PROGRAMMIERTE STRUKTUREN

R. Gnatz, Technische Universität München

1. Einleitung

Methoden zur Konstruktion korrekter Programme sind in den letzten Jahren immer mehr zu einem zentralen Anliegen der Informatik geworden. Die Bemühungen konzentrieren sich zum Teil auf das Problem, die Korrektheit von Programmen zu beweisen. Die Feststellung von HOARE [1] "...the cost of error in certain types of program may be almost incalculable - a lost spacecraft, a collapsed building, a crashed aeroplane, or a world war" - nun, diese Feststellung geht unter die Haut. DIJKSTRA's [2] Aussage "Program testing can be used to show the presence of bugs, but never to show their absence" ist heute in aller Munde (und wird gelegentlich dazu benutzt, Kollegen, die bei allzu langem Testen ertappt werden, zu hänseln).

Andererseits bringt nun DIJKSTRA [3] zum Ausdruck, daß zwar vom mathematischen Standpunkt der formale Korrektheitsbeweis eines gegebenen Programms das attraktivere Problem ist, daß aber in der Praxis der Konstruktion des Programms selbst die größere Bedeutung zukommt, also der Frage, wie zu gegebenen Spezifikationen ein geeignetes Programm gefunden werden kann. Wesentlich dabei ist, daß das Programm so zu konstruieren ist, daß seine Korrektheit evident ist oder nachgewiesen werden kann. Dieser konstruktive Aspekt ist durch DIJKSTRA [4] einen wesentlichen Schritt weiter gebracht worden. Der sich abzeichnende Trend (vgl. BAUER [5]) hat den Seiteneffekt, daß angesichts der Notwendigkeit zur Formalisierung die Zusammenhänge zwischen den

Algorithmen bzw. Programmen einerseits und den, den konkreten Objekten der Algorithmen aufgeprägten, algebraischen Strukturen andererseits deutlicher hervortreten und bewußter werden. Es ist selbstverständlich keine neue Erkenntnis, daß bei der Umformung eines konkreten Algorithmus (etwa zum Zwecke der Optimierung) Rechengesetze, also Eigenschaften der konkret gegebenen Objekte, angewendet werden können bzw. angewendet werden müssen. Die physikalisch konkret gegebenen Objekte auf Maschinenebene sind etwa gewisse Magnetisierungszustände im Magnetkernspeicher eines Rechners.

Es ist gewiss kein trivialer gedanklicher Schritt, wenn man sich nun frei macht von der obigen Voraussetzung, daß die Objekte, auf denen die Algorithmen operieren, konkret gegeben seien und man sich lediglich auf ihre abstrakten Eigenschaften abstützt, daß man also von Besonderheiten der konkreten Objekte abstrahiert.

1.1 Die Entwicklung in der Mathematik

A b s t r a k t i o n s v e r m ö g e n ist primär eine Fähigkeit des menschlichen Geistes. A b s t r a k t i o n in der Mathematik ist so alt wie die Mathematik selbst; die axiomatische Behandlung mathematischer Probleme, die ja Abstraktion voraussetzt, ist jedoch erst zum Beginn dieses Jahrhunderts zur vollen Blüte gekommen, obwohl die Wurzeln bis ins Altertum (EUKLID) zurückreichen.

REDEI [7] nennt E. STEINITZ den Begründer der modernen Algebra, dessen berühmt gewordene Arbeit [8] im Jahre 1910 erschienen ist. STEINITZ hat das fundamentale Isomorphieprinzip in der Algebra verankert: I s o m o r p h e S t r u k t u r e n s i n d a l s n i c h t w e s e n t l i c h v e r s c h i e d e n a n z u s e h e n . Demnach spielt die Beschaffenheit der Elemente der Struktur - oder wie wir sagen der Objekte - bei algebraischen Untersuchungen keine Rolle; es kommt nur auf ihre Eigenschaften, die durch die Strukturaxiome

angegeben werden bzw. aus diesen deduzierbar sind, an. Der Begriff der Isomorphie kommt schon bei GALOIS (1811-1832) vor; das Prinzip wurde aber erst von STEINITZ formuliert.

Es ist historisch interessant, daß sich die axiomatische Denkweise nicht zuletzt an einer ursprünglich so konkreten Disziplin, wie es die Geometrie ist, entwickelt hat. Der Weg dort ist gekennzeichnet durch Namen wie EUKLID, der sich bei seinen Beweisen nicht auf den "gesunden Menschenverstand" sondern auf die Verknüpfungsregeln der Logik abstützte, GAUSS, der das Parallelenaxiom zu Fall brachte, KLEIN, der sich in seinem "Erlanger Programm" mit den Strukturgleichheiten in der Geometrie befaßte und HILBERT, der in seinen "Grundlagen der Geometrie" die Axiomatik der Geometrie zu einem Abschluß brachte.

Die axiomatische Methode (im Gegensatz zur konstruktiven) hat sich in der Mathematik nicht zuletzt deshalb durchgesetzt, weil sie, ökonomisch betrachtet, günstig ist: Ein mathematischer Satz, der lediglich mit Hilfe der Axiome einer algebraischen Struktur bewiesen ist, gilt für alle Modelle dieser Struktur. Wir halten fest, daß die axiomatische Methode in der Mathematik die Abstraktion vom konkreten Modell bedingt und dies entspricht auch der historischen Entwicklung.

1.2 Die Entwicklung in der Informatik

Die Informatik als jüngere Wissenschaft kann sich der Ergebnisse, der Methoden, aber auch der Denkweisen der Mathematik bedienen: Trotzdem scheint sich mit einer gewissen Notwendigkeit in der Informatik eine entsprechende, wenn auch zeitlich auf wenige Jahre geraffte Entwicklung zu ergeben. Methoden, die auf Abstraktion allein aufgebaut sind, scheinen Voraussetzung und Nährboden für die Einführung einer axiomatischen Betrachtungsweise zu sein: So erscheint Abstraktion für sich sehr früh in der Informatik: UNCOL [6] aus dem Jahre 1958 ist ein erster, wenn auch damals wenig erfolgreicher Versuch, das Portabilitätsproblem durch Abstraktion allein zu lösen (The Three-Level Concept ("UNCOL")). Im UNCOL-Report [6] werden erste Diskussio-

nen dieses Konzeptes in das Jahr 1954 zurückdatiert. Die Notwendigkeit zur Abstraktion wird damals - und dies gilt auch heute noch - bestimmt durch die Vielfalt der verschiedenen Rechenanlagen und durch die Tatsache, daß sie in relativ kurzer Zeit veralten. Generell kann man sagen, daß die Idee der höheren Programmiersprache, wie sie z.B. in FORTRAN und ALGOL 60 realisiert wurde, das Ergebnis eines Abstraktionsprozesses ist.

Bei der Konstruktion des "THE - Multiprogramming Systems" demonstriert DIJKSTRA [11,3] die Leistungsfähigkeit des Abstraktionsprinzips in Verbindung mit dem Nachweis des korrekten logischen Entwurfs (Synchronisation!). Eine Arbeit des Autors [12], in der ein abstraktes Zeichengerät definiert wird, beruht ebenfalls auf Abstraktion. Die Realisierung des abstrakten Zeichengerätes erfolgt dabei durch eine geeignete Parameterisierung der Software.

In den Jahren ab 1962 (McCARTHY [10]) erkennt man angesichts der "software crisis" immer deutlicher, daß die Korrektheit von Programmen wegen der erdrückenden kombinatorischen Komplexität durch Testen allein nicht ausreichend gewährleistet werden kann. Abstraktion reduziert die Komplexität (DIJKSTRA [11] und [2]) und hilft so ein Stück weiter. Letztlich bleibt doch nur der formale Beweis der Korrektheit eines Programmes. Diese Einsicht setzt sich zögernd durch (McCARTHY [10] (1962), NAUR [13] (1966), FLOYD [14] (1967), deBAKKER [15] (1968), BURSTALL [16] (1968)). Die eingangs bereits zitierte Arbeit von HOARE [1] aus dem Jahr 1969 ist programmatisch: In der Einleitung stellt er fest: "Computer programming is an exact science in that all the properties of a program and all the consequences of executing it in any given environment can, in principle, be found out from the text of the program itself by means of purely deductive reasoning. Deductive reasoning involves the application of valid rules of inference to sets of valid axioms. It is therefore desirable and interesting to elucidate the axioms and rules of inference which underlie our reasoning about computer programs." Damit ist die axiomatische Methode in der Informatik verankert. Die axiomatische Definition der Semantik einer Programmiersprache (FLOYD [14], HOARE, WIRTH [17]) als "Kontrakt" zwischen Sprachdesigner, Übersetzerbauer und Benutzer, als Ba-

sis für formale Beweise von Programmeigenschaften und als Antrieb zur maschinenunabhängigen Benutzung der Sprache ist eine konsequente und natürliche Entwicklung, die in erster Linie - wie schon im Zusammenhang mit UNCOL erwähnt - durch die Verschiedenartigkeit der Rechenanlagen und ihre Eigenschaft, rasch zu veralten, bedingt ist.

1.3 Weitere Entwicklung

Abstraktion und axiomatisch, deduktive Methode lassen sich nicht nur dazu einsetzen, die Struktur eines Systems zu definieren, durchsichtig und handhabbar zu machen (DIJKSTRA [11], PARNAS [18]), also das Produkt zu strukturieren, sondern sie lassen sich auch dazu verwenden, den Herstellungsprozeß für das Produkt, also die Tätigkeit des Programmierens, zu strukturieren. DIJKSTRA spricht dies klar und losgelöst von Beispielen in der Arbeit [2] aus. Die Konstruktion eines Programmes sollte mit seiner abstraktesten Form beginnen, also mit einem abstrakten Programm, das aus abstrakten Anweisungen, die auf abstrakten Datenstrukturen operieren, aufgebaut ist. Er beobachtet beim Verfeinern eines abstrakten Programms, also beim Obersetzen des abstrakten Programms in eine niedrigere Abstraktionsebene, das Phänomen des j o i n t r e f i n e m e n t , daß nämlich die Verfeinerung (Konkretisierung) der abstrakten Datenstrukturen (Objekte) Hand in Hand zu gehen hat mit der Verfeinerung der abstrakten Operationen für diese Datenstrukturen. Andere Arbeiten, die in diese Richtung gehen, stammen von NAUR [19], ZURCHER, RANDELL [21] und WIRTH [20]. Zu nennen sind aber auch die Arbeiten von MEALY [22] und BALZER [23], beide aus dem Jahr 1967. Konsequent ist in diesem Zusammenhang der Versuch, die hier knapp angedeutete Methodik des S t r u k t u r i e r t e n P r o g r a m m i e r e n s durch geeignet entworfene Programmiersprachen (etwa LISKOV, ZILLES [24]) zu unterstützen. Es sei noch erwähnt, daß die Methodik des "top-down program development" ihre Ergänzung und Unterstützung durch "top-down teaching", wie es etwa in BAUER, GOOS [25] versucht wird, erfahren kann.

2. Strukturiertes Programmieren und programmierte Strukturen

DIJKSTRA's [2] abstraktes Programm ist aus abstrakten Anweisungen (Operationen) aufgebaut und operiert auf abstrakten Objekten. Das heißt, daß von den Besonderheiten der Objekte und der Operationen abstrahiert wird und nur auf die strukturellen, algebraischen Eigenschaften zurückgegriffen wird. Dazu präzisieren wir zunächst den Begriff der (algebraischen) Struktur.

2.1 (Algebraische) Strukturen und ihre Modelle

Wir folgen mit unserer Darstellung den Begriffsbildungen, wie sie bei LORENZEN [26] (aber auch bei GERICKE [27]) zu finden sind: Eine (algebraische) S t r u k t u r wird durch ein A x i o m e n s y s t e m dargestellt, also durch ein System von Aussagen, von denen angenommen wird, daß sie gelten. Für diese Struktur gelten dann alle Aussagen, die aus dem Axiomensystem logisch deduziert werden können. Die Aussagen eines Axiomensystems sind, wenn man sie als Formeln betrachtet, aus Primformeln, aus Primtermen und eventuell aus Primkonstanten, also aus gewissen P r i m s y m b o l e n , aufgebaut.

So wird zum Beispiel durch die Axiome

(A1)	$\forall u,v,w\colon u\cdot(v\cdot w) \simeq (u\cdot v)\cdot w$	(Assoziativität)
(A2)	$\forall u,v\colon u\cdot v \simeq v\cdot u$	(Kommutativität)
(A3)	$\forall u,w\colon \exists v\colon u\cdot v \simeq w$	(Existenz einer Lösung)

die Struktur einer kommutativen Gruppe dargestellt. Man sagt, daß zwei verschiedene Axiomensysteme dieselbe Struktur darstellen, wenn von beiden dieselben Aussagen logisch ableitbar sind. Bekanntlich kann die Struktur einer kommutativen Gruppe auch durch das folgende Axiomensystem gegeben sein:

(B1) wie (A1)
(B2) wie (A2)

(B3) $\forall u: \exists e: u \cdot e \simeq u$ (Existenz der Eins)
(B4) $\forall u: \exists v: u \cdot v \simeq e$ (Existenz des Inversen)

Der Beweis, daß beide Systeme dieselbe Struktur geben, wird dadurch geführt, daß man (A1)-(A3) aus (B1)-(B4) logisch deduziert und umgekehrt.

LORENZEN führt nun den Begriff des G e b i l d e s ein: Eine Menge zusammen mit einem System von Relationen und Funktionen (Verknüpfungen), die auf ihr definiert sind, heißt ein Gebilde. So ist etwa die Menge Z der ganzen Zahlen mit der üblichen Gleichheitsrelation = und mit der üblichen Addition + ein Gebilde $(Z, =, +)$. Werden nun den Primsymbolen eines Axiomensystems eineindeutig gewisse Relationen, Funktionen und Elemente eines Gebildes zugeordnet, so nennt man diese Zuordnung eine I n t e r p r e t a t i o n des Axiomensystems. Die Objektvariablen der Axiome sind dabei als Variable für die Elemente der Menge des Gebildes zu interpretieren. Gehen bei einer Interpretation alle Axiome des Systems in wahre Aussagen für das Gebilde über, dann heißt das Gebilde ein M o d e l l des Axiomensystems und die Menge des Gebildes T r ä g e r der durch das Axiomensystem dargestellten Struktur.

Das oben erwähnte Gebilde $(Z, =, +)$ ist bekanntlich ein Modell des durch (A1)-(A3) gegebenen Axiomensystems oder - wie man auch sagt - Z t r ä g t die Struktur einer kommutativen Gruppe oder noch kürzer Z i s t eine kommutative Gruppe. Durch die Interpretation gehen die Axiome (A1)-(A3) über in die Aussagen

$(A1_Z)$ $\forall u,v,w \in Z: u+(v+v) = (u+v)+w$
$(A2_Z)$ $\forall u,v \in Z: u+v = v+u$
$(A3_Z)$ $\forall u,w \in Z: \exists v \in z: u+v = w$

Alle Aussagen, die aus dem Axiomensystem ableitbar sind, gelten im Modell des Axiomensystems ebenfalls, und zwar in jedem Modell,und darin liegt ja auch der Grund, warum - wie bereits in 1.1 angedeutet - die axiomatische Methode arbeitsökonomisch günstig ist. Eine axiomatische Betrachtungsweise lohnt sich dann, wenn man bei verschiedenartigen Anwendungen die gleichen

Aussagensysteme gefunden hat. Man spricht dann von Strukturgleichheit.

2.2 Abstrakte Programme

DIJKSTRA's abstrakte Programme können aufgefaßt werden als Programme, die in einer Struktur im Sinne von 2.1 operieren. Abstrakte Programme werden geschrieben, indem man lediglich die Existenz eines Modells einer Struktur annimmt, ohne sich jedoch um die Besonderheiten des Modells zu kümmern. In die Programmkonstruktion können dann nur die abstrakten (modellunabhängigen) Eigenschaften, also letztlich die Strukturaxiome eingehen. Dieser Sachverhalt sollte durch die Wahl des Titels dieser Arbeit "Programmierte Strukturen" zum Ausdruck gebracht werden.

Es erfordert selbstverständlich noch eine gewisse Überlegung einzusehen, daß Aussagen über ein abstraktes Programm Aussagen sind, die aus den Axiomen der Struktur abgeleitet werden können und somit für sämtliche Modelle der Struktur gelten. Die Überlegung läßt sich kurz in der folgenden Weise skizzieren. Man betrachtet die Gesamtheit der abbrechenden Berechnungen, die das Programm für die verschiedenen Parameterkonstellationen durchführen kann. Der Wert der Resultatvariablen kann dann als endliche Formel der Eingangsparameter beschrieben werden. Diese Formeln sind aus Primsymbolen der Struktur aufgebaut. Aussagen über diese Formeln können somit als Aussagen über das Programm betrachtet werden.

Ist beispielsweise ein sehr einfaches Programmstück

```
x := a, y := b;
if x=y then x := S(x,y) else y := T(x,y) fi
```

gegeben, wobei S und T zwei Verknüpfungen einer (abstrakten) Struktur sind und gilt unmittelbar vor Ausführung des Programmstückes P(a,b) , dann gilt unmittelbar nach seiner Aus-

führung

(1) $P(a,b) \wedge ((a=b \wedge x=S(a,b) \wedge y=b) \vee (a\neq b \wedge x=a \wedge y=T(a,b)))$

Dies ist nun sicher eine in unserer Struktur zulässige Formel (falls P zulässig ist), da sie neben den logischen Verknüpfungen und der Gleichheitsrelation nur die Verknüpfungen S und T enthält.

Weiß man nun, daß $P(a,b) \simeq a\neq b$ bedeutet, so kann man aus der obigen Formel

(2) $a\neq b \wedge x=a \wedge y=T(a,b)$

deduzieren. Diese Deduktion kann allein mit Hilfe der Regeln eines logischen Kalküls durchgeführt werden, d.h. die Axiome, die unsere Struktur darstellen, wurden dabei gar nicht benötigt. Ist nun die Verknüpfung T kommutativ, gilt also das Strukturaxiom

$$\forall u,v: T(u,v) = T(v,u),$$

dann läßt sich mit Hilfe dieses Axioms die Aussage

(3) $a\neq b \wedge x=a \wedge y=T(b,a)$

deduzieren, also eine strukturabhängige Umformung durchführen.

Es ist hier nicht Absicht, eine Begründung für die axiomatische Behandlung von Programmeigenschaften zu geben; man vergleiche dazu etwa MANNA, PNUELI [28]. Von Interesse ist hier vielmehr die Bedeutung der Strukturaxiome für die Tätigkeit des Programmierens. So erlaubt beispielsweise die Kommutativität von T die Umformung des Programms in die folgende Form

```
x := a, y := b;
if x=y then x := S(x,y) else y := T(y,x) fi,
```

die die Aussage

(1') $P(a,b) \wedge ((a=b \wedge x=S(a,b) \wedge y=b) \vee (a\neq b \wedge x=a \wedge y=T(b,a)))$

als Konsequenz hat und die ja wegen der Kommutativität von T und nur wegen dieser für jedes Prädikat P mit (1) äquivalent ist.

Den Zusammenhang zwischen Strukturaxiomen und abstrakten Programmen wollen wir an einem weiteren Beispiel deutlich machen: Gegeben sei eine Halbgruppe M , d.h. wir nehmen an, daß ein Gebilde

$$(M, =, \cdot)$$

existiert, für welches das assoziative Gesetz (vgl. A1)

(A) $\forall x,y,z\in M: x\cdot(y\cdot z) = (x\cdot y)\cdot z$

gilt. Für jedes Element $a\in M$ kann man den Ausdruck $\underbrace{a\cdot a\cdot \ldots \cdot a}_{n}$ (abgekürzt a^n) betrachten, wobei n eine positive ganze Zahl[1] ist; dieser Ausdruck heißt die n-te Potenz von a .

Man kann nun die n-te Potenz als eine Funktion $p \mid M\times N \to M$ auffassen. Der Halbring der positiven, ganzen Zahlen wird dadurch zum Operatorenbereich der Halbgruppe M . Aufgrund des assoziativen Gesetzes lassen sich für Potenzen die Rechenregeln

(P1) $a^n\cdot a^m = a^{n+m}$
(P2) $(a^n)^m = a^{n\times m}$

---

1) Die Menge N der positiven, ganzen Zahlen mit der üblichen Gleichheitsrelation =, Addition + und Multiplikation × - d.h. wir betrachten das Gebilde (N, =, +, ×) - trägt die Struktur eines kommutativen Halbringes mit Einselement bezüglich der Multiplikation. Es gelten somit für die Addition die Axiome (B1) und (B2), für die Multiplikation die Axiome (B1), (B2) und (B3), und es gilt darüber hinaus das distributive Gesetz, also

(D) $\forall a,b,c\in N: a\times(b+c) = (a\times b)+(a\times c)$

mit $a \in M$ und $n,m \in N$ beweisen. Diese Rechenregeln kann man ausnutzen, um die n-te Potenz rekursiv zu berechnen:

$$(4) \quad a^n = \begin{cases} a & \text{falls } n=1 \\ a \cdot a^{n-1} & \text{sonst} \end{cases}$$

Dies läßt sich direkt in eine Prozedur p1 umschreiben:

```
(4')  proc p1 = (M a, N n) M:
                  if n=1 then a else a·p1(a,n-1) fi
```

Hier wurde nur das Rechengesetz (P1) ausgenützt; eine effizientere Möglichkeit ergibt sich, wenn man auch (P2) ausnützt. Es gilt ja für gerades n

$$a^n = (a^2)^{n/2}$$

und für ungerades $n>1$

$$a^n = a \cdot (a^2)^{(n-1)/2}.$$

Somit erhalten wir

$$(5) \quad a^n = \begin{cases} a & \text{falls } n=1 \\ (a^2)^{n/2} & \text{falls } n \text{ gerade} \\ a \cdot (a^2)^{(n-1)/2} & \text{sonst} \end{cases}$$

oder umgeschrieben[1)]

```
(5')  proc p2 = (M a, N n) M:
                  if n=1     then a
                elsf even n  then p2(qua(a), n/2)
                             else a·p2(qua(a), (n-1)/2) fi,
```

mit

```
      proc qua = (M a) M: a·a .
```

---

1) Man beachte, daß der Übergang von (4) nach (4') bzw. von (5) nach (5') eigentlich nur orthographischer Natur ist.

Wir haben hier eine Form der Potenzberechnung, die häufig angewendet wird, da sich der Test, ob n geradzahlig ist und die Division durch 2 leicht realisieren lassen. Die Rechengesetze (P1) und (P2) geben aber Raum auch für andere Verfahren:

```
(6)   proc p3 = (M a, N n) M:
          if n = 1        then a
        elsf n = 2        then qua(a)
        elsf n mod 3 = 0  then cub(p3(a, n/3))
        elsf n mod 3 = 1  then a·cub(p3(a, (n-1)/3))
                          else qua(a)·cub(p3(a, (n-2)/3)) fi
```

mit

```
      proc cub = (M a) M: a·qua(a).
```

Auf eine nähere Diskussion von (6) oder auf die Frage, wann (6) gegenüber (5') vorteilhafter sein könnte, wollen wir hier nicht näher eingehen. Natürlich sind die drei Rechenvorschriften p1, p2 und p3 äquivalent in dem Sinne, daß sie dieselbe Funktion repräsentieren.

Gibt es nun in der Halbgruppe M ein Einselement, dann haben wir zusätzlich zum Axiom (A) das Axiom

(E) $\exists e \in M: \forall x \in M: x \cdot e = e \cdot x = x$.

Das Einselement ist eindeutig bestimmt, denn gibt es neben e ein zweites e' , dann folgt

$e \cdot e' = e$, aber auch
$e \cdot e' = e'$, also
$e = e'$.

Somit können wir mit Hilfe eines Kennzeichnungsterms eine freigewählte Bezeichnung für dieses Einselement einführen

```
      M eins = ι e∈M: ∀x∈M: x·e = e·x = x.
```

Gilt Axiom (E) , so kann die Definition der Potenz auf die Menge NU{0} der nicht-negativen ganzen Zahlen ausgedehnt wer-

den, was bekanntlich in der Weise geschieht, daß man a^0 = eins setzt. Da nun jede Halbgruppe mit Einselement eine Halbgruppe ist, wird man darüber hinaus fordern, daß die Definition der Potenz für die erstere Struktur kompatibel ist mit der für die letztere. Wir bekommen also die Rechenvorschrift

(7) proc q = (M a, Nu{0} n) M:
    if n = 0 then eins else p(a,n) fi,

wobei p mit p1 , p2 oder p3 identifiziert werden kann, z.B. proc(M,N)M p = p2. Der Rückgriff auf eine bereits vorhandene Rechenvorschrift p bei der Konstruktion von q kann im Sinne eines defensiven, d.h. die Korrektheit eines Programmes konservierenden, Programmierstils sehr zweckmäßig sein. Auch im Hinblick auf die Änderungsfreundlichkeit ist diese Vorgehensweise ratsam, sollte man nämlich später wirklich auf die Idee kommen, etwa anstelle von p2 mit p3 arbeiten zu wollen, so kann die Änderung dadurch geschehen, daß man die Identitätsdeklaration

proc(M,N)M p = p2

durch

proc(M,N)M p = p3

ersetzt.

Nimmt man nun zu den Axiomen (A) und (E) die Existenz des Inversen dazu,

(I) $\forall x \in M: \exists y \in M: x \cdot y = y \cdot x = eins$,

dann erhält man die Struktur einer Gruppe. Da das Inverse wegen

$$y = y \cdot (x \cdot y') = (y \cdot x) \cdot y' = y'$$

eindeutig bestimmt ist, können wir wieder mit Hilfe eines Kennzeichnungsterms die Intention der Funktion, die jedem Element sein inverses zuordnet, formal niederschreiben und eine freigewählte Bezeichnung für diese Funktion einführen. Wir verfahren dabei ähnlich wie bei der Einführung der Bezeichnung eins .

```
proc invers = (Mx) M:
                  ιy∈M: y·x = x·y = eins
```

Wir nennen diese Deklaration i n t e n t i o n a l , weil diese Deklaration lediglich die Eigenschaft der Funktionswerte angibt, nicht aber ein Verfahren zur Konstruktion der Funktionswerte. Entsprechend ist auch die Deklaration für die freigewählte Bezeichnung eins intentional, da dort zwar eine Eigenschaft eines Objektes, die es eindeutig bestimmt, angegeben ist aber kein Verfahren zur Auswahl dieses Objektes aus der Menge M (die ja sowieso nicht konkret gegeben ist).

Die übliche Erweiterung der Potenz für Gruppen auf die Menge Z der ganzen Zahlen unter Wahrung der Kompatibilität liefert dann sofort die Rechenvorschrift

(8)
```
proc r = (Ma,Zn) M:
      if n<0 then q(invers(a), -n)
             else q(a,n)          fi
```

Eine Modifikation der Verfahren zur Berechnung der Potenz kann sich ergeben, wenn weitere Struktureigenschaften bekannt sind, wenn beispielsweise gewisse Elemente idempotent oder von endlicher Ordnung sind.

Wir halten fest, daß in den abstrakten Programmen die Primsymbole, die auch im Axiomensystem der zugeordneten Struktur Verwendung finden, auftreten können, beispielsweise die Bezeichnung M einer Menge (verkleidet als "Artindikation") oder das Verknüpfungssymbol · . Daneben treten intentionale Deklarationen für freigewählte Bezeichnungen von Objekten, Relationen oder Funktionen auf.

2.3 Konkrete Programme

Die V e r f e i n e r u n g eines abstrakten Programmes, wie sie bei DIJKSTRA [2] oder bei WIRTH [20] auftritt , bedeu-

tet im Grunde nichts anderes als die Interpretation einer Struktur im Sinne von 2.1, wobei die Behandlung der intentionalen Deklarationen des abstrakten Programms besonders zu diskutieren sein wird. Die Interpretation der Struktur geschieht dadurch, daß die Primformeln, die Primterme und die Primkonstanten der Axiome mit den entsprechenden Bildungen eines (gegebenen) Gebildes identifiziert werden. Die Interpretation einer Struktur wird direkt auf die zu ihr gehörenden abstrakten Programme ausgedehnt, indem man ihnen die entsprechenden Identitätsdeklarationen hinzufügt. (Die Identitätsdeklarationen beispielsweise von ALGOL 68 sind hier der adequate Ansatz, der jedoch vom Standpunkt des Algebraikers noch der Verallgemeinerung bedarf.) Es ist dann zu zeigen, daß das Gebilde unter der gewählten Interpretation ein Modell der Struktur ist, d.h. daß die Strukturaxiome für das Gebilde in wahre Aussagen übergehen. Ist das Gebilde ein Modell der Struktur, dann gelten alle Aussagen, die über das abstrakte Programm aufgrund der Strukturaxiome gemacht werden können, auch für das durch die Interpretation "verfeinerte" Programm. Die Tatsache, daß dies richtig ist für jedes abstrakte Programm der Struktur und für jedes Modell der Struktur, begründet den mehrfach erwähnten arbeitsökonomischen Gesichtspunkt.

Betrachten wir wieder die Berechnung der Potenz in einer Halbgruppe. Die Menge N der positiven ganzen Zahlen mit der üblichen Gleichheit und Addition bilden ein Modell einer Halbgruppe. Bezeichnet N die Art der positiven ganzen Zahlen, dann haben wir die Identitätsdeklaration

```
mode M = N
```

und bezeichnen equal und plus Gleichheitsrelation und Addition für natürliche Zahlen, dann haben wir die Deklarationen

```
op = = equal,
op · = plus
```

oder ausführlicher

```
op = = (Nx,y) bool: x equal y;
op · = (Nx,y) N: x plus y .
```

Wir haben nun zu zeigen, daß die Addition der natürlichen Zahlen assoziativ ist. [1] Dazu stellen wir die natürlichen Zahlen durch senkrechte Striche dar I, II, III,..., so daß die Anzahl der Striche mit der dargestellten natürlichen Zahl übereinstimmt. Die Addition zweier natürlicher Zahlen erklären wir durch die Konkatenation der Strichsequenzen. Wenn bewiesen ist - und wir nehmen das der Kürze halber hier an - daß diese Konkatenation assoziativ ist, dann ist auch die Assoziativität der Addition gezeigt. Zwei Bemerkungen dazu: (a) Die obigen Strichsequenzen mit der Konkatenation als Verknüpfung bilden ihrerseits ein Modell der natürlichen Zahlen, das einen konstruktiven Beweis der Rechengesetze ermöglicht (LORENZEN [26]). (b) Die Dualdarstellung der Zahlen und ihre darauf aufbauende physikalische Realisierung in Rechenanlagen liefern ein Modell für einen Teil der natürlichen Zahlen: Unsere Überlegungen ziehen sich somit durch bis zu den Mikroprogrammen von Rechenanlagen (BAUER [5]). Ein anderes Modell einer Halbgruppe bilden die Objekte der Art string mit der Konkatenation als Verknüpfung oder die Selbstabbildungen einer Menge mit dem Hintereinanderausführen als Verknüpfung.

Die rationalen Zahlen mit der üblichen Multiplikation als Verknüpfung bilden ein Modell einer Gruppe. Die Interpretation der zur Gruppenstruktur gehörenden abstrakten Programme kann durch die folgenden Identitätsdeklarationen erfolgen:

```
mode M = struct(int z, n);[2]
op = = (Ma,b) bool:
        z of a × n of b = n of a × z of b,
op · = (Ma,b) M:
        (z of a × z of b, n of a × n of b)
```

---

[1] Neben der Assoziativität sind natürlich auch noch die Axiome für das Modell nachzuweisen, die implizit unterstellt werden, nämlich die Geschlossenheit der Verknüpfung oder daß equal eine Äquivalenzrelation ist.

[2] Die Null ist auszuschließen, was durch die Deklaration mode M = (struct(int z, n)a) bool: n of a ≠ 0 (vgl. etwa GNATZ [29, 30]) geschehen könnte.

Bei der Potenz für Gruppen haben wir die zwei intentionalen Deklarationen für eins und invers zu behandeln: Der einfachste Weg ist, die intentionalen Deklarationen zu ersetzen:

```
M eins = (1, 1),
proc invers = (M a) M:
              if z of a ≠ 0 then (n of a, z of a) fi
```

Der Nachweis, daß diese Interpretationen zulässig sind, wäre hier formal durchzuführen; wir begnügen uns jedoch mit dem Verweis auf einschlägige Lehrbücher der Algebra (z.B. REDEI [7]).

Ein anderes, zum vorangehenden nicht-isomorphes Modell einer Gruppe bilden z.B. die Permutationen der Zahlen 1 bis 4:

```
mode M = ([1:4]int a) bool:
         ⌈ bool b := true;
           for i to 4 do
           ⌈ b := 1≤a[i] ∧ a[i]≤4 ∧ b;
             for j from i+1 to 4 do
                   b := a[i]≠a[j] ∧ b ⌋;
           b                          ⌋

op = = (Ma,b) bool:
         ⌈ bool b := true;
           for i to 4 do b := a[i]=b[i] ∧ b;
           b                                ⌋

op · = (Ma,b) M:
       ⌈ M c;
         for i to 4 do c[i] := a[b[i]];
         b                             ⌋
```

Die intentionalen Deklarationen ersetzen wir durch

```
M eins = (1, 2, 3, 4),
proc invers = (Ma) M:
              ⌈ M c;
                for i to 4 do c[a[i]] := i;
                c                          ⌋
```

Auch hier wäre wieder der Nachweis zu führen, daß das Gebilde eine Gruppe ist, bzw. daß die Substitution der intentionalen Deklarationen zulässig ist.

Wir halten also fest, daß die Modelle einer algebraischen Struktur nicht isomorph zu sein brauchen. Das klassische Beispiel hierfür aus der Algebra ist der euklidische Algorithmus zur Berechnung des größten gemeinsamen Teilers: Er funktioniert für die ganzen Zahlen, für die ganzen GAUSS'schen Zahlen, für Polynome über den ganzen Zahlen usw. Auch hier kann also das gleiche abstrakte Programm an verschiedene Modelle adaptiert werden.

Aufgrund der Strukturaxiome bzw. der daraus abgeleiteten Rechengesetze können gelegentlich mehrere logisch äquivalente Versionen eines abstrakten Programmes entwickelt werden, wie etwa die Rechenvorschriften p1 , p2 und p3 für die Potenz in Halbgruppen. Die Frage, welche der logisch äquivalenten Versionen für ein konkretes Problem auszuwählen ist, interessiert den Mathematiker in der Regel überhaupt nicht, um so mehr jedoch den Informatiker wegen der damit verbundenen Optimierungsmöglichkeit. Diese Frage kann gelegentlich auf der abstrakten Ebene gar nicht entschieden werden, sondern ist erst zu klären, wenn das konkrete Modell bekannt ist. Beispielsweise ist es zweckmäßig, die gewöhnliche Multiplikation natürlicher Zahlen, wenn sie auf die Addition zurückgeführt werden muß, mit Hilfe der Rechenvorschrift p2 zu realisieren. Will man jedoch Polynome potenzieren, dann kann die Verwendung der Rechenvorschrift p3 wesentlich zweckmäßiger sein.

Gelegentlich wird es vorkommen, daß äquivalente Programmtransformationen erst nach der Adaptation an ein Modell, also auf einer niedrigeren Abstraktionsebene, durchgeführt werden können. So kann es etwa für die rationalen Zahlen zweckmäßig sein, die Operationen qua und cub durch

```
proc qua = (M a) M: (z of a ↑ 2, n of a ↑ 2)
proc cub = (M a) M: (z of a ↑ 3, n of a ↑ 3)
```

zu ersetzen. Bei der Multiplikation natürlicher Zahlen geht qua auf der Mikroprogrammebene über in die Linksverschiebung eines Binärwortes.

3. Schlußbemerkung

Der Autor ist sich darüber im klaren, daß die hier aufgezeigten Vorgehensweisen nicht neu sind. Er sieht jedoch in dem Versuch, die Zusammenhänge zwischen der Kunst des Programmierens und den deduktiven Methoden der abstrakten Algebra sichtbar und bewußter zu machen, einen Beitrag zur Methodik des Programmierens oder um mit DIJKSTRA [3] zu sprechen "a relevant step in the process of transforming the Art of Programming into the Science of Programming". Dieser Beitrag sollte ganz allgemein dazu anregen, formale mathematische Methoden und Denkweisen bei der Tätigkeit des Programmierens zum Einsatz zu bringen, wie z.B. das Isomorphieprinzip oder - wie es bei REDEI [7] genannt wird - das STEINITZ'sche Prinzip. Die Entdeckung von Strukturgleichheiten in den verschiedenartigsten Einsatzgebieten elektronischer Rechenanlagen mit dem Ziel, Standardaufgaben heraus zu kristallisieren und allgemein gültig zu behandeln, ist seit langem ein erklärtes, zentrales Anliegen der Informatik.

Referenzen

[1] C.A.R. Hoare, "An Axiomatic Basis for Computer Programming", Comm. ACM, 12, 10, (1969) 576-581.

[2] E. W. Dijkstra, "Structured Programming", in: I. N. Buxton and B. Randell (ed.), "Software Engineering Techniques", Report on a conference at Rome, Oct. 27-31, 1969.

[3] E. W. Dijkstra, "A Constructive Approach to the Problem of Program Correctness", BIT 8 (1969) 174-186.

[4] E. W. Dijkstra, "A Simple Axiomatic Basis for Programming Language Constructs" (Report EWD 372). Lectures, given at the Marktoberdorf Summer School 1973.

[5] F. L. Bauer, "A Philosophy of Programming", Lectures given at the Imperial College of Science and Technology, University of London (1973).

[6] I. Strong et al., "The Problem of Programming Communication with Changing Machines", Comm. ACM, 1, 8 (1958) 12-18.

[7] L. Redei, "Algebra I", Akademische Verlagsgesellschaft Geest u. Portig K.G., Leipzig (1959).

[8] E. Steinitz, "Algebraische Theorie der Körper", J. reine angew. Math., 137, (1910) 167-3o9.

[9] IU. I. Ianov, "On the Equivalence and Transformation of Program Schemes", Doklady, AN USSR, 113, 1, (1957) 39-42 (ins Englische übersetzt von Morris D. Friedman).

[10] J. McCarthy, "Towards a mathematical theory of computation", Proc. IFIP Cong. 1962, North Holland Pub. Co., Amsterdam, (1963).

[11] E. W. Dijkstra, "The Structure of the 'THE'-Multiprogramming System", Comm. ACM, 11, 5 (1968) 341-346.

[12] R. Gnatz, "DII: A Plotter - Independent Intermediate Language for Graphical Output", Calcolo, Suppl. IX, (1972) 69-92.

[13] P. Naur, "Proof of algorithms by general snapshots", BIT 6 (1966) 310-316.

[14] R. W. Floyd, "Assigning Meanings to Programs" Proc. Amer. Math. Soc., Symposia in Applied Mathematics, 19, (1967) 19-32.

[15] J. W. deBakker, "Axiomatics of simple assignment statements". M.R. 94, Mathematisch Centrum, Amsterdam (1968).

[16] R. Burstall, "Proving properties of programs by structural induction", Experimental Programming Reports: Nr. 17 DMIP, Edinburgh, (1968).

[17] C. A. R. Hoare and N. Wirth, "An Axiomatic Definition of the Programming Language PASCAL", Acta Informatica 2, 4, (1973) 335-355.

[18] D. L. Parnas, "Information distribution aspects of design methodology", Proc. IFIP Cong. 71, Booklet TA-3, (1971) 26-30.

[19] P. Naur, "Programming by Action Clusters" BIT, 9, (1969) 250-258.

[20] N. Wirth, "Program Development by Stepwise Refinement" Comm. ACM, 14, 4, (1971) 221-227.

[21] F. W. Zurcher and B. Randell, "Iterative Multi-Level Modelling, a Methodology for Computer System Design", Proc. IFIP Cong. 68, North-Holland Pub. Co., Amsterdam, (1969) 867-871.

[22] G. Mealy, "Another look at data", Proc. AFIPS, 31, (1967) 525-534.

[23] R. M. Balzer, "Dataless programming", Proc. AFIPS, 31, (1967) 557-566.

[24] B. Liskov, S. Zilles, "Programming with Abstract Data Types", SIGPLAN Notices, Proc. Symposium on Very High Level Languages, 9, 4, (1974) 5o-59.

[25] F. L. Bauer, G. Goos, "Informatik, eine einführende Übersicht" 2 Bände, Springer (1971).

[26] P. Lorenzen, "Metamathematik", BI-Hochschultaschenbücher, 25, Mannheim (1962).

[27] H. Gericke, "Theorie der Verbände" BI-Hochschultaschenbücher, 38/38a, Mannheim (1963).

[28] Z. Manna and A. Pneuli, "Axiomatic Approach to Total Correctness of Programs", Acta Informatica, 3, 3 (1974) 243-263.

[29] R. Gnatz, "On Basic Concepts of Higher Graphic Languages" in: F. Nake and A. Rosenfeld: "Graphic Languages", North-Holland Pub. Co., Amsterdam (1972) 302-320.

[30] R. Gnatz, "Sets and Predicates in Programming Languages" Lectures given at the Marktoberdorf Summer School (1973).