Siegfried Wendt
Universität Kaiserslautern

Datentyp – Wertebereich oder Komponententyp

1. Der Datenbegriff

Umgangssprachlich wurde das Wort *Daten* schon gebraucht, lange bevor der Computer erfunden wurde und die Abkürzung EDV für "Elektronische Datenverarbeitung" in die Alltagssprache gelangte. So sagte beispielsweise der Steuerberater zu seinem Klienten: "Bevor ich Ihre Steuererklärung fertig machen kann, brauche ich von Ihnen noch ein paar Daten." Oder der Straßenbaureferent einer Stadt schrieb an den Oberbürgermeister: "Für die Entscheidung, welche der beiden in Frage stehenden Straßen vorrangig ausgebaut werden soll, müssen wir noch eine Datenerhebung durchführen." Bei diesen Daten ging es zwar oft um Zahlen – Geldbeträge, Anzahl der Kinder, Anzahl der Beschäftigungsmonate, gezählte Autos –, aber eine Gleichsetzung von Daten mit Zahlen wäre falsch. Zum einen wären Zahlen ohne mitgelieferte Wörter wie Monatseinkommen, Kinderzahl u.ä. für den Steuerberater nutzlos, zum anderen will das Finanzamt u.a. auch den Arbeitgeber des Steuerpflichtigen wissen, und dazu muß eine Adresse angegeben werden, aber keine Zahl.

Diese Vorüberlegungen legen den Schluß nahe, daß man immer dann von Daten spricht, wenn es sich um Informationen handelt, von denen man sich vorstellen kann, daß man sie in Felder eines Formulars einträgt.

> *Daten* sind diskrete Informationen oder – mit anderen Worten – informationelle Individuen.

Wegen ihrer Diskretheit sind Daten durch endliche Symbolstrukturen darstellbar. Die Diskretheit schließt jedoch nicht aus, daß Kontinuumselemente als Daten vorkommen können. So ist beispielsweise die Zahl π ein Individuum aus dem eindimensionalen Kontinuum, das durch ein einziges Symbol dargestellt werden kann. Die Unmöglichkeit, diese Zahl als endliche Dezimalziffernfolge darzustellen, ist kein Argument gegen ihre Diskretheit.

Die jeweiligen Symbolstrukturen, die zur Darstellung von Daten gewählt werden, sind selbst nicht Teil der Daten, d.h. Daten sind nur die Informationen, die man durch Interpretation der Symbolstrukturen gewinnt. In diesem Sinne sind die Symbolstrukturen *dreizehn*, *thirteen*, *13* und *XIII* vier Alternativen zur Darstellung ein und desselben Datenindividuums, welches in diesem Falle eine bestimmte natürliche Zahl ist.

2. Datentyp als Wertebereich

Da das Wort *Daten* – bzw. die in anderen Sprachen diesem Wort entsprechenden Wörter – weltweit in einheitlicher Bedeutung verwendet wird, könnte man annehmen, daß es auch für das Wort *Datentyp* eine einheitliche und aus seinen Bestandteilen *Daten* und *Typ* ableitbare Bedeutung gibt.

> Naheliegende, aber nicht immer zutreffende Definition:
>
> Ein *Datentyp* ist ein "auf einen Typ verdichtbares" Datenrepertoire.

Unter der Verdichtbarkeit einer Menge auf einen Typ ist die Möglichkeit zu verstehen, die Menge auf einfachere Weise als durch Aufzählung ihrer Elemente mitzuteilen. Beispiele hierfür sind:

– { wahr, falsch } Typ: Binärmenge der Wahrheitswerte

– { Karo, Herz, Pik, Kreuz } Typ: Kartenfarben im Skatspiel

– { 2, 4, 6, 8, 10, 12, 14, 16, ... } Typ: Menge der geraden natürlichen Zahlen

– { x | x ist ein syntaktisch korrektes PASCAL–Programm }

In den letzten beiden Beispielen handelt es sich um unendliche Mengen, die auf keinen Fall durch Aufzählung ihrer Elemente spezifiziert werden könnten. In diesen Fällen ist die Verdichtbarkeit auf einen Typ eine notwendige Voraussetzung für die Spezifizierbarkeit, d.h. hier ist die Spezifikation selbst zwangsläufig eine Verdichtung auf einen Typ.

Bei endlichen Mengen ist die Verdichtbarkeit auf einen Typ keine formal feststellbare Eigenschaft, sondern gehört in den Bereich der intuitiven Semantik. So kann beispielsweise der Autor die Menge { 1618, 1648 } auf den Typ "Jahresgrenzen des dreißigjährigen Krieges" verdichten, wogegen er für die Menge { 1610, 1747 } keine Verdichtungsmöglichkeit sieht.

Wenn man unter dem Wort *Datentyp* bestimmte Arten von Mengen versteht, dann verwendet man es synonym zum Wort *Wertebereich*.

> Eine Menge nennt man *Wertebereich*, wenn man betonen will, daß sie die Menge derjenigen Elemente ist, die als Belegung bestimmter Variablen vorkommen können oder zugelassen sind.

Wenn es sich dabei um Variable im mathematischen Sinne handelt, stehen sie für Argumente oder Ergebnisse von Funktionen. So ist beispielsweise der Wertebereich der Argumentvariablen der reellen Wurzelfunktion gleich der Menge der nichtnegativen reellen Zahlen.

Wenn es sich um Variable im Sinne der Programmierung handelt, stehen sie für Speicher oder Kanäle in Systemen, und die Elemente des Wertebereichs sind die alternativen Beobachtungsergebnisse, die zu unterschiedlichen Zeitpunkten durch Hinschauen in den Speicher oder auf den Kanal gewonnen werden können.

Man denke beispielsweise an ein Formular oder eine Karteikarte, wo das Feld "Geburtsdatum" vorkommt. Dieses Feld ist ein Beobachtungsort vom Speichertyp, und die dort zugelassenen Eintragungen sind eindeutige Datumsangaben. Ein Beispiel für einen Beobachtungsort vom Kanaltyp ist eine Schreibmaschinen– oder Computertastatur. Da zu einem Zeitpunkt immer nur höchstens eine Taste gedrückt sein darf, enthält die Menge der möglichen beobachtbaren Werte alle Tasten und noch ein weiteres Element, welches den Sachverhalt ausdrückt, daß keine Taste gedrückt ist.

Wenn man im Zusammenhang mit Software den Begriff *Datentyp* im Sinne von *Wertebereich* gebraucht, hat man selbstverständlich keine Beobachtungsorte vor Augen, die unmittelbar der menschlichen Wahrnehmung zugänglich sind. Vielmehr denkt man an Speicherplätze und Kanäle, die sich irgendwo innerhalb des Computersystems befinden und auf die man "mit dem geistigen Auge" schaut.

Man kann zwischen elementaren und strukturierten Wertebereichen unterscheiden. Bei einem Element aus einem elementaren Wertebereich macht es keinen Sinn zu fragen, aus welchen anderen Individuen dieses Element aufgebaut sei. Typische Beispiele für elementare Wertebereiche sind die Zahlenwertebereiche oder der binäre Wertebereich der Boolschen Wahrheitswerte.

Die Bezeichnung *strukturierter Wertebereich* soll darauf hinweisen, daß es sich um eine Menge von Strukturen handelt, d.h. daß jedes Element dieser Menge als Tupel aus Mengen und Relationen angesehen werden kann. Typische Beispiele für strukturierte Wertebereiche sind Mengen von Folgen oder Mengen von Mengen. So ist beispielsweise der Eingangswertebereich des Compilers eine Menge von Folgen von Schriftzeichen und Zwischenräumen. Ein weiteres Beispiel für einen strukturierten Wertebereich ist die Menge der Ableitungsbäume zu syntaktisch korrekten Programmen einer bestimmten Programiersprache.

3. Datentyp als Komponententyp

Wenn der Begriff Datentyp in der Bedeutung von Komponententyp verwendet wird, geht es um Komponenten, die als Automaten mit einem Wertebereich repX für die Eingabe, einem Wertebereich repZ für den Zustand und einem Wertebereich repY für die Ausgabe modelliert werden können. Zum Automatenmodell gehören neben diesen drei Wertebereichen noch die Zustandsübergangsfunktion δ und die Mealy–Ausgabefunktion $\omega[X(n), Z(n)]$ bzw. die Moore–Ausgabefunktion $\sigma[Z(n)]$.

Eine Sonderstellung nimmt der Komponententyp *Speicher* ein. Bei diesem Komponententyp lassen sich die beiden Wertebereiche repX und repY aus dem Zustandswertebereich repZ ableiten, wenn man weiß, um welche Speicherart es sich handelt (s. Bild 1). Es gibt zwei Arten von Speichern, nämlich zustandsanzeigende Speicher und zustandsmeldende Speicher. Einen zustandsanzeigenden Speicher kann man sich als eine Tafel vorstellen, deren Beschriftung jeder sehen kann, der hinschaut. Das Hinschauen ist kein Eingabevorgang bezüglich des Speichers. Anders liegt der Fall beim zustandsmeldenden Speicher. Einen solchen kann man sich vorstellen als Apparat, der seinen Inhalt akustisch mitteilt, wenn er zu einer solchen Mitteilung durch einen Eingabevorgang aufgefordert wird.

Anzeigender Speicher (Moore–Maschine)	Meldender Speicher (Mealy–Maschine)
$repX = repZ$ $repY = repZ$	$repX = \{ \text{Anfrage} \} \cup repZ$ $repY = repZ \cup \{ ok \}$
$Y(n) = Z(n)$	$Y(n) = \begin{cases} Z(n) & \text{falls } X(n) = \text{Anfrage} \\ ok & \text{sonst} \end{cases}$
$Z(n+1) = X(n)$	$Z(n+1) = \begin{cases} Z(n) & \text{falls } X(n) = \text{Anfrage} \\ X(n) & \text{sonst} \end{cases}$

Bild 1 Automatenmodelle für alternative Speicherarten

Die Speichervariablen, die in Programmiersprachen vereinbart werden, beispielsweise

```
j:INTEGER
```

stellt man sich zweckmäßigerweise als zustandsanzeigende Speicher vor. Die Bezeichner für solche Speicher können links oder rechts vom Zuweisungsoperator vorkommen, beispielsweise

```
j:=j+5
```

Man kann sich nun vorstellen, daß wer immer diese Anweisung ausführt, die aktuelle Anzeige

$$Y(n) = Z(n) = j(n)$$

zur Kenntnis nimmt, 5 hinzuaddiert und das Ergebnis als Eingabe X(n) dem Speicher übergibt, so daß dieser dann anschließend den neuen Wert

$$Z(n+1) = j(n+1) = X(n) = j(n) + 5$$

gespeichert hat.

Einen zustandsanzeigenden Speicher kann man sich auch als einfaches getaktetes Register vorstellen, wie es links in Bild 2 gezeigt ist.

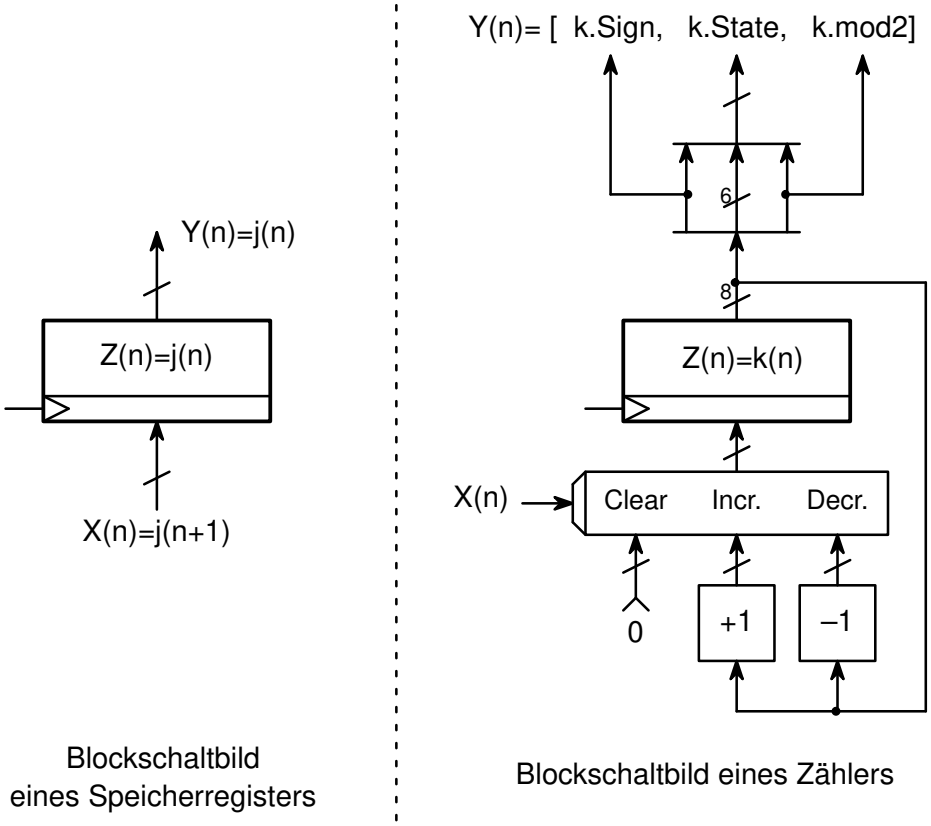

Bild 2 Schaltwerke als Beispiele automatenmodellierbarer Komponenten

Rechts in diesem Bild ist ein etwas komplexeres Schaltwerk gezeigt, nämlich ein Zähler mit dem Eingaberepertoire { `Clear`, `Increment`, `Decrement` }. Die gewählte Bündelbreite von 8 Bit legt den Zustandswertebereich dieses Zählers fest auf das Zahlenintervall $-64 \leq k \leq +63$. Es wird in diesem Fall nicht nur der Zustand k angezeigt, sondern auch die beiden aus dem Zustand ableitbaren Binär-

werte Vorzeichen(k) und $k_{mod\,2}$. In einem Programm könnte eine Komponente von diesem Typ eingeführt werden durch die Deklaration

```
k:Zähler
```

Anweisungen, die auf diese Komponente Bezug nehmen, könnten dann beispielsweise wie folgt aussehen:

```
k.Clear

k.Increment

j:=k.State

IF k.mod2 EQUAL 0 THEN ...
```

Es wäre zwar keine gute Entscheidung bei der Gestaltung der Programmiersprache, aber es wäre denkbar, daß das sprachliche Gebilde

```
k:Increment
```

nicht nur eine Eingabe symbolisiert, die den Zähler veranlaßt, seinen aktuellen Zählerstand um Eins zu erhöhen, sondern daß dieses sprachliche Gebilde auch noch die Meldung des Zählerstands nach der Erhöhung impliziert, so daß man schreiben könnte

```
j:=k.Increment
```

Es wäre in diesem Fall besser, wenn die Programmiersprache es erlauben würde zu schreiben

```
j:=k.Increment.State
```

Man sollte einem sprachlichen Gebilde immer ansehen, ob es nur eine Eingabe, nur eine Ausgabe oder beides symbolisiert.

Es soll an dieser Stelle darauf hingewiesen werden, daß das Schaltbild des Zählers mehr Information liefert als eine bloße Zählerspezifikation. Das Bild sagt nämlich auch aus, daß der jeweilige Zählerstand als Binärwort codiert ist, bei dem das linke Bit als Vorzeichen und das rechte Bit als Modulo–2–Wert des Zählerstandes interpretiert werden können. Eine reine Spezifikation würde lediglich aussagen, daß aus dem Zählerstand das Vorzeichen und der Modulo–2–Wert ableitbar sind, aber in der Spezifikation würde nichts darüber gesagt, wie diese Ableitung zu geschehen habe.

Die Wörter *Speicher* oder *Zähler* lassen kaum eine andere Interpretation zu als daß es sich dabei um Systemkomponenten handelt. Niemand würde wohl auf die Idee kommen, diese Wörter zur Bezeichnung von Wertebereichen zu verwenden. Es gibt aber durchaus Bezeichnungen, die sowohl auf Wertebereiche als auch auf Systemkomponenten passen. Ein typisches Beispiel für eine solche Bezeichnung ist das Wort *Liste*. Ein mit dem Wort *Liste* bezeichneter Wertebereich ist die unendliche Menge aller endlichen Folgen, deren Elemente aus einem vorgegebenen Repertoire stammen. So ist beispielsweise der Wertebereich mit der Bezeichnung `LIST_OF_INTEGER` die unendliche Menge aller endlichen Folgen von Zahlen aus dem INTEGER–Bereich, und der mit `LIST_OF_BOOLEAN` bezeichnete Wertebereich ist die unendliche Menge aller endlichen Binärwörter.

Man kann aber unter dem Wort *Liste* auch eine Systemkomponente bestimmten Typs verstehen, zu der beispielsweise die im folgenden angegebenen Repertoires gehören, aus denen man leicht auf die nicht angegebenen Funktionen ω und δ schließen kann:

$$\text{repZ} = \text{repElement}^0 \cup \text{repElement}^1 \cup \text{repElement}^2 \cup \text{repElement}^3 \cup \ldots$$

$$\begin{aligned}\text{repX} = {} & \{ \text{Clear, Length, Read_First, Read_Last, Delete_First, Delete_Last} \} \cup \\ & \cup \{ \text{Read_Position} \} \times \text{INTEGER} \cup \\ & \cup \{ \text{Delete_Position} \} \times \text{INTEGER} \cup \\ & \cup \{ \text{Add_as_First} \} \times \text{repElement} \cup \\ & \cup \{ \text{Add_as_Last} \} \times \text{repElement} \cup \\ & \cup \{ \text{Insert_as_Position} \} \times \text{INTEGER} \times \text{repElement}\end{aligned}$$

$$\text{repY} = \{ \text{ok, illegal_input} \} \cup \text{INTEGER} \cup \text{repElement}$$

So wie das Wort *Liste* wahlweise als Bezeichnung eines Wertebereichs oder als Bezeichnung einer Systemkomponente verwendet werden kann, kann beispielsweise die Bezeichnung *Delete_First* wahlweise als Bezeichner für eine Funktion oder eine Operation interpretiert werden. Zwar legt die Bezeichnung *Delete_First* eher die Interpretation als Operation nahe, denn man verbindet damit die Vorstellung eines Geschehens, dennoch könnte damit die Funktion gemeint sein, die ihrem Argument, das eine nichtleere Liste sein muß, als Ergebnis die um das erste Element verkürzte Argumentliste zuordnet. Bei Funktionen gibt es kein Geschehen, sondern Funktionen beschreiben immer nur einen statischen Zusammenhang zwischen Argumenten und Ergebnissen. Das Geschehen der Funktionsberechnung gehört nicht zum Funktionsbegriff.

Die dargestellten Überlegungen sollten den Leser davon überzeugen, daß es für die verständliche Kommunikation unerläßlich ist, in allen Fällen, wo eine Verwechslung möglich ist, explizit darauf hinzuweisen, ob man von einem Wertebereich oder von einer Systemkomponente bzw. ob man von einer Funktion oder von einer Operation spricht.