MATHEMATISCHES KABINETT

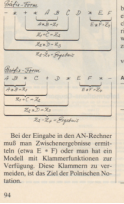
Teil 2

Wie ein Taschenrechner funktioniert

Im letzten Heft an gleicher Stelle hatte Harald Nahrstedt aus Hamm begonnen, in die funktionellen Arbeitsweisen von Taschenrechnern einzuführen. Wie werden eingetippte Operanden (Zahlen) und eingetippte Operationszeichen (Befehle) miteinander verknüpft, damit am Ende das Ergebnis in der Anzeige erscheint? Diese Frage stand im Mittelpunkt.

Es gibt zwei Rechnertypen, die sich in der Behandlung arithmetischer Ausdrücke unterscheiden. Der eine Typ folgt der uns vertrauten algebraischen Notation (AN).
Ergebnis: = (A + B + C) × D − (E + F), was wir als Ausdruck (1) bezeichnen wollen.

Abb. 1



Bei der Eingabe in den AN-Rechner muß man Zwischenergebnisse ermitteln (etwa E + F) oder man hat ein Modell mit Klammerfunktionen zur Verfügung. Diese Klammern zu vermeiden, ist das Ziel der Polnischen Notation.

Bei der Präfix-Form stehen die Operationszeichen vor den Operanden, und (1) würde sich schreiben als:
Ergebnis: = − + + ABCD × EF (2).

Die andere Klasse der im Handel befindlichen Rechnertypen basiert auf der Postfix-Form oder umgekehrten Polnischen Notation (UPN):
Ergebnis: = AB + C + D × EF + − (3).
(1) ist jener Ausdruck, der im letzten Heft als Übungsbeispiel gegeben war.
(2) und (3) sind die gesuchten Schreibformen. Zur Erinnerung veranschaulicht die Abbildung 1 die Auflösungsvorschrift beider Notationen.

Die Multiplikation (A + B + C) × D in (1) läßt sich wegen der Kommutativität auch schreiben D × (C + B + A). Damit würde (3) übergehen in:
Ergebnis: = DCBA + + EF × − (3a).

Von dieser Form geht auch die Abbildung 2 aus, wo die Schrittfolge in einem UPN-Rechner mit vier Speichern im Stapel dargestellt ist. Die Variable D, obwohl erst spätter benötigt, wird zuerst eingegeben und muß daher zunächst „tief in den Keller".

Warum nun haben sich diese zwei verschiedenen Rechnertypen durchgesetzt? Jeder hat einen Vorteil, der zugleich Nachteil beim anderen ist. Der AN-Typ rechnet und wird bedient in derselben Schrittfolge, wie wir es im Kopf oder schriftlich tun würden. Wir müssen vor dem Eintüttern der Daten nicht umdenken wie beim UPN-Typ.

Diesen Vorteil ist aber, daß er selbst bei komplizierten Ausdrücken keine Klammern braucht. Und dann fällt auch sein Hauptvorteil ins Gewicht: Der UPN-Rechner kommt bei der Auflösung arithmetischer Ausdrücke mit weniger Befehlen aus.

Diese Behauptung stand am Schluß im letzten Heft und soll jetzt bewiesen werden. Dabei wollen wir nicht einer strenge mathematischen „Behauptung-Beweis-Form" folgen, sondern eine „Beispiel-Erkenntnis-Form" vorziehen.

Zuvor sollen noch zwei Fehler ausgemerzt werden, die sich in das vorige Heft eingeschlichen hatten:

In der Abbildung 5 mußte die 10. Zeile richtig heißen
− | A + B × (C − D) | A + B × (C − D) − | B × (| C −
und in der Abbildung 7 wäre die richtige Zeichenfolge in der letzten Zeile
= ○ ▷ ▷ ▷ ▷ ▷ ○ ○.

Zunächst einige Festlegungen:
Die Menge der zulässigen Operationen für Ausdrücke in AN ist:
O (AN) = { +, −, ×, :, ↑, ↓, f ( ), (,), = }.

Die Menge für zulässige Operationen für Ausdrücke in UPN ist kleiner:
O (UPN) = { +, −, ×, :, ↑, ↓, f ( ) }.

Zur Erinnerung: Die Pfeile stehen für Potenzieren und Radizieren, f ( ) vertritt die bekannten Funktionen wie etwa sinus. Das „=" gehört zusätzlich zur AN-Menge, weil es alle Zwischenergebnisse abschließt. Beim UPN-Typ stehen die Zwischenergebnisse nach jedem Rechenschritt fest, also auch das Endergebnis.

Zu den Operationen braucht man Variable, die dann nach den uns bekannten Regeln arithmetische Ausdrücke a(AN) und a(UPN) bilden. Wir bezeichnen zwei Ausdrücke a(AN) und a(UPN) äquivalent, wenn die Ergebnisse bei gleicher Wertezuweisung an die Variablen gleich sind, wobei auch die gleichen Variablen in beiden Ausdrücken in derselben Reihenfolge stehen.

Damit ergibt sich von selbst die erste Erkenntnis: Die Anzahl der Variablen ist gleich groß in beiden Ausdrücken, eine Programmschritt-Ersparnis kann also nur in der Anordnung der Operationszeichen liegen.

Seien nun zwei äquivalente a(AN) und a(UPN) gegeben. Beide lassen sich aufgrund ihrer rekursiven Definition auf eine Ausgangsvariable V_1 zurückführen. In jeder Stufe dieses Vorgangs, das zeigt die Abbildung 3 an dem von letzten Mal bekannten Standardbeispiel, sind beide Seiten äquivalent.

Abb. 2

Befehl	DCBA++EF×−			
	Stapelspeicher			
	1	2	3	4
D	D			
C	C	D		
B	B	C	D	
A	A	B	C	D
+	B + A	A	D	D
−	C + B + A	D	D	D
×	D × (C + B + A)	D	D	D
E	E	D × (C + B + A)	D	D
F	F	E	D × (C + B + A)	D
+	E + F	D × (C + B + A)	D	D
−	Ergebnis	D	D	D

In der letzten Zeile, der Grundstufe, ist der Befehlsaufwand gleich groß. Aus der vorletzten Zeile ziehen wir aber die Erkenntnis, daß ein a(AN) ohne die Operationszeichen „(" und „)" einen um einen Befehl größeren Programmschrittaufwand benötigt als der äquivalente a(UPN).

Versuchsweise wird das allein durch das Operationszeichen „=" Außerdem entnehmen wir der Abbildung 3, daß für jede Klammerebene zwei Programmschritte dazukommen. Wir können zusammenfassen: Die Anzahl der Mehrbefehle d_1 beträgt bei einem a(AN) gegenüber einem a(UPN)
$d_1 = 1 + 2 * k_1$
worin k für die Anzahl der Klammerebenen steht.

Diese Formel ist allerdings nicht allgemeingültig. Da das „=" alle abgespeicherten Zwischenwerte entsprechend verknüpft, läßt sich das Beispiel A ↑ (B × (C − D)) =
vereinfachen zu
A ↑ (B × (C − D) =,
womit zwei Programmschritte gespart werden.

Der Fall
A ↑ (B − C) − D =
zeigt jedoch, daß eine Ersetzung nicht immer eine Ersparnis bringt:
A ↑ (B − C = − D =.
Dagegen klappt es wieder bei
(A − B) ↑ C =,
denn wir können von acht auf sieben Schritte reduzieren:
A − B = ↑ C =.

Der Begriff der randorientierten Klammerebene hilft hier weiter. Sie liegt vor, wenn die öffnende Klammer vor der ersten Variablen im a(AN) oder die schließende Klammer hinter der letzten Variablen von links gesehen steht. Damit kommen wir zu folgender Aussage: In einem a(AN) lassen sich die schließenden Klammern randorientierter Klammerebenen durch das Operationszeichen „=" ersetzen. Für die Differenz der Programmschritte gilt nun:
$d_1 = 1 + 2 * k_1 - r_1$
mit r für die Anzahl randorientierter Klammerebenen.

An einem Beispiel wollen wir das verifizieren. Ein a(AN) mit drei Klammerebenen, von denen zwei randorientiert sind, benötigt gegenüber dem a(UPN) einen Befehlsmehraufwand von $d_1 = 1 + 2 * 3 - 2 = 5$. (Siehe Gegenüberstellung in der Abbildung 4.)

Abb. 3

Abb. 4

{A ↑ (B × C) + D − (E − F) =} ≡ {ABC − ↑ DEF − − +}
10 Operationszeichen 8 Operationszeichen

Mitunter läßt sich ein a(AN) so umformen, daß er randorientierte Klammerebenen erhält:
{A + (B − C) ↑ D − E =} ≡ {(B − C) ↑ D + A − E =}.

Weiteren Boden gewinnt ein a(AN) gegenüber dem a(UPN), wenn der UPN-Rechner im Speicher weniger Stapelplätze hat als Variable im a(UPN) nebeneinanderstehen. Solche Ausdrücke lassen sich zwar verarbeiten, bedürfen aber zeitfalls einiger zusätzlicher Vertauschungsbefehle. Und das wiederum läßt sich oft umgehen, wenn man den Ausdruck vor Eingabe in den Rechner umformt.

So würde der Ausdruck (3a) nicht direkt auf einen UPN-Rechner mit nur drei Speicherplätzen passen, weil mit DCBA vier Variable nacheinander eingekellert werden müßten. Der gleichwertigen Form (3) wäre hier der Vorzug zu geben.

Bevor wir die Begrenzung der Speicherstapel in der Formel für die Programmschrittdifferenz berücksichtigen können, müssen wir noch einige Vorüberlegungen treffen.

Einen Ausdruck a(UPN) können wir in Operatoren- und Variablenfolgen zerlegen. So finden wir im Beispiel ABC − ↑ DEF − − × die Folgen „− ↑" und „− − ×" sowie „ABC" und „DEF".

Besitzt ein Ausdruck höchstens eine Operatorenfolge mit n Operatoren, so heißt sie [n + 1]-wertig. Obiges Beispiel hat also die Wertigkeit 4.

Jetzt definieren wir den Begriff der Auflösung eines Ausdrucks a(UPN). Gemeint ist dabei die Zuordnung von je zwei Variablen zu ihrem Operator & O(UPN). Bei einer Variablenfolge geht diese Auflösung schrittweise vor sich, wie schon in der Abbildung 1 gezeigt.

Die Abbildung 5 gibt noch einmal den allgemeinen Fall wieder und eine vereinfachte Darstellung mit der Endvariablen V_{n+1}, nach erfolgter Auflösung der ganzen Folge. Wir nennen das eine Auflösungsebene.

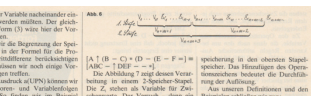
Abb. 5

Nach obiger Definition der Wertigkeit ist eine Auflösungsebene mit n Variablen auch n-wertig. Ein a(UPN) hat in der Regel mehrere Auflösungsebenen, die zur 1. Auflösungsstufe gehören. Im weiteren Verlauf der Auflösung werden daraus weitere Auflösungsebenen der 2. Auflösungsstufe und so fort (Abbildung 6).

Hat ein Ausdruck a(UPN) s Auflösungsstufen, bekommt er den Rang (s + 1) zugewiesen.

Nach all der Theorie wieder zu einem, jetzt etwas komplexeren Beispiel:

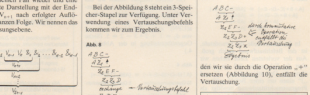
Abb. 6



{A ↑ (B − C) × (D − (E − F =} ≡ {ABC − ↑ DEF − − ×}.

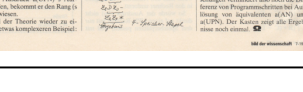
Die Abbildung 7 zeigt dessen Verarbeitung in einem 2-Speicher-Stapel. Die Z_i stehen als Variable für Zwischenwerte. Der Versuch – denn ein solcher ist es nur – endet mit drei Variablen, die sich nicht einkellern lassen. Der Ausdruck ist ohne Zwischenspeicher nicht lösbar. Damit ist aber auch keine Programmschritt-Ersparnis gegenüber der AN-Methode möglich.

Abb. 7



Bei der Abbildung 8 steht ein 3-Speicher-Stapel zur Verfügung. Unter Verwendung eines Vertauschungsbefehls kommen wir zum Ergebnis.

Abb. 8



Das klappt natürlich auch beim 4-Speicher-Stapel in der Abbildung 9.

Die Darstellungsform sei noch einmal erläutert. Das Auflegen einer Variablen von rechts entspricht der Einspeicherung in den obersten Stapelspeicher. Das Hinzufügen des Operationszeichens bedeutet die Durchführung der Auflösung.

Abb. 9

Aus unseren Definitionen und den Beispielen schließen wir nun:
1) die Auflösung eines n-wertigen Ausdrucks a(UPN) kann ohne Vertauschungsbefehle nur in einem n-Speicher-Stapel gelingen.
2) Die Auflösung eines s-rangigen Ausdrucks a(UPN) kann nur in einem Rechner mit mindestens s Speichern geschehen. Es gilt also s ≤ n.

Vertauschungsbefehle sind damit weniger Ersparnis treten nur dann auf, wenn s < n und der benutzte Operator nicht kommutativ ist.

Bei unserem Beispiel in der Abbildung 8 passierte das durch die nichtkommutative Subtraktion „−". Wir

Abb. 10

den wir sie durch die Operation „+" ersetzen (Abbildung 10), entfällt die Vertauschung.

$$d = 1 + 2 * k - r - v$$

k = Anzahl der Klammerebenen des a(AN)
$k = k_{max}$ = Anzahl möglicher Klammerebenen eines Taschenrechners
r = Anzahl der randorientierten Klammerebenen des a (AN)
v = Anzahl der Vertauschungsoperationen, die durch nichtkommutative Operationen hervorgerufen werden
$v \leq r$ ⇒ $0 \leq v \leq n$
s = Anzahl der Auflösungsstufen

Die Anzahl notwendiger Vertauschungen vermindert also noch die Differenz von Programmschritten bei Auflösung von äquivalenten a(AN) und a(UPN). Der Kasten zeigt alle Ergebnisse noch einmal. ☛