H. Noltemeier, editor, Graphen, Algorithmen, Datenstrukturen, Appl. Comp. Sc. 4, pages 53–64, München, 1976. Hanser–Verlag

# EINE MATHEMATISCHE SEMANTIK FÜR DIE BESCHREIBUNG VON DATENSTRUKTUREN

H.-D. EHRICH

## 1. Einleitung

Semantik wird hier verstanden im Sinne der Zurückführung auf - als bekannt angesehene - Konstrukte, wie z.B. Wiener Objekte in der Wiener Definitionssprache VDL [4,6]. Von einer mathematischen Semantik spricht man, wenn diese Konstrukte mathematische Objekte sind, wie z.B. Funktionen in der Fixpunkt-Theorie von Programmen [5,7].

In einer einleitenden Übersicht soll in dieser Arbeit eine Klasse mathematischer Objekte diskutiert werden, die für die semantische Spezifikation rekursiver Datentypen verwendet werden kann. In die Lehrbuchliteratur haben vor allem zwei mathematische Konstrukte Eingang gefunden, in denen sich relevante Eigenschaften strukturierter Daten wiederspiegeln, nämlich Graphen und Relationen (vgl. z.B. [6,8]). Beide Konzepte erlauben in ihrem jeweiligen Anwendungsbereich anschauliche und einleuchtende Beschreibungen von Datenstrukturen, wobei die Übersichtlichkeit im einen Falle durch bildhafte Diagramme, im anderen Falle durch systematische Tabellen erreicht wird.

Der große Nachteil dieser konventionellen Beschreibungsmittel ist, daß sie rein deskriptiven Charakter haben, d.h. nur eine explizite Darstellung gestatten:

Ein Graph wird durch seine Knoten, seine Kanten, etwaige Markierungen etc. festgelegt, eine Relation durch ihre Attribute, ihre Wertebereiche, funktionale Abhängigkeiten der Attribute, Schlüsselattribute etc.. Die Beschreibungsmethodik enthält jedoch keine Operationen auf den Konstrukten, mit deren Hilfe implizite Beschreibungen erklärbar wären, etwa folgender Art:

Gemeint ist die (unbekannte) Struktur a, die durch die Operation(en) f nicht verändert wird (die also die Gleichung u=f(u) löst).

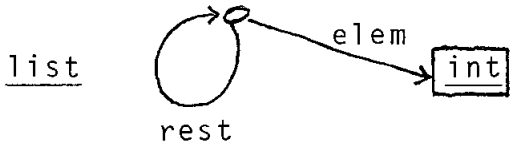
Gerade solche rekursiven Spezifikationsmethoden spielen jedoch bei der Beschreibung allgemeiner komplexer Strukturen eine große Rolle, meist in der Form eines Reference-Konzepts (wie in ALGOL 68, SIMULA, etc).

Betrachten wir ein Beispiel in Anlehnung an [3,8]:

type list = struct (list rest, int elem)

beschreibt einen rekursiven Datentyp, der intuitiv durch folgenden

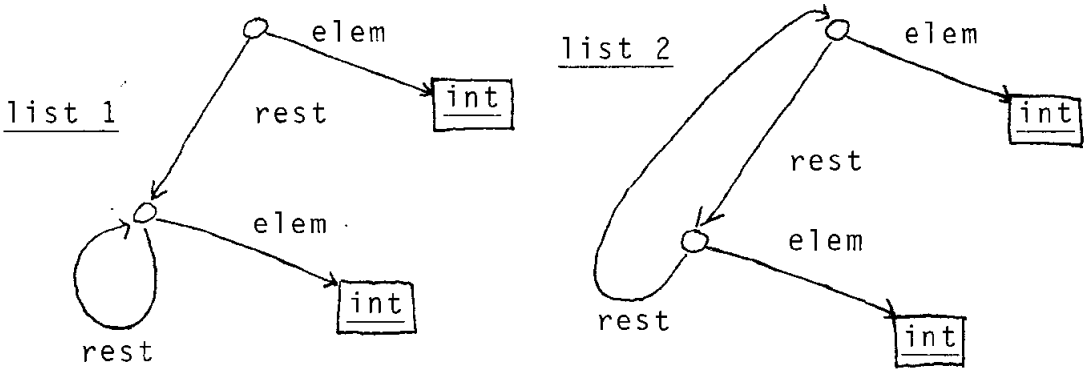
Graphen dargestellt werden kann:




Auf dieser intuitiven Basis würde man die nachfolgenden Typenvereinbarungen durch die zugeordneten Graphen erklären wollen:

```
type list1  = struct (llist1 rest, int elem)
type llist1 = struct (llist1 rest, int elem)

type list2  = struct (llist2 rest, int elem)
type llist2 = struct (list2  rest, int elem)
```




Die drei Graphen sind verschieden, jedoch sind die drei Modi äquivalent: sie beschreiben in jedem Fall die Menge der Listen, deren Elemente vom elementaren Typ int sind, und auf deren i-tes Element man durch eine Selektorfolge der Art

$$\underbrace{\text{rest.rest. ... . rest}}_{\text{i-mal}}\text{.elem}$$

zugreifen kann.

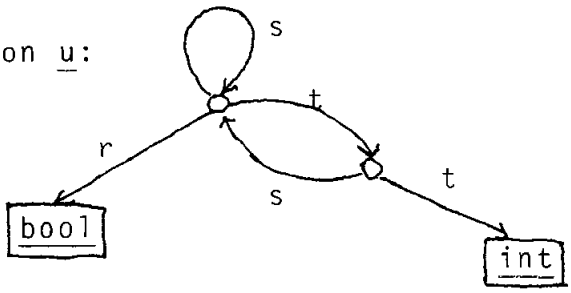
Dies Beispiel zeigt, daß die intuitive Zuordnung von Graphen zu Vereinbarungs-Texten i.a. nicht ausreicht, die Semantik befriedigend zu erklären. Wir erkennen auch, daß sich das Problem der Semantik von Datenstruktur-Beschreibungen in zwei Teilprobleme gliedert:

1. Welcher Datentyp wird durch den vorliegenden Beschreibungstext spezifiziert?
2. Welches ist genau die Menge der "zulässigen" Exemplare, die zu diesem Typ gehören?

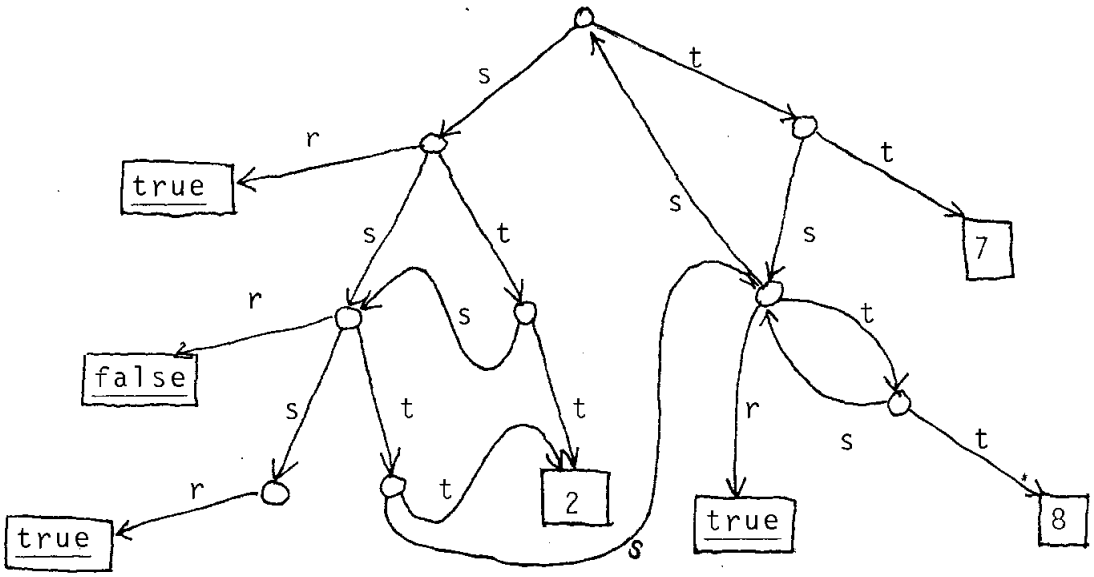
Zur Illustration der zweiten Frage betrachte man den Datentyp u, gegeben durch

type u = struct (bool r, u s, v t)
type v = struct (u s, int t)

Graph von u:




Gehört das folgende Exemplar zu diesem Typ?




In den folgenden Abschnitten werden mathematische Objekte sowie grundlegende Operationen auf ihnen definiert, die einerseits anschaulich als Graphen der obigen Art aufgefasst werden können, andererseits aber auch einer algebraischen Behandlung zugänglich sind, so daß insbesondere die Lösbarkeit von Gleichungssystemen untersucht werden kann. Die Theorie dieser Objekte wird dann zur Beantwortung der aufgeworfenen Fragen herangezogen.

## 2. Strukturierte Objekte

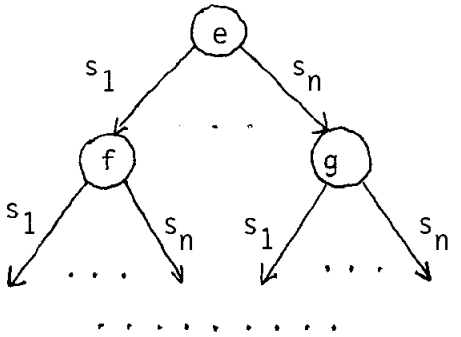
Wir setzen voraus, daß eine Menge E von Elementardaten und eine endliche Menge $S = \{s_1,\ldots,s_n\}$ von Selektoren gegeben sind. E enthalte ein spezielles Element 0, das Nulldatum. Die Menge der Worte über S sei mit $S^*$ bezeichnet, das leere Wort mit 1, und $S^*-\{1\}$ mit $S^+$.

Definition 2.1: Die Menge der (strukturierten) Objekte ist

$$\mathbb{D} := E^{S^*}$$

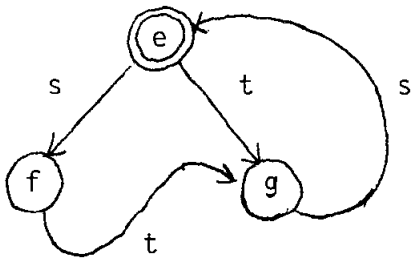
Ein Objekt $a \in \mathbb{D}$ ist also eine Abbildung $a: S^* \to E$, d.h. jeder Selektorfolge ("Zugriffsweg") wird ein Elementardatum zugeordnet. Das Nulldatum darf dabei als "nicht spezifiziert" oder "irrelevant" interpretiert werden.

Objekte werden graphisch dargestellt durch (unendliche) Wurzelbäume mit Knotenmarkierungen aus E und Kantenmarkierungen aus S, wobei von jedem Knoten für jedes $s \in S$ genau eine mit s markierte Kante ausgeht. Der Funktionswert von $x = s^1 s^2 \ldots s^m \in S^*$ ist die Markierung des Knotens, zu dem man gelangt, wenn man von der Wurzel in dieser Reihenfolge nacheinander die mit $s^1, s^2, \ldots, s^m$ markierten Kanten verfolgt.




Um in speziellen Fällen zu endlichen Darstellungen zu gelangen, treffen wir folgende Vereinbarungen:

1. Unterbäume, deren sämtliche Knoten mit 0 markiert sind, werden weggelassen.
2. Periodisch wiederkehrende Unterbäume werden durch Schleifen dargestellt.




So stellt z.B. der nebenstehende Graph mit angezeichnetem (Wurzel-) Knoten eindeutig ein Objekt a dar, für das gilt: a(1)=e, a(s)=f, a(st)= a(t)=g, a(ststssts)=a(ts) =a(1)=...=e, a(ss)=a(tt)=... =0, etc.

Definition 2.2: Ein Objekt $a \in \mathbb{D}$ heißt elementar genau dann, wenn für jedes nichtleere Selektorwort $x \in S^+$ gilt: a(x)=0. Das elementare Objekt mit a(1)=0 heißt Nullobjekt.

Das Nullobjekt wird somit durch den leeren Graphen dargestellt, die übrigen elementaren Objekte durch einzelne markierte Knoten. Die Menge der elementaren Objekte wird mit $\mathbb{E}$ bezeichnet, und wir werden die Bezeichnung $\mathbb{A} := \mathbb{D} - \mathbb{E}$ für die Menge der nicht-elementaren (oder "zusammengesetzten") Objekte verwenden. Es gibt eine offensichtliche 1-1-Zuordnung zwischen $\mathbb{E}$ und E, und wir werden im folgenden elementare Objekte und Elementardaten in der Bezeichnung

nicht unterscheiden.

Definition 2.3: Die charakteristische Menge eines Objektes $a \in \mathbf{D}$ ist

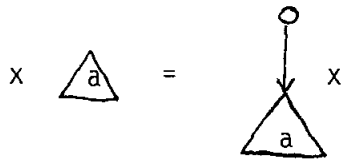
$$\chi(a) := \{x \in X^* \mid a(x) \neq 0\}$$

Offenbar ist ein Objekt a genau dann elementar, wenn $\chi(a) \subset \{1\}$ ist, und es ist das Nullobjekt genau dann, wenn $\chi(a) = \emptyset$ ist.

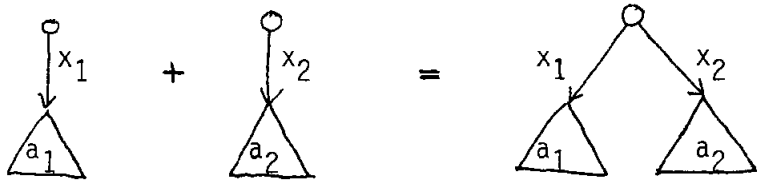
## 3. Grundoperationen

Auf der Menge **D** der strukturierten Objekte führen wir drei Grundoperationen mit der folgenden intuitiven Bedeutung ein:

1. Konstruktion: Jedem Selektorwort x(Name) und jedem Objekt a wird ein neues ("benanntes") Objekt xa zugeordnet. Dies modelliert die Benennung bzw. Bezeichnung von Objekten:

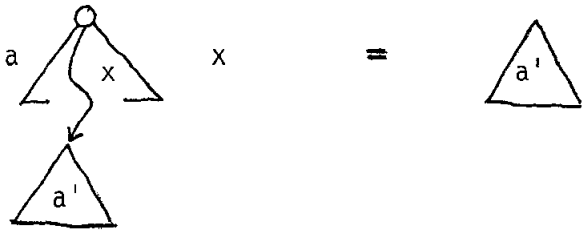



2. Addition: Die Idee ist, verschieden benannte Objekte zu einem strukturierten Objekt zusammenzufassen:




Um jedoch den algebraischen Vorteil einer totalen, d.h. für alle Objekte definierten Verknüpfungen zu haben, wird diese entsprechend erweitert.

3. Selektion: Jedem Objekt a und jedem Selektwort x wird das "mit x benannte Unterobjekt von a" zugeordnet.




Zur formalen Definition setzen wir voraus, daß eine zweistellige Verknüpfung + auf E gegeben ist, so daß (E,+,0) ein Monoid ist. (Diese Annahme verhindert keine Anwendungen, bei denen diese

Voraussetzung nicht erfüllt ist: man kann sich die Dinge dann so vorstellen, daß lediglich kein Gebrauch gemacht wird von einer an sich vorhandenen, wenn auch unbekannten Verknüpfung + auf E.)

Definition 3.1: $\forall a,b \in \mathbb{D}\ \forall x,y,z \in S^*$:

1. Konstruktion $[xa](y) := \begin{cases} a(z), & \text{falls } y=xz \\ 0 & \text{sonst} \end{cases}$

2. Addition $[a+b](y) := a(y) + b(y)$

3. Selektion $[ax](y) := a(xy)$

Der folgende Satz listet eine Anzahl einfacher, grundlegender Eigenschaften dieser Verknüpfungen auf. Hier bedeutet $x \not\leq y$, daß weder x ein Präfix von y ist noch umgekehrt für $x,y \in S^*$; für eine Menge $X \subset S^*$ und ein $x \in S^*$ sei

$$\partial_x(X) := \{y \mid xy \in X\}$$

die Ableitung von X nach x.

Satz 3.2: $\forall a,b \in \mathbb{D}\ \forall x,y \in S^*$:

1. $a1 = a$
2. $(ax)y = a(xy)$
3. $1a = a$
4. $x(ya) = (xy)a$
5. $(a+b)x = ax + bx$
6. $x(a+b) = xa + xb$
7. $x0 = 0$
8. $xa = xb \Rightarrow a = b$
9. $x \neq 1 \wedge a \neq 0 \Rightarrow xa \in \mathbb{A}$
10. $\chi(a) \cap \chi(b) = \emptyset \Rightarrow a+b = b+a$
11. $\chi(a+b) \subset \chi(a) \cup \chi(b)$
12. $\chi(ax) = \partial_x(\chi(a))$
13. $\chi(xa) = x\chi(a)$
14. $(xa)\,x = a$
15. $x \not\leq y \Rightarrow (xa)y = 0$
16. $a = a(1) + \sum_{s \in S} s(as)$
17. $a = \sum_{x \in \chi(a)} xa(x)$

Diese "Rechenregeln" für Objekte ermöglichen die Beschreibung von Objekten durch algebraische Ausdrücke und ihre Manipulation durch entsprechende algebraische Umformungen. Die Anwendung auf die Semantik von Datenbeschreibungen sei an folgendem Beispiel erläutert. Der Typenvereinbarung

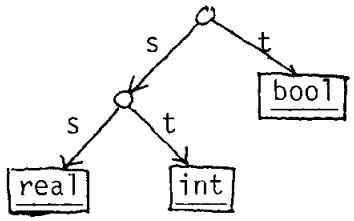
type u = struct (v s, bool t)
type v = struct (real s, int t)

ordnen wir die folgenden Gleichungen zu

u = s v + t bool
v = s real + t int

Die Rolle der Elementardaten spielen hier die elementaren Typen bool, int, real, ... . Die Gleichungen lassen sich nach u auflösen, wenn man die zweite in die erste einsetzt:

u = s(s real + t int) + t bool
= ss real + st int + t bool




Beim nebenstehenden Graphen der Lösung sind die inneren Knoten mit dem Elementardatum 0 markiert.

Gleichungssysteme der obigen Art, die sich durch einfaches Einsetzen lösen lassen, ergeben als Lösung gerade die endlichen Objekte, d.h. Objekte mit endlicher charakteristischer Menge. Endliche Objekte sind genau die, die sich durch endliche Bäume darstellen lassen.

Rekursive Typenvereinbarungen ergeben Gleichungssysteme, die sich nicht mehr durch einfaches Einsetzen lösen lassen. Wir betrachten daher im nächsten Abschnitt einen allgemeineren Typ von Gleichungssystemen.

## 4. Rationale Gleichungssysteme

Die allgemeine Theorie zur Auflösung von Fixpunktgleichungen der Art u = f(u) führt über den hier gesteckten Rahmen hinaus; es sei auf [2] verwiesen. Wir beschränken uns hier auf Gleichungssysteme der folgenden Art.

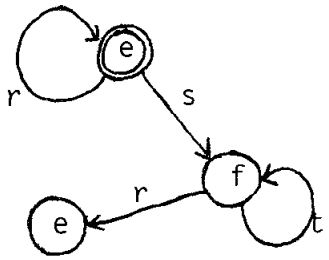
Definition 4.1: Sei $U=\{u_1,\ldots,u_m\}$ eine endliche geordnete Menge (von "Unbekannten"), $U\cap E=\emptyset$, und sei $V=U\cup E$. Ein rationales Gleichungssystem (rGls) ist ein Gleichungssystem der Form

$$u_i=\sum_{k=1}^{n} s_k v_{ik} + e_i \quad , \quad i=1,\ldots,m \ ,$$

wobei $u_i\in U$, $v_{ik}\in V$ und $e_i\in E$ ist für alle $i=1,\ldots,m$ und alle $k=1,\ldots,n$.

Ein Objekt $a_1$ ist eine Lösung dieses rGls, falls es Objekte $a_2,\ldots,a_m$ gibt, so daß alle Gleichungen erfüllt sind, wenn man $a_i$ für $u_i$ einsetzt für $i=1,\ldots,m$.

Beispiel: $u = ru + sv + e$
$v = re + tv + f$




ist ein rGls. Die graphische Lösung ist nebenstehend angegeben.

Definition 4.2: Ein Objekt $a\in D$ heißt rational genau dann, wenn die Menge $\{ax \mid x\in S^*\}$ endlich ist.

Rationale Objekte sind, anschaulich gesehen, gerade diejenigen, die auf jedem Weg von der Wurzel irgendwann "periodisch werden". Sie sind daher durch endliche Graphen darstellbar. Man zeigt leicht, daß die charakteristischen Mengen rationaler Objekte immer reguläre Mengen sind. Die rationalen Objekte entsprechen genau den Lösungen der rGls'e. Bevor wir dies Ergebnis im Satz 4.4. präzise formulieren, sei ein Lemma angeführt, welches die Grundlage gibt für die Auflösung von rGls'en.

Ist $X\subset S^*$ eine sog. "unabhängige" Menge, d.h. ist $x\neq y$ für je zwei verschiedene $x,y\in X$, so benutzen wir die folgende übersichtlichere Schreibweise:

$$Xa := \sum_{x\in X} xa$$

Lemma 4.3: Sei $R\subset S^*$ eine reguläre unabhängige Menge, $R\neq\{1\}$. Sei $b\in D$ so, daß $RS^*\cap\chi(b)=\emptyset$. Dann hat die Gleichung $u=Ru + b$ die eindeutige Lösung $a = R^*b$. Diese Lösung ist rational, sofern $b$ rational ist.

Auf den Beweis soll hier verzichtet werden (s. [2]). Wir benutzen dies Ergebnis zur schrittweisen Auflösung von rGls'en "von unten nach oben". Dies Verfahren sei am obigen Beispiel demonstriert:

$$v = re + tv + f = tv + (re+f)$$

Die Lösung ist

$$a' = t^* (re+f) = t^* re + t^* f$$

Eingesetzt in die erste Gleichung ergibt dies

$$u = ru + s (t^* re + t^* f) + e$$

Die Lösung ist

$$a = r^* (st^* r \vee 1) e + r^* st^* f$$

Allgemein ist die Lösung eines rGls von der Form

$$a = \sum_{j=1}^{p} R_j e_j \quad ,$$

wobei $e_1,\ldots,e_p$ die im rGls vorkommenden Elementardaten sind. Die $R_i$ sind reguläre Mengen.

Satz 4.4: Jedes rGls hat eine eindeutige rationale Lösung. Umgekehrt ist jedes rationale Objekt Lösung eines rGls's.

Bzgl. des Beweises sei wiederum auf [2] verwiesen. In der Anwendung auf die Semantik von Datenbeschreibungen besagt dieser Satz zweierlei:

1. Sofern nur der Zusammenhang zwischen dem Vereinbarungstext und dem rGls eindeutig hergestellt werden kann, ist die Eindeutigkeit der vereinbarten Struktur gesichert.

2. Die Beschreibung durch rGls ist hinreichend allgemein insofern, als jedes rationale - d.h. jedes durch einen endlichen Graphen darstellbare - Objekt beschrieben werden kann.

## 5. Typen und Exemplare

Es bleibt die zweite der eingangs aufgeworfenen Fragen zu klären: welches ist die Menge der "zulässigen" Exemplare, die zu einem vereinbarten Strukturtyp gehören?

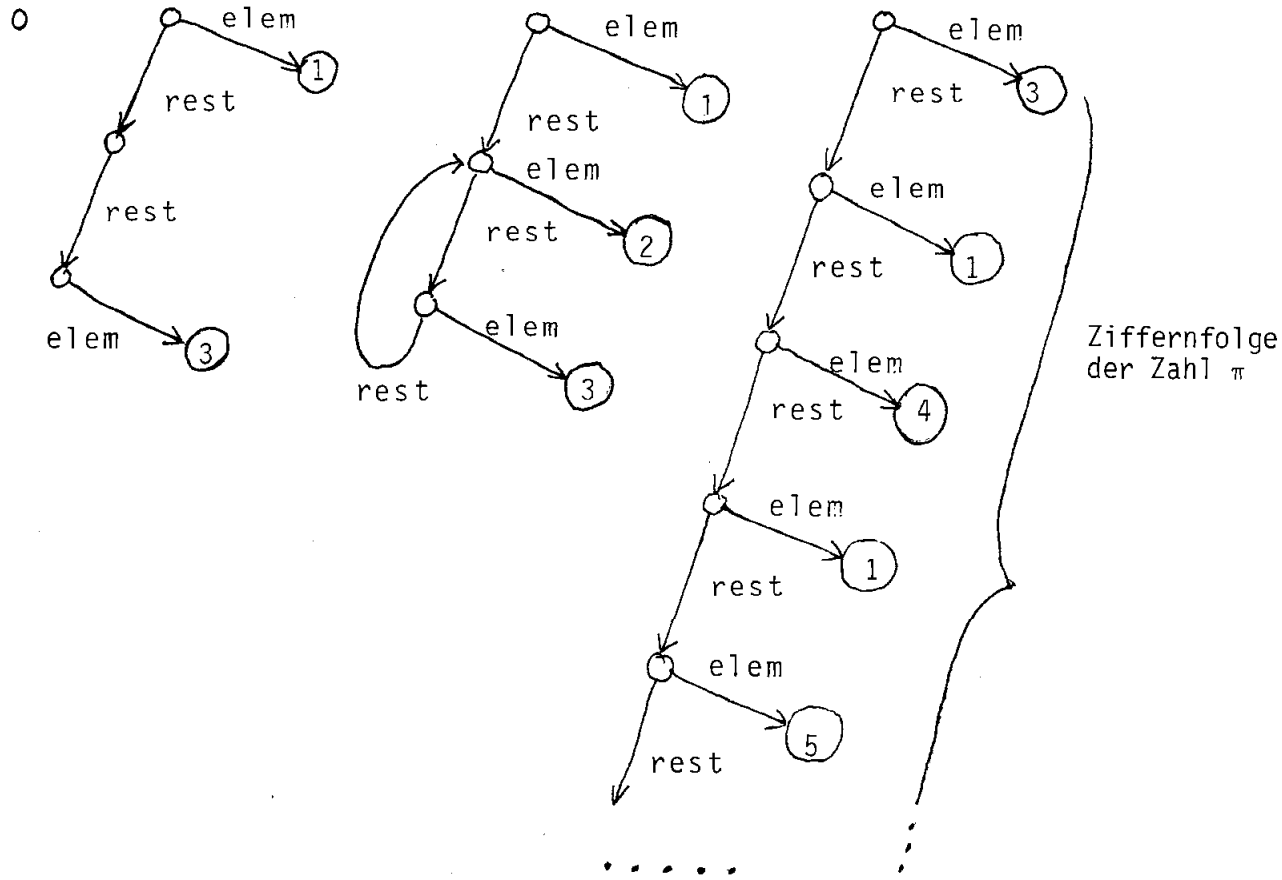
Wir setzen voraus, daß strukturierte Datentypen durch rGls'e beschrieben werden, bei denen die Elementardaten die elementaren Datentypen wie bool, int, etc. sind. Wir setzen ferner voraus, daß unser Problem für elementare Datentypen gelöst ist, daß man also weiß, welche Menge elementarer Datenexemplare durch die elementaren Datentypen gegeben sind.

Sei demnach E die Menge der Elementardaten (elementaren Exemplare), und sei $\mathcal{E} = \{\{0\}, E_1, \ldots, E_g\}$ eine Familie von Teilmengen von E, so daß $\bigcup E_j = E$ und $E_i \cap E_j = \{0\}$ ist für alle $1 \leq i,j \leq g$ (elementare Typen). Nach Satz 3.2 (17) läßt sich jeder strukturierte Typ T darstellen durch eine Summe.

$$(*) \qquad T = \sum_{x \in S^*} x E^x \quad , \; E^x \in \mathcal{E} \quad \text{für alle } x \in S^* .$$

Definiert man nun für jedes Selektorwort $x \in S^*$ und jede Teilmenge $F \subset E$ das Komplexprodukt $xF := \{xf \mid f \in F\}$ und für je zwei Teilmengen $A,B \subset \mathbb{D}$ die Summe $A+B := \{a+b \mid a \in A, b \in B\}$ so stellt jeder Typ durch den Ausdruck (*) eine Menge von Exemplaren dar, nämlich - grob gesagt - alle diejenigen Objekte, die unter jedem Zugriffsweg ein "passendes" Elementardatum haben. Man kann folgendes zeigen [2]: sind die $E^x$ Untermonoide (bzgl. +) von E, so sind die Typen T der Form (*) Untermonoide (bzgl.+) von $\mathbb{D}$, und jedes Untermonoid von $\mathbb{D}$ ist enthalten in einem Untermonoid der Form (*) von $\mathbb{D}$ für geeignete Untermonoide $E^x$ von E.

Beispiel: Exemplare des eingangs betrachteten Typs list sind:



Die ersten beiden Exemplare sind endliche, die ersten drei sind rationale Objekte; das letzte Objekt ist nicht rational. Es ist ohne Schwierigkeiten möglich, den Typ-Begriff je nach Zweckmäßigkeit auf die rationalen oder die endlichen Exemplare einzuschränken. Für solche Typen gelten sinngemäß die obigen Aussagen über Untermonoid-Beziehungen, da die Menge der endlichen sowie die der rationalen Objekte gegen Addition (und auch gegen Konstruktion und Selektion mit beliebigen Selektworten) abgeschlossen sind. Darüberhinaus läßt sich leicht zeigen, daß für rationale Typen und rationale Exemplare die Inklusions-Relation entscheidbar ist.

So können wir z.B. nachprüfen, daß das in der Einleitung gegebene (rationale) Exemplar in der Tat vom dort angegebenem Typ ist.

Endliche Objekte lassen sich durch endliche Ausdrücke mit Hilfe der Elementarobjekte und der drei Grundverknüpfungen darstellen. Dies kann man ausnutzen, um die endlichen Exemplare eines rationalen Typs durch einen grammatikalischen Erzeugungsprozesses zu beschreiben. Wir beschränken uns hier auf die Angabe von Beispielen, aus denen das generelle Prinzip leicht ablesbar ist:

Der Datentyp

$$\underline{\text{list}} = \underline{\text{rest}}\ \underline{\text{list}} + \underline{\text{elem}}\ \underline{\text{int}}$$

hat genau die Menge endlicher Exemplare, die durch die folgendermaßen generierten Ausdrücke beschrieben werden:

$$\langle \text{LIST} \rangle ::= \underline{\text{rest}}\ \langle \text{LIST} \rangle + \underline{\text{elem}}\ \langle \text{INT} \rangle \mid 0,$$

sofern aus <INT> alle int-Werte sowie das Nullobjekt generiert werden.

Ebenso werden alle endlichen Exemplare des Typs

$$u = ru + sv + e$$
$$v = re + tv + f$$

beschrieben durch die Menge der folgendermaßen generierten Ausdrücke:

$$\langle U \rangle ::= r \langle U \rangle + s \langle V \rangle + e \mid 0$$
$$\langle V \rangle ::= re + t \langle V \rangle + f \mid 0$$

## 6. Schlußbemerkungen

In einer einführenden Übersicht haben wir eine Klasse mathematischer Objekte vorgestellt, die für die Spezifikation der Semantik rekursiver Datentypen brauchbar ist. Auf dieser Grundlage läßt sich präzise klären (i) welcher Datentyp durch eine Vereinbarung spezifiziert wird und (ii) welches die Menge der dadurch charakterisierten Exemplare ist.

## L i t e r a t u r


1. Ehrich, H.-D.: Ein axiomatischer Ansatz für eine Algebra strukturierter Objekte.
   Bericht Nr. 5/74, Abteilung Informatik, Universität Dortmund, 1974

2. Ehrich, H.-D.: Outline of an Algebraic Theory of Structured Objects.
   Wird veröffentlicht.

3. Hoare, C.A.R.: Recursive Data Structures.
   Report Stan-CS-73-4oo, Standford Artificial Intelligence Laboratory, Oct. 1973

4. Lucas, P.; Lauer, P.; Stigleitner, H.: Method and Notation for the Formal Definition of Programming Languages.
   Tech. Report TR 25.o87, IBM Lab. Vienna, 1968, rev. 197o

5. Manna, Z.: Mathematical Theory of Computation.
   McGraw-Hill, New York, 1974

6. Mühlbacher, J.: Datenstrukturen
   Carl Hanser Verlag, München, Wien, 1975

7. Scott, D.; Strachey, C.: Toward a Mathematical Semantics for Computer Languages.
   In: Proc. Symp.Computers and Automata, Microwave Res.Inst.Symp.Series Vol.XXI, Polytechnic Press, Brooklyn, N.Y. 1971, pp. 19-46

8. Wedekind, H.: Datenbanksysteme I.
   Reihe Informatik Band 16, Bibliogr. Institut, Mannheim 1974.

9. van Wijngaarden, A. et al: Revised Report on the Algorithmic Language ALGOL 68. Acta Informatica 5 (1975), pp. 1-236.