Generisches Programmieren in der Algebra

Jens Storjohann *

Helmut–Schmidt–Universität
Universität der Bundeswehr Hamburg
Theoretische Elektrotechnik

*jens.storjohann@hsu-hh.de

Generisches Programmieren: Warum?

- Spezifisches Programmieren für kommerzielle Aufgaben langweilig: Fast immer das Gleiche !
  - Suchen, Selektieren, Vergleichen und Sortieren
  - Kalender, Fristen, Termine
  - Aufsummieren, Verhältnisse (”Prozente”) bilden
  - ...

Technisch-wissenschaftlichesProgrammieren

Datentyp	Feld 1	Feld 2	Feld 3
Skalar	integer	real	complex
Vektor	integer	real	complex
Polynom	integer	real	complex
Matrix	integer	real	complex

Wiederholungen im Aufbau von Datentypen und Algorithmen:

- Gleichungslöser, Eigenwerte, Eigenvektoren, ...

Arithmetik

Wiederholungen:

- Addition, Subtraktion
- Inneres Produkt
- Matrixprodukt
- Kreuzprodukt

Was wollen wir?

- Definitionen von Generischem Programmieren, Objektorientiertem Programmieren, Polymorphismus, ... sind schwierig.

- Wir wollen nicht nur Zahlen (ganze, reelle, komplexe,...), sondern auch Vektoren, Matrizen wie Zahlen auf dem Taschenrechner behandeln können.

- Wo Mathematiker ein Operationszeichen mit konkret unterschiedlicher Bedeutung und abstrakt gleicher Bedeutung verwenden, wollen wir es auch tun.

Lösung für Forth–Programmierer

Ein vollständiges System **kaufen und konfigurieren** (SAP) oder in Matlab programmieren lernen und Toolboxes nutzen oder **selbst programmieren:**

- Alle Objekte sind gespeichert als **Verbund von Deskriptor und Daten.**

- Der Deskriptor charakterisiert die Daten und enthält die Adresse, wo sie zu finden sind. Sie sind nicht im Deskriptor selbst gespeichert.

- Die Deskriptoren selektieren die Rechenoperationen mit Hilfe von Adress–Rechnungen und lassen sie mit **execute** ausführen. Dies geschieht mit Hilfe eines **Deskriptoren-Stapels.**

- Im Programmtext erscheinen "unspezifische" (mathematische) Operationszeichen wie $+ \; - \; *$. **Spezifische** wie d+, f+ erscheinen nicht.

- Das Rechnen mit Daten ohne Deskriptor ist damit weitgehend unmöglich.

- Aber ein Zwischenergebnis ist automatisch mit Deskriptor versehen. Das System legt einen Deskriptor auf den Deskriptorenstapel.

Format des Deskriptors

Nr	Name	Länge	Erklärung
1	data–adress	1 cell	Adresse Anfang der Daten
2	ref_counter	1 cell	Referenz–Zähler
3	nr_field	1 cell	Körper/Ring–Nr.
4	nr_indices	1 cell	Zahl der Indizes
5	nr_structure_const	1 cell	Typ der Strukturkonstanten
6	Bedeutung	1 cell	Bedeut.:Matrix, Polynom,...
7	Speicherart	1 cell	zeilenweise /spaltenw. ...
8	indexgrenze1	1 cell	obere Grenze 1. Index
9	indexgrenze2	1 cell	obere Grenze 2. Index
10	indexgrenze3	1 cell	obere Grenze 3. Index
11	indexgrenze4	1 cell	obere Grenze 4. Index
12-16	pad	5 cells	beliebig (”‘Notizblock”’)

Beispiel für Operation, die durch "$*$" aufgerufen wird:
Matrix–Multiplikation

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} * \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} =$$

$$\begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$$

Einheitliche Multiplikation und Addition, weil die Matrizen mit Daten vom einheitlichen Typ besetzt sind.

Sehr vereinfachtes Beispiel für **vectored execution** abhängig vom Typ

```
1 constant single
2 constant double
variable add_type 2 cells allot
' +  add_type single cells + !
' d+ add_type double cells + !
: add add_type swap cells + @ execute ; \ n1 n2 n -- n1+n2
( Anwendung: )
5 10 single add . 15 ok
50. 100. double add d. 150 ok
```

In der echten Anwendung müssen die Selektierungen der Operationen aus den Deskriptoren kommen.

Zusammenfassung

Generisches Programmieren mit Forth ist möglich.

Late binding mit Hilfe von Deskriptoren passt gut zur Stapel–Orientierung und UPN.

Speicherplatz braucht heute(!) nicht gespart zu werden. Einzeln benannte Größen erhalten durch den Deskriptor ca. 16 Zellen Verwaltungsinformation. Bei einem 64–Bit–Forth sind dies 16*8=128 Byte. Für eine einfach–genaue (real*4) 100×100–Matrix bedeutet dies einen overhead von ca 0,3 % .

Denn keineswegs benötigt jede Matrix– oder Vektor**komponente** Verwaltungsinformation.

Vielen Dank für die Aufmerksamkeit! Gerne Fragen!

Die Hinweise auf langweilige Programme, late binding und generisches Programmieren verdanke ich einem Vortrag ca.1980 eines IBM-Mitarbeiters, der eine Programiersprache mit der Möglichkeit des generischen Programmierens entwickelt hatte. Sein Name ist mir leider entfallen.
Dank an die vielen Forth-Programmierer, die mit ihren Matrizen gekämpft haben!

Die Wikipedia–Seite "Generic Programming" liefert weitergehende Informationen.