Unterscheidung zweier Programmzwecke

Der Nutzen eines Programms ergibt sich bei seiner Ausführung; dann erlebt der Nutzer das Verhalten des programmierten Systems. Dabei ist der Programmzweck entweder ergebnisorientiert oder prozessorientiert.

Ergebnisorientierung liegt vor, wenn es demjenigen, der einen Nutzen von dem System haben will, genügt, eine Anfangssituation und eine Endsituation zu betrachten. In der Anfangssituation liefert der Nutzer das nötige Material - im Falle der Informationsverarbeitung also die nötigen Daten - und erteilt dem System den Auftrag zur Schaffung des Ergebnisses. In der Endsituation übergibt ihm das System das Ergebnis.

Ergebnisorientierung kann man sich veranschaulichen, indem man an gegenständliche Ergebnisse denkt. Das Ergebnis könnte beispielsweise ein Schrank sein, den ein Schreiner für einen Auftraggeber baut. In der Anfangssituation liefert der Auftraggeber dem Schreiner das erforderliche Holz, und in der Endsituation nimmt er den Schrank entgegen. Im Falle der Ergebnisorientierung hat der Nutzer keinerlei Interesse, den Prozess der Ergebnisherstellung zu beobachten.
Bei der Informationsverarbeitung liegt Ergebnisorientierung immer dann vor, wenn aus den in der Anfangssituation bereitgestellten Informationen durch Verknüpfung ein Ergebnis gewonnen wird. Ergebnisorientierung äußert sich in diesem Falle immer als Anwendung einer Funktion auf ein gegebenes Argument – also beispielsweise als Berechnung des Wurzelwertes 5 zum Argument 25.

Prozessorientierung liegt vor, wenn der Nutzen des Systems im Prozess selbst liegt. Aus einem solchen Prozess kann man natürlich nur einen Nutzen ziehen, indem man den Prozess wahrnimmt oder sogar aktiv daran teilnimmt.
Ein anschauliches Beispiel für Prozessorientierung ist ein Konzert, aus dem man keinen Nutzen ziehen kann, wenn man nicht im Saal anwesend ist und zuhört. Im Gegensatz dazu braucht derjenige, der einen Schrank in Auftrag gegeben hat, nicht in der Schreinerwerkstatt anwesend zu sein, um einen Nutzen aus dem Schrank zu ziehen.

Dem Paar Ergebnisorientierung/Prozessorientierung entspricht in einer Analogie das Paar Warenproduktion/Dienstleistung. Eine Ware kann man herstellen lassen und später abholen, eine Dienstleistung aber nicht. Der Dienst wird an dem Nutzer des Dienstes geleistet: Der Kranke wird gepflegt, der Theaterbesucher wird unterhalten, der Schüler wird unterrichtet.

Computerspiele sind typische Beispiele für Prozessorientierung.

Innerhalb eines prozessorientierten Programmes wird es im Normalfall immer Abschnitte geben, die ergebnisorientiert sind. Denn es werden im Prozess immer wieder Schritte auszuführen sein, bei denen Funktionen auf Argumente angewendet werden.

Alternative Möglichkeiten der Programmformulierung

Obwohl Programme in künstlichen Sprachen formuliert werden, können für ihre Interpretation dennoch keine anderen Kategorien gelten als diejenigen, die man aus dem Bereich der na-

türlichen Sprachen kennt. Im Bereich der natürlichen Sprachen unterscheidet man zwischen Ausdrücken und ganzen Sätzen, und bei den Sätzen unterscheidet man zwischen Aussagen, Fragen und Anweisungen. Dementsprechend kann ein Programm

– *imperativ*, d.h. als Anweisung
– *deklarativ*, d.h. als Paar aus Aussage und Frage
– *funktional*, d.h. als Ausdruck

formuliert werden. Für jede dieser Programmformen ist festgelegt, worin die Programmausführung bestehen soll:

– *Ein imperatives Programm* ist als Anweisung formuliert und wird ausgeführt, indem die Anweisung ausgeführt wird. Diese Programmkategorie stand offensichtlich Pate bei der Wortwahl ”Programmausführung”.

– *Ein deklaratives Programm* ist als Paar aus Aussage und Frage formuliert und wird ausge–führt, indem die Frage beantwortet wird. Dabei können nur solche Fragen beantwortet wer–den, deren Antwort aus der zuvor gemachten Aussage folgt.

– *Ein funktionales Programm* ist als Ausdruck formuliert, der eine Funktion und ein zugehöriges Argument umschreibt. Ein solches Programm wird ausgeführt, indem das Ergebnis der umschriebenen Funktion für das ebenfalls in der Umschreibung angegebene Argumenttupel berechnet wird.

Im folgenden werden die drei Sprachkategorien für Programme näher betrachtet.

Imperative Programme

Anweisungen verlangen Operationen. Eine *Operation* ist eine Zustandsveränderung. Deshalb kann man mit jeder Operation ein Zustandspaar verbinden, welches aus dem Zustand vor der Operation – z.B. der Blinddarm ist noch drin – und dem Zustand nach der Operation – der Blinddarm ist raus – besteht. Es ist üblich, diese beiden Zustände durch die Bezeichnungen PRE und POST zu unterscheiden.
Man kann zwischen elementaren und zusammengesetzten Anweisungen unterscheiden. In einer zusammengesetzten Anweisung wird die verlangte Operation dadurch umschrieben, dass angegeben wird, wie sich die große Operation aus kleineren Operationen zusammensetzt.

Deklarative Programme

Das Programm besteht aus einem Paar aus Aussage und Frage. Die Aussage – es wird sich im allgemeinen um eine zusammengesetzte Aussage handeln, d.h. um eine Aussage, die als Verknüpfung elementarer Aussagen formuliert ist – wird üblicherweise als *Wissensbasis* bezeichnet, und anstelle des Wortes ’Frage’ wird üblicherweise das Wort *Anfrage* benutzt. Wenn man sagt, daß eine Wissensbasis aus *Wissenselementen* bestehe, meint man, dass die Wissensbasis eine Aussage ist, die durch konjunktive Verknüpfung aus einfacheren Aussagen gewonnen wird; diese konjunktiv verknüpften Aussagen werden als Wissenselemente bezeichnet.

Zur Formulierung von Aussagen bzw. Aussageformen stehen grundsätzlich die Sprachmittel der Aussagenlogik und der Prädikatenlogik zur Verfügung.

Eine Aussage, die ohne Quantifikation und damit variablenfrei formuliert ist, bezeichnet man als *elementares Faktum*. Durch elementare Fakten werden entweder faktisch wahre Sachverhalte ausgedrückt oder vorgegebene Axiome. Auch Axiome können als Sachverhalte angesehen werden, die aber nicht die beobachtbare Welt betreffen, sondern formal konstruierte abstrakte Welten. Mit einem Faktum verbindet man keine Erkenntnis, sondern nur Wissen. So wird man beispielsweise nicht von einer Erkenntnis sprechen, wenn gesagt wird, dass Johann Wolfgang v. Goethe im Jahre 1832 gestorben sei, oder dass1 eine natürliche Zahl ist.

Eine Aussage, die mit Quantifikatoren und damit mit Individuenvariablen formuliert ist, bezeichnet man als *Regel*. Mit dem Begriff Regel verbindet man intuitiv die Vorstellung einer allgemeingültigen Erkenntnis, und in der Formulierung erkennt man dies am Vorkommen eines Quantifikators. Eine Regel kann eine logische Regel oder eine faktische Regel sein. Eine *logische Regel* ist aus sprachlichen Gründen wahr, wogegen sich die Wahrheit einer *faktischen Regel* auf Sachverhalte der äußeren Welt gründet. Dass sich die Erde alle 24 Stunden einmal um ihre Achse dreht oder dass alle Kinder der Frau Anna männlich sind, sind ”regelhafte Fakten”, d.h. Fakten, die unter Verwendung von Quantifikatoren formuliert werden.
In *logischen Regeln* wird nicht auf irgendetwas Beobachtbares Bezug genommen. Wenn beispielsweise gesagt wird, daß jede Elternteil–Kind–Beziehung, in der beide Beteiligten männlich sind, eine Vater–Sohn–Beziehung sei, so ist dies keine Erkenntnis aus der Beobachtung, sondern lediglich eine Konsequenz der Bedeutung der Wörter ’Elternteil’, ’Kind’, ’männlich’, ’Vater’ und ’Sohn’.

Eine Frage ist entweder eine *Behauptungsfrage* oder eine *Selektionsfrage*.
Eine *Behauptungsfrage* hat die umgangssprachliche Form: ”Ist die Behauptung *xxx* wahr ?”
Im Programm schreibt man dies in der Kurzform ”*xxx* ?”, denn die umgebenden Wörter kann man sich dazudenken.
Selektionsfragen in umgangssprachlicher Form beginnen i.a. mit einem Fragewort: ”Wer hat ... ?” oder ”Wann wurde .... ?” oder ”Wo steht ... ?”. Diese Formen kann man verallgemeinern zu der Form ”Welche Individuentupel machen die Aussageform *P*(*Tupel der Argumentvariablen*) wahr ?”, und dies wiederum kann man durch Weglassen der implizit assoziierbaren Wörter verkürzen auf ” *P*(*Tupel der Argumentvariablen*) ? ”.

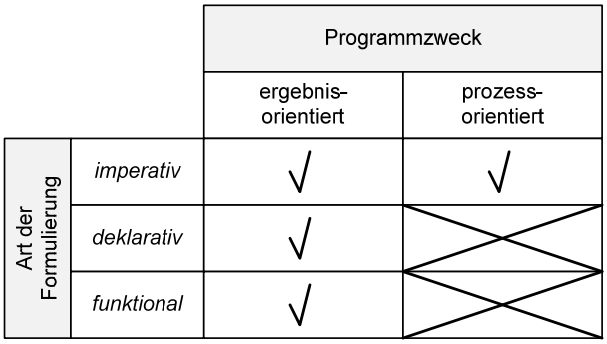
Funktionale Programme

Ein funktionales Programm ist eine Umschreibung und besteht im Normalfall aus zwei Teilen – einem Teil, der eine Funktion f(...) festlegt, und einem anderen Teil, der das Argumenttupel festlegt. Im Sonderfall der nullstelligen Funktionen besteht das Programm nur aus einem Teil, nämlich der Angabe der Funktion, die eine Konstante darstellt.

Die Klassifikation funktionaler Umschreibungen in primitiv funktionale und komponiert funktionale Umschreibungen ist auf die funktionalen Programme zu übertragen. In komponiert funktionalen Programmen kommen zwangsläufig definierende Funktionsumschreibungen vor. Für die Formulierung definierender Funktionsumschreibungen hat man eine große Vielfalt von Möglichkeiten. So ist es auch möglich, in definierenden Funktionsumschreibungen Anweisungen zu verwenden. Jede imperative Programmiersprache bietet die Möglichkeit, Funktionen unter Verwendung von Anweisungen zu formulieren.
In *rein funktionalen Programmen* ist jedoch die Möglichkeit ausgeschlossen, definierende Funktionsumschreibungen mit Hilfe von Anweisungen zu formulieren.

Vergleichende Betrachtungen

Das sich aufgrund der beiden eingeführten Klassifikationskriterien ergebende Klassifikationsschema für Programme lässt sich in Form der unten stehenden Matrix darstellen.



In dieser Matrix sind zwei Felder ausgestrichen, denn die Formulierung von prozessorientierten Programmen erfordert die Verwendung von Anweisungen, in denen Operationen verlangt werden, und Anweisungen stehen in rein funktionalen oder rein deklarativen Sprachen nicht zur Verfügung.

In Sprachen, in denen keine Anweisungen zur Verfügung stehen, kann auch der Begriff der Speicherzelle nicht relevant sein. Speicherzellen braucht man als Orte, an denen die Zustände PRE und POST beobachtet werden. In diesem Sinne ist auch das Papier, welches bei der Ausführung einer Druckanweisung seinen Zustand ändert, eine Speicherzelle. Die Relevanz des Begriffs der Speicherzelle ist ein Merkmal imperativer Sprachen.
Praktisch gibt es bei allen Aufgaben, die durch Programmierung zu lösen sind, einen Bedarf an identifizierbaren Speicherzellen und an der Möglichkeit, explizit zu verlangen, dass bestimmte End– oder Zwischenergebnisse in solchen Speicherzellen abgelegt werden. Das Denken in PRE und POST ist sehr natürlich, und schon bei einfachsten Aufgaben ist dieses Denken unerlässlich – man denke an den Wunsch, berechnete Ergebnisse ausdrucken zu lassen.

Die imperativen Sprachen sind also die universellen Sprachen, und in diese Sprachen lassen sich funktionale oder deklarative Anteile ohne ”Verrat an den Konzepten” einbringen. Denn rechts vom Zuweisungsoperator könnte grundsätzlich ein beliebiges ergebnisorientiertes Programm stehen:
Speichervariable := Ergebnisorientiertes Programm

Bei den bekannten imperativen Sprachen sind diese rechts vom Zuweisungsoperator zugelassenen ergebnisorientierten Programme auf primitiv funktionale Programme beschränkt, z.B.
v := SQRT (ABS (SIN ((x+0.338) / 2.0)))
Solche Einschränkungen können aber lediglich mit praktischen Erwägungen begründet werden, d.h. es gibt dafür keine logisch zwingenden Gründe.

Im Gegensatz hierzu ist die Formulierung prozessorientierter Programme mit rein funktionalen und deklarativen Sprachen grundsätzlich unmöglich. Anstatt nun aber, was nahe läge, die imperativen Sprachen funktional und deklarativ anzureichern, hat man alle funktionalen und deklarativen Sprachen imperativ angereichert und damit eine kategorielle Verwirrung gestiftet. So bleibt beispielsweise die sprachliche Form PRINT (Ergebnis) eine Anweisung, auch wenn man sie als Funktion oder als Prädikat bezeichnet – so wie ein Löwe ein Tier bleibt, auch wenn man ihn ”Hustensaft” nennt.

Sowohl einem imperativen Programm als auch einem funktionalen Programm kann der Programmausführende explizit entnehmen, was er zu tun hat. Zwar kommen nur in imperativen Programmen explizite Anweisungen vor, aber auch dem funktionalen Programm kann der Ausführende mühelos entnehmen, welche einzelnen Schritte er ausführen muss. Er muss nach der selbstverständlichen Regel verfahren, dass zuerst die Argumentbelegungen bestimmt werden müssen, bevor eine Funktionsauswertung erfolgen kann. Deshalb muss er mit der Berechnung der inneren Funktionen beginnen und sich sukzessive nach außen vorarbeiten, bis er zur Auswertung der äußersten Funktion kommt.

Im Gegensatz hierzu erhält derjenige, der ein deklaratives Programm ausführen soll, aus dem Programm keinerlei Hinweise darauf, welche einzelnen Schritte er nebenläufig oder nacheinander ausführen soll. Die Konstruktion einer Maschine, die deklarative Programme ausführen kann, liegt somit sehr viel weniger auf der Hand als die Konstruktion von Maschinen zur Ausführung imperativer oder funktionaler Programme. Man kann ein deklaratives Programm als ”ein Gleichungssystem mit Unbekannten” ansehen und die Programmausführung als Auflösung dieses Gleichungssystems. Einem Gleichungssystem sieht man i.a. überhaupt nicht an, was man tun muss, um es nach den Unbekannten aufzulösen.

Einordnung der Objekt–Orientierung

Möglicherweise haben Sie schon von objektorientierten Programmen gehört. Da liegt die Frage nahe, ob denn die objektorientierten Programme nicht außerhalb der drei vorgestellten Sprachkategorien liegen und man deshalb die obige Tabelle in Bild 5 um eine vierte Zeile erweitern müsse. Die Antwort auf diese Frage lautet: Die objektorientierten Programme gehören zur Kategorie der imperativen Programme, denn es handelt sich auch hier um Programme, die als Anweisungen formuliert sind. Das Merkmal *objektorientiert* bezieht sich auf die Art und Weise, wie die Anweisungen formuliert werden – genauer gesagt, wie in den Anweisungen die Operatoren und Operanden identifiziert werden. Die Objektorientierung ist eine Alternative zu einer anderen imperativen Formulierungsart, die man *operator–orientiert* nennen könnte.