderen Stelle angegeben wird (den letzteren werde ich kurz den „verwendeten f- (bzw. h-) Wert" nennen); doch in (D_2') nicht als ein Argument einer anderen Funktion, wie in (D_1') als ein Argument von g. *Dies* ist das Entscheidende (in Nr. **12** war der springende Punkt, daß g garnicht zu Worte kommen konnte), nicht das, wie die Argumente des verwendeten f- (bzw. h-) Wertes beschaffen sind, woran die Primitivität der Rekursion bisher gelegen ist (nämlich daran, daß der verwendete f- (bzw. h-) Wert zur unmittelbar vorangehenden Stelle gehören mußte).

So könnte hinsichtlich der Programmierung jene Definition durch Fallunterscheidung eine primitive Rekursion genannt werden, wodurch der Wert der zu definierenden Funktion f an einer beliebigen Stelle in jedem Fall entweder unabhängig von f, oder als ein Wert von f an einer anderen Stelle angegeben wird, wo von dieser anderen Stelle nur soviel verlangt wird, daß sie von der ursprünglichen Stelle auf bereits bekannter Weise abhängen soll.

14. Würde man aber sagen, daß nun jene Funktionen primitiv-rekursiv genannt werden sollen, die von Ausgangsfunktionen ausgehend durch endlichmaliges Verwenden von Substitutionen und von eben beschriebenen neuartigen primitiven Rekursionen aufgebaut werden können, so käme man zum verblüffenden Ergebnis, daß jede partiell-rekursive Funktion primitiv-rekursiv ist! Denn die partiell-rekursiven Funktionen entstehen aus primitiv-rekursiven Funktionen (nebst Substitutionen) durch Verwendung der in Nr. **2** geschilderten Operation

$$f(n_1, \ldots, n_r) = \mu_i[g(i, n_1, \ldots, n_r) = 0],$$

und von diesem f wurde in Nr. **7** erwähnt, daß es durch die Substitution

$$f(n_1, \ldots, n_r) = h\big(n_1, \ldots, n_r, 0, g(0, n_1, \ldots, n_r)\big)$$

aus der Hilfsfunktion h entsteht, welche durch die neuartige primitive Rekursion

$$h(n_1, \ldots, n_r, i, a) = \begin{cases} i, & \text{falls} \quad a = 0 \\ h\big(n_1, \ldots, n_r, i+1, g(i+1, n_1, \ldots, n_r)\big) & \text{sonst} \end{cases}$$

definiert werden kann.

Wie man sieht, ist es garnicht so einfach zu entscheiden, was in der Mathematik einfach ist.

EÖTVÖS LORÁND UNIVERSITÄT
BUDAPEST, UNGARN

(Eingegangen am 7. Oktober 1971)