Eigenschaften von Algorithmen

- Zur Erinnerung: wichtige Eigenschaften, die Algorithmen haben können/sollten, sind:
  1. Präzise, endliche Beschreibung.
  2. Effektive Verarbeitungsschritte.
  3. Elementare Verarbeitungsschritte.
  4. Ein Algorithmus heißt *terminierend*, wenn er bei jeder Anwendung nach endlich vielen Verarbeitungsschritten zum Ende kommt.
  5. Ein Algorithmus heißt *deterministisch*, wenn die Wirkung und die Reihenfolge der Einzelschritte eindeutig festgelegt ist, andernfalls *nicht-deterministisch*.
  6. Ein Algorithmus heißt *determiniert*, wenn das Ergebnis der Verarbeitung für jede einzelne Anwendung eindeutig bestimmt ist, andernfalls *nicht-determiniert*.

Eigenschaften von Algorithmen

Desweiteren ist natürlich die möglicherweise wichtigste Eigenschaft eines Algorithmus die *Korrektheit*, d.h. informell, „der Algorithmus tut, was er tun soll“.

1. Ein Algorithmus heißt *partiell korrekt*, wenn für alle gültigen Eingaben das Resultat der Spezifikation des Algorithmus entspricht.
2. Ein Algorithmus heißt *(total) korrekt*, wenn der Algorithmus partiell korrekt ist, und für alle gültigen Eingaben terminiert.

Wir werden uns später noch genauer mit der Frage beschäftigen, wie man sicher sein kann, dass ein Algorithmus partiell/total korrekt ist.

Running Example

- Wir wollen diese Eigenschaften anhand eines Beispiels diskutieren.
- Dazu betrachten wir folgende Aufgabe:

  Ein Kunde kauft Waren für $1 \leq r \leq 100$ EUR und bezahlt mit einem 100 EUR Schein (r sei ein voller EUR Betrag ohne Cent-Anteil). Gesucht ist ein Algorithmus, der zum Rechnungsbetrag r das Wechselgeld w bestimmt. Zur Vereinfachung nehmen wir an, dass w nur aus 1 EUR oder 2 EUR Münzen oder 5 EUR Scheinen bestehen soll. Es sollen möglichst wenige Münzen/Scheine ausgegeben werden (also ein 5 EUR Schein statt fünf 1 EUR Münzen).

Datendarstellung

- Zunächst müssen wir zur Lösung dieser Aufgabe die Darstellung (Modellierung) der relevanten Daten festlegen.
- Für den Rechnungsbetrag r ist dies trivial, denn offensichtlich ist $r \in \mathbb{N}$. Wir nehmen an, dass r in Dezimaldarstellung gegeben ist.
- Das Wechselgeld w kann auf verschiedene Weise modelliert werden, z.B. als Folge oder Multimenge von Wechselgeldmünzen. Ein aus zwei 1-EUR-Münzen, einer 2-EUR-Münze und zwei 5-EUR-Scheinen bestehendes Wechselgeld könnte als Folge $(1, 1, 2, 5, 5)$ dargestellt sein.

Datendarstellung

- Wir legen also folgende Datendarstellung fest:
  - r: als natürliche Zahl in Dezimaldarstellung.
  - w: als Folge von Werten 1, 2 oder 5.
- Wir benutzen die Sprechweise „nimm x zu w hinzu“ für die Konkatenation $w \circ x$.

Erster Lösungsansatz

Idee (Wechselgeld 1):

- Ausgehend vom ursprünglichen Rechnungsbetrag r zählen wir sukzessive um 1,2 und 5 hoch (unter Hinzunahme der entsprechenden Münzen/Scheine) bis wir bei 100 angelangt sind.
- Dabei möglichst wenige Münzen/Scheine ausgeben:
  - Wenn die letzte Ziffer 2,4,7 oder 9 ist, muss 1 hoch gezählt werden
  - Wenn die letzte Ziffer 1 oder 6 ist, muss 2 hoch gezählt werden
  - Wenn die letzte Ziffer 3 oder 8 ist, muss nochmals 2 hoch gezählt werden
  - Ab dann: nur noch 5er hoch zählen

Algorithmus 1 (Wechselgeld 1)

Eingabe: $r \in \mathbb{N}$

Führe folgende Schritte der Reihe nach aus:

1 Setze $w = ()$.

2 Falls die letzte Ziffer von r eine 2,4,7 oder 9 ist,
dann erhöhe r um 1 und nimm 1 zu w hinzu.

3 Falls die letzte Ziffer von r eine 1 oder 6 ist,
dann erhöhe r um 2 und nimm 2 zu w hinzu.

4 Falls die letzte Ziffer von r eine 3 oder 8 ist,
dann erhöhe r um 2 und nimm 2 zu w hinzu.

5 Solange $r < 100$: Erhöhe r um 5 und nimm 5 zu w hinzu.

Ausgabe: w

Erster Lösungsansatz

- Ablaufbeispiel: Sei $r = 81$.

	$r = 81$		(Ausgangssituation)
Schritt 1	$r = 81$	$w = ()$	
Schritt 2	(keine Änderung, da die letzte Ziffer keine 2,4,7,9 ist)		
Schritt 3	$r = 83$	$w = (2)$	
Schritt 4	$r = 85$	$w = (2, 2)$	
Schritt 5	$r = 90$	$w = (2, 2, 5)$	
	$r = 95$	$w = (2, 2, 5, 5)$	
	$r = 100$	$w = (2, 2, 5, 5, 5)$	

Erster Lösungsansatz

- Eigenschaften:
  - Endliche Aufschreibung: OK.
  - Effektive und elementare Einzelschritte: ?
  - Der Algorithmus ist terminierend, deterministisch, determiniert und partiell korrekt (und damit auch total korrekt).

Variante des ersten Lösungsansatzes

Idee: Fasse ähnliche Schritte zusammen.

Algorithmus 2 (Wechselgeld 2)

Eingabe: $r \in \mathbb{N}$

Führe folgende Schritte der Reihe nach aus:

1 Setze $w = ()$.

2 Solange $r < 100$:

Führe jeweils (wahlweise) einen der folgenden Schritte aus:

3 Falls die letzte Ziffer von r eine 2,4,7 oder 9 ist,
dann erhöhe r um 1 und nimm 1 zu w hinzu.

4 Falls die letzte Ziffer von r eine 1,2,3,6,7 oder 8 ist,
dann erhöhe r um 2 und nimm 2 zu w hinzu.

5 Falls die letzte Ziffer von r eine 0,1,2,3,4 oder 5 ist,
dann erhöhe r um 5 und nimm 5 zu w hinzu.

Ausgabe: w

Variante des ersten Lösungsansatzes

- Ablaufbeispiel:

	$r = 81$		(Ausgangssituation)
Schritt 1	$r = 81$	$w = ()$	
Schritt 2b	$r = 83$	$w = (2)$	
Schritt 2b	$r = 85$	$w = (2, 2)$	
Schritt 2c	$r = 90$	$w = (2, 2, 5)$	
Schritt 2c	$r = 95$	$w = (2, 2, 5, 5)$	
Schritt 2c	$r = 100$	$w = (2, 2, 5, 5, 5)$	

Variante des ersten Lösungsansatzes

- Alternativer Ablauf:

	$r = 81$		(Ausgangssituation)
Schritt 1	$r = 81$	$w = ()$	
Schritt 2b	$r = 83$	$w = (2)$	
Schritt 2c	$r = 88$	$w = (2, 5)$	
Schritt 2b	$r = 90$	$w = (2, 5, 2)$	
Schritt 2c	$r = 95$	$w = (2, 5, 2, 5)$	
Schritt 2c	$r = 100$	$w = (2, 5, 2, 5, 5)$	

Variante des ersten Lösungsansatzes

- Eigenschaften:
  - Die Variante (Algorithmus 2) arbeitet nach dem selben Grundprinzip wie der urspr. Algorithmus (Algorithmus 1), lässt aber gewisse “Freiheiten” bei der Auswahl des nächsten Schrittes in Schritt 2 zu.
    Daher: Algorithmus 2 ist nicht-deterministisch.
  - Algorithmus 2 ist nicht determiniert (siehe alternativer Ablauf): Bei ein und derselben Anwendung erzeugt der Algorithmus zwei unterschiedliche Folgen $w = (2, 2, 5, 5, 5)$ und $w = (2, 5, 2, 5, 5)$.

Variante des ersten Lösungsansatzes

- Bemerkung 1: wenn wir statt Folgen Multimengen (bei denen die Reihenfolge der Elemente keine Rolle spielt) bei der Darstellung von w verwendet hätten, wären die Ergebnisse $w = \{2, 2, 5, 5, 5\}$ und $w = \{2, 5, 2, 5, 5\}$ gleich, d.h. in diesem Fall wäre Algorithmus 2 determiniert.
- Bemerkung 2: Würden wir r nicht in Dezimal- sondern z.B. in Binärdarstellung repräsentieren, wären beide Algorithmen in der angegebenen Form sinnlos.
- Die Wahl der Datendarstellung kann also erheblichen Einfluss auf Eigenschaften von Algorithmen haben.

Zweite Variante des ersten Lösungsansatzes

Idee:

- Wir lösen uns von der Abhängigkeit von der Datendarstellung, indem wir r nicht mehr hochzählen, sondern die Differenz $100 - r$ berechnen und diese in möglichst wenige Teile der Größen 1, 2 und 5 aufteilen.
- Wir gehen dabei davon aus, dass die Differenz "$-$" für beliebige Zifferndarstellungen (dezimal, binär, oktal, etc.) definiert ist.
- Die Zahl 100 müsste in dieselbe Darstellung gebracht werden, die auch für r benutzt wird.

Zweite Variante des ersten Lösungsansatzes

Algorithmus 3 (Wechselgeld 3)

Eingabe: $r \in \mathbb{N}$

Führe folgende Schritte der Reihe nach aus:

1 Berechne $d = 100 - r$ und setze $w = ()$.

2 Solange $d \geq 5$: Vermindere d um 5 und nimm 5 zu w hinzu.

3 Falls $d \geq 2$, dann vermindere d um 2 und nimm 2 zu w hinzu.

4 Falls $d \geq 2$, dann vermindere d um 2 und nimm 2 zu w hinzu.

5 Falls $d \geq 1$, dann vermindere d um 1 und nimm 1 zu w hinzu.

Ausgabe: w

Zweite Variante des ersten Lösungsansatzes

- Ablaufbeispiel:

	$r = 81$		(Ausgangssituation)
Schritt 1	$d = 19$	$w = ()$	
Schritt 2	$d = 14$	$w = (5)$	
	$d = 9$	$w = (5, 5)$	
	$d = 4$	$w = (5, 5, 5)$	
Schritt 3	$d = 2$	$w = (5, 5, 5, 2)$	
Schritt 4	$d = 0$	$w = (5, 5, 5, 2, 2)$	
Schritt 5	(keine Änderung, da $d \geq 1$ nicht gilt)		

Diskussion

- Alle drei Varianten weisen eine gemeinsame “operative” Auffassung auf: Die Berechnung des Rückgelds wird durch eine Folge von “Handlungen” vollzogen.
- Diese Handlungen sind charakterisiert durch Veränderungen der Größen r bzw. d und w.
- Die Abfolge dieser “Aktionen” wird durch typische Konstruktionen gesteuert:
  - *Fallunterscheidung*: Falls . . ., dann . . .
    ermöglicht, die Ausführungen von Aktionen vom Erfülltsein gewisser Bedingungen abhängig sein zu lassen.
  - *Iteration*: Solange . . .: . . .
    steuert die Wiederholung von Aktionen in Abhängigkeit gewisser Bedingungen.

Diskussion (cont.)

- Neben der strikten operativen Auffassung gibt es eine Sichtweise von Algorithmen, die durch eine rigorose mathematische Abstraktion gewonnen wird.
- In unserem Beispiel ist die Zuordnung eines Wechselgelds w zu einem Rechnungsbetrag r mathematisch nichts anderes als eine Abbildung

$$h : r \mapsto \text{“herauszugebendes Wechselgeld”}$$

- Die Aufgabe ist dann, eine Darstellung der Abbildung h zu finden, die “maschinell auswertbar” ist.

Diskussion (cont.)

- Eine triviale Lösung für h ist, in einer kompletten Aufstellung alle möglichen Werte für r mit zugehörigem Wechselgeld $w = h(r)$ aufzulisten.
- Dies ist allgemein aber nur für endliche Definitionsbereiche möglich und sogar nur für “kleine” Definitionsbereiche sinnvoll (für unser Problem also möglich, aber bestenfalls unpraktisch).
- Allgemein wird man eine kompakte Darstellung suchen, in der die zu bestimmende Abbildung in geeigneter Weise aus einfachen (elementar auswertbaren) Abbildungen zusammengesetzt ist (Stichwort: Komposition).

- Für eine solche Darstellung der Abbildung *h* benötigen wir zunächst zwei “Hilfsabbildungen”, die in der Informatik oft gebräuchlich sind:

$$DIV : \mathbb{N}_0 \times \mathbb{N} \to \mathbb{N}_0 \qquad \text{und} \qquad MOD : \mathbb{N}_0 \times \mathbb{N} \to \mathbb{N}_0$$

- *DIV* berechnet das Ergebnis der ganzzahlige Division zweier natürlicher Zahlen, z.B. $DIV(7, 3) = 2$.
- Der *Modulo*-Operator *MOD* berechnet den Rest der ganzzahligen Division zweier natürlicher Zahlen, z.B. $MOD(7, 3) = 1$.

- Formal: Sind $k \in \mathbb{N}_0$ und $l \in \mathbb{N}$, so kann man k durch l mit ganzzahligem Ergebnis q teilen, wobei eventuell ein Rest r übrigbleibt, d.h. es gibt $q, r \in \mathbb{N}_0$ mit

$$r < l \text{ und } k = q \cdot l + r.$$

Dann ist

$$DIV(k, l) = q \quad \text{und} \quad MOD(k, l) = r.$$

Funktionaler Lösungsansatz

Idee:

- Ähnlich wie Algorithmus 3 berechnen wir das Wechselgeld $100 - r$ und teilen dies dann durch 5.
- Der ganzzahlige Quotient $q_1 = DIV(100 - r, 5)$ ist die Anzahl der 5-EUR-Scheine im Wechselgeld $h(r)$.
- Der Rest $r_1 = MOD(100 - r, 5)$ ist der noch zu verarbeitende Wechselbetrag. Offensichtlich gilt $r_1 < 5$.
- r_1 muss nun auf 1 und 2 aufgeteilt werden, d.h. analog bilden wir $q_2 = DIV(r_1, 2)$ und $r_2 = MOD(r_1, 2)$.
- q_2 bestimmt die Anzahl der 2 EUR Münzen und r_2 die Anzahl der 1 EUR Münzen in $h(r)$.

Algorithmus 4 (Wechselgeld 4)

$$\begin{aligned} h(r) \quad = \quad & (5_1, \ldots, 5_{DIV(100-r,5)} \\ & 2_1, \ldots, 2_{DIV(MOD(100-r,5),2)} \\ & 1_1, \ldots, 1_{MOD(MOD(100-r,5),2)}) \end{aligned}$$

Bemerkungen:

- Dabei sind alle Elemente in der Ergebnisfolge durchnummeriert, um die Anzahl der entsprechenden Elemente anzudeuten, d.h. das Element 5_i bezeichnet den i-ten 5-EUR-Schein im Ergebnis.
- In dieser Schreibweise kann z.B. $(5_1, 5_0, \ldots)$ auftreten, was bedeuten soll, dass die Folge kein Element 5 enthält.

Funktionaler Lösungsansatz

- Beispielanwendung $r = 81$:

$$DIV(100 - r, 5) = DIV(19, 5) = 3$$
$$MOD(100 - r, 5) = MOD(19, 5) = 4$$
$$DIV(MOD(100 - r, 5), 2) = DIV(4, 2) = 2$$
$$MOD(MOD(100 - r, 5), 2) = MOD(4, 2) = 0$$

d.h. man erhält $h(81) = (5_1, 5_2, 5_3, 2_1, 2_2, 1_1, 1_0)$ oder, einfacher geschrieben,

$$h(81) = (5, 5, 5, 2, 2).$$

Funktionaler Lösungsansatz

- Eigenschaften:
  - Wenn man von der Bildung der Folgen aus den berechneten Anzahlen der Elemente 1, 2 und 5 absieht, sind nur die Auswertungen der Operationen „$-$“, *DIV* und *MOD* als Einzelschritte anzusehen. Setzt man diese als elementar voraus, ist die angegebene Definition von *h* eine Beschreibung des gesuchten Algorithmus.
  - Diese Darstellung nimmt keinen Bezug mehr auf eine „Abfolge von Aktionen“ (d.h. wie das Ergebnis „berechnet“ werden soll) sondern beschreibt den „funktionalen Zusammenhang“ zwischen *r* und *w* durch ineinander eingesetzte Anwendungen (Komposition) gewisser Basisoperationen (und damit wie das Ergebnis „aussehen“ soll).

Variante des funktionalen Lösungsansatzes

- Leider lässt sich in vielen Fällen die gesuchte Abbildung nicht in derartig einfacher Weise explizit angeben.
- Ein wichtiges Prinzip für den Allgemeinfall von Abbildungen auf natürlichen Zahlen (und anderen v.a. induktiv definierten Mengen) ist das Prinzip der Rekursion, das wir im vorigen Kapitel kennengelernt haben.

Variante des funktionalen Lösungsansatzes

Algorithmus 5 (Wechselgeld 5)

$$h(r) = \begin{cases} \text{falls } r = 100, \text{ dann} \quad (), & (1) \\ \text{falls } 100 - r \geq 5, \text{ dann} \quad (5) \circ h(r+5), & (2) \\ \text{falls } 5 > 100 - r \geq 2, \text{ dann} \quad (2) \circ h(r+2), & (3) \\ \text{falls } 2 > 100 - r \geq 1, \text{ dann} \quad (1) \circ h(r+1), & (4) \end{cases}$$

Hier ist der Basisfall (1) für $h(100)$ definiert und die Rekursionsfälle (2),(3),(4) werden auf die Fälle $h(r+5)$, $h(r+2)$ und $h(r+1)$ zurückgeführt.

Variante des funktionalen Lösungsansatzes

- Beispielanwendung $r = 81$:

$$
\begin{aligned}
h(81) &= (5) \circ h(86) && \text{(Fall 2)} \\
&= (5) \circ (5) \circ h(91) && \text{(Fall 2)} \\
&= (5) \circ (5) \circ (5) \circ h(96) && \text{(Fall 2)} \\
&= (5) \circ (5) \circ (5) \circ (2) \circ h(98) && \text{(Fall 3)} \\
&= (5) \circ (5) \circ (5) \circ (2) \circ (2) \circ h(100) && \text{(Fall 3)} \\
&= (5) \circ (5) \circ (5) \circ (2) \circ (2) \circ () && \text{(Fall 1)} \\
&= (5, 5, 5, 2, 2)
\end{aligned}
$$

Variante des funktionalen Lösungsansatzes

- Trotz der rein funktionalen Darstellung der Zuordnung $r \mapsto w$ erinnert die Auswertung wieder an die Abfolge der Aktionen im entsprechenden Ablaufbeispiel (ähnlich wie in Algorithmus 3).
- Tatsächlich kann das Prinzip der Rekursion auch in der operativen Auffassung der Algorithmen 1-3 eingerichtet werden und dabei einen ähnlichen Effekt wie die Iteration erzielen.

Variante des funktionalen Lösungsansatzes

- In einer rekursiven Abbildungsdefinition greift man auf Werte dieser Abbildung für kleinere (oder hier: größere) Argumente zurück.
- Analog kann in einem Algorithmus, der die Abfolge gewisser Aktionen in Abhängigkeit von Eingabewerten steuert, die Ausführung des Algorithmus selbst auch als eine derartige Aktion auftreten.

Algorithmus 6 (Wechselgeld 6)

Eingabe: $r \in \mathbb{N}$

Setze $w = ()$

Führe den in Abhängigkeit von r rekursiv definierten Algorithmus $A(r)$ aus:

$A(r)$:

Führe denjenigen der folgenden Schritte (1) - (3) aus, dessen Bedingung erfüllt ist (ist keine Bedingung erfüllt, ist die Ausführung zu Ende):

(1) Falls $100 - r \geq 5$, dann nimm 5 zu w hinzu
und führe anschließend $A(r + 5)$ aus.

(2) Falls $5 > 100 - r \geq 2$, dann nimm 2 zu w hinzu
und führe anschließend $A(r + 2)$ aus.

(3) Falls $5 > 100 - r \geq 1$, dann nimm 1 zu w hinzu
und führe anschließend $A(r + 1)$ aus.

Ausgabe: w

Algorithmische Konzepte

- In den vergangenenen Abschnitten haben wir einige grundlegende algorithmische Konzepte kennengelernt:
  - Eingabe- und Ergebnisdaten (insbesondere solche von komplexer Art) müssen geeignet dargestellt werden.
  - Die Ausführung gewisser Aktionen bzw. die Auswertung gewisser Operationen wird als in elementaren Schritten durchführbar vorausgesetzt.
  - Einzelne Schritte können zusammengesetzt werden (z.B. Nacheinanderausführung, Auswahl von Aktionen, Kompositionen von Abbildungen, etc.)
  - Fallunterscheidung, Iteration und Rekursion ermöglichen die Steuerung des durch den Algorithmus beschriebenen Verfahrens.

Algorithmische Konzepte

- Diese Konzepte sind z.T. abhängig vom verwendeten Programmierparadigma.
- Zum Abschluss dieses Kapitels diskutieren wir daher nochmal die verschiedenen Paradigmen:
  - Imperatives (prozedurales) Programmieren
  - Funktionales Programmieren
  - Objektorientiertes Programmieren
  - Logik-Programmieren

Historische Entwicklung

- Das “ursprüngliche” Paradigma
- Abgeleitet aus der Architektur der Computer (insbs. der zugrundeliegenden Befehlssteuerung: binär codierte Befehle, direkte Adressierung von Speicherzellen)
- Abstraktion von der Maschinenorientierung durch Variablenkonzept (statt binäre Speicheradressierung) und Rekursivität der Programmsteuerung (Programme können durch andere Programme verändert werden)
- Problemorientierte Programmierung

Wesentliche Charakteristika

- Programm ist Folge elementarer Anweisungen
- Diese Anweisung haben typischerweise einen Effekt auf die zu verarbeitenden Daten (z.B. werden Daten verändert)
- Programmausführung: Abarbeitung der Anweisungen
- Wichtige Konzepte: Variablen, Verzweigung, Schleifen, Prozeduren
- Vertreter: FORTRAN, ALGOL, COBOL, Pascal, BASIC, C, Modula (und z.T., als deren Weiterentwicklungen: C++ und Java))

Historische Entwicklungg

- Höhere Abstraktion: weg von der Maschinen-orientierten Problemauffassung hin zur Beschreibungsform der Problemstellung (daher auch applikative, d.h. anwendungsbezogene Programmierung)
- Eigenschaft des Problems, bzw. der Lösung, wird beschrieben (deklarativ), orientiert am Menschen statt an der Rechenanlage

Wesentliche Charakteristika

- Programm ist Funktion von Eingabe- in Ausgabewerte
- Die Funktion hat keine Effekte auf die zu verarbeitenden Daten (z.B. werden Eingabedaten nicht verändert), sie gibt lediglich die Ausgabe zurück
- Programmausführung: Berechnung der Funktion
- Wichtige Konzepte: Ausdruck, Substitution/Auswertung, λ-Kalkül
- Vertreter: LISP, SML, Haskell

Historische Entwicklung

- Bei großen SW-Systemen werden imperative/funktionale Programme schnell unübersichtlich
- Idee der abstrakten Datentypen wird zu benutzerdefinierten Datentypen erweitert
- Noch stärkere Abstraktion durch Datenkapselung

Grundidee

- Daten sind Objekte, die miteinander interagieren (sich Nachrichten schicken)
- Klassen stellen benutzerdefinierte Datentypen zur Verfügung, d.h. definieren eine Menge von Objekten mit gewissen Eigenschaften (Attribute, die den Zustand der Objekte spezifizieren) und Verhalten (Methoden, die z.B. den Zustand verändern)
- Objekte verschiedener Klassen interagieren durch gegenseitiges Aufrufen von Methoden
- Einbettung von Klassen in eine Hierarchie ermöglicht Vererbung (Wiederverwendung) von Eigenschaften

Beispiel: Wechselgeld

- In unserem Beispiel Wechselgeld (was eigentlich zu klein für eine klassische oo Programmierung ist) würde man ggfls. einen neuen Datentyp “Wechselgeld” definieren.
- Ein Objekt vom Typ “Automat”, der das Wechselgeld berechnet, könnte einem entsprechenden Wechselgel-Objekt Nachrichten schicken (eine entspr. Methode aufrufen), z.B. “erhöhe um 5 EUR”.
- Voraussetzung dafür: der Datentyp Wechselgeld bietet diese “Funktionalität” (Methode) an (andernfalls könnte er mit dieser Nachricht nichts anfangen).

Beispiel: Wechselgeld

- D.h. man müsste sich überlegen, welche Nachrichten Objekte vom Typ “Wechselgeld” verarbeiten können sollten, d.h. welche Methoden bereitgestellt werden sollen.
- Neben dem Erhöhen des Betrags, könnte das z.B. die Abfrage der Anzahl der einzelnen Münzen/Scheine sein.
- Ein Automat könnte diese Informationen benötigen, um die entsprechende Anzahl an Münzen und Scheinen auszuzahlen.

Beispiel: Wechselgeld

- Was ist der Vorteil?
  - Der Datentyp Wechselgeld kann unabhängig vom Wechselgeld-Algorithmus implementiert und verwendet werden.
  - Der Wechselgeld-Algorithmus ist damit weitgehend unabhängig von der Darstellung des Wechselgelds (Details des Datentyps “Wechselgeld”): solange die Schnittstelle klar ist (d.h. informell welche Methoden zur Verfügung stehen, wie diese aufgerufen werden können, und was das Resultat ist), muss der Automat keinerlei Details des Wechselgeld-Objekts kennen (weder wie die Methoden genau funktionieren, noch wie Wechselgeld-Objekte aufgebaut sind).

Beispiel: Wechselgeld

- Wie gesagt, in diesem Beispiel ist das noch kein großer Ah-Effekt, bei komplexen Software-Systemen mit mehreren Entwicklern ist der Effekt allerdings umso größer.
- Nochmal: Einzelne Aspekte können komplett unabhängig von einander entwickelt und (wieder-)verwendet werden.

Wesentliche Charakteristika

- Programm ist Menge von Klassen und Objekten
- Programmausführung: Interaktion zwischen Objekten
- Wichtige Konzepte: Klassen und Objekte, Datenkapselung, Vererbung, Polymorphie
- Vertreter: Java, C++, Python

Grundidee

- Ausgehend von einer Menge von Fakten (Grundtatsachen und Axiome) wird mit Hilfe festgelegter Schlussregeln eine Aussage abgeleitet (oder falsifiziert).
- Programm ist Sammlung von Fakten und Regeln
- Programmausführung: Suche nach Antworten auf Anfragen
- Wichtige Konzepte: logisches Schließen, regelbasiert, prädikativ, constraintbasiert
- Vertreter: PROLOG

Zusammenfassung

- Imperative Algorithmen:
  - spezifizieren die Abfolge von Aktionen, die typischerweise bestimmte Größen verändern.
  - spiegeln die technischen Möglichkeiten von Rechneranlagen wider, die Schritt für Schritt bestimmte grundlegende Aktionen ausführen können.
- Funktionale Algorithmen:
  - spezifizieren eine auswertbare Darstellung des funktionalen Zusammenhangs zwischen Ein- und Ergebniswerten.
  - spiegeln ein höheres Abstraktionsniveau wider: die eigentliche Spezifikation des Algorithmus als Abbildung ist praktisch losgelöst von den technischen Möglichkeiten der Rechenanlage (Aktionen Schritt für Schritt auszuführen).

Zusammenfassung

- Objektorientierte Algorithmen:
  - spezifizieren eine höhere Abstraktionsebene zur Darstellung von komplexen Eingabe- und Ergebnisdaten sowie Zwischenzuständen während der Berechnung.
  - Idee der (Daten-)Objekte, die sich gegenseitig Nachrichten schicken
  - verwenden zur eigentlichen Lösung der gegebenen Aufgabe (wie die Nachrichten verarbeitet werden) funktionale und/oder imperative Algorithmen.