Überblick

2. Daten und Algorithmen
2.1 Zeichenreihen
2.2 Datendarstellung durch Zeichenreihen
2.3 Syntaxdefinitionen
2.4 Algorithmen

Syntax

- Die Gestalt einer Datendarstellung nennt man *Syntax*.
- Die Bedeutung der dargestellten Objekte heißt *Semantik*.
- Die Syntax von Darstellungen von Daten/Objekten kann ohne Bezugnahme auf die Semantik definiert werden.

Syntaxdefinition von natürlichen Zahlen

- Die Menge der natürlichen Zahlen in Dezimaldarstellung kann durch folgende Regeln definiert werden:
  1. 0 ist eine $\langle Dezimalzahl \rangle$.
  2. Jede Ziffer $x \in \mathcal{A}_{10} \setminus \{0\}$ ist eine $\langle Nichtnulldarstellung \rangle$.
  3. Ist a eine $\langle Nichtnulldarstellung \rangle$ und $y \in \mathcal{A}_{10}$ so ist $a \circ y$ eine $\langle Nichtnulldarstellung \rangle$.
  4. Jede $\langle Nichtnulldarstellung \rangle$ ist eine $\langle Dezimalzahl \rangle$.
- Dabei bezeichnet $\circ$ die *Konkatenation* zweier Zeichenketten, z.B. ergibt die Konkatenation $123 \circ 456$ die Zeichenkette 123456.
- Beispiel: Die Zeichenreihe 308 ist eine Dezimalzahl gemäß folgender Anwendungen der Regeln:

3	ist $\langle Nichtnulldarstellung \rangle$ nach Regel 2
30	ist $\langle Nichtnulldarstellung \rangle$ nach Regel 3
308	ist $\langle Nichtnulldarstellung \rangle$ nach Regel 3
308	ist $\langle Dezimalzahl \rangle$ nach Regel 4

Syntaxdefinition von natürlichen Zahlen

- Die Begriffe $\langle Dezimalzahl \rangle$ und $\langle Nichtnulldarstellung \rangle$ in den Regeln werden *syntaktische Variablen* genannt. Sie kennzeichnen den zu definierenden Begriff sowie einen Hilfsbegriff.
- Allgemein können in Syntaxdefinitionen noch mehr syntaktische Variablen vorkommen. Wir heben sie durch kursive Schrift und die eckigen Klammern hervor.
- Die Zeichen des vorgegeben Alphabets heißen *Terminalzeichen*.

Backus-Naur-Form

- Eine formale, häufig gebrauchte Form von Syntaxdefinitionen ist die *Backus-Naur-Form (BNF)*
- Eine Syntaxdefinition in BNF is gegeben durch eine endliche Anzahl von *Syntaxregeln* der Form

$$\alpha ::= \beta$$

- Dabei ist α eine syntaktische Variable und β eine *BNF-Satzform* (siehe nächste Folie).
- Eine ausgezeichnete syntaktische Variable ist das *Startsymbol* α_S.

BNF-Satzform

- Syntaktische Variablen und Terminalzeichen sind BNF-Satzformen.
- Auswahl:
  Sind $\gamma_1, \ldots, \gamma_n$ BNF-Satzformen, dann auch $\gamma_1 | \ldots | \gamma_n$.
- Verkettung (Konkatenation):
  Sind $\gamma_1, \ldots, \gamma_n$ BNF-Satzformen, dann auch $\gamma_1 \ldots \gamma_n$.
- Klammerung:
  Ist γ eine BNF-Satzform, dann auch (γ).
- Iteration:
  Ist γ eine BNF-Satzform, dann auch $(\gamma)^*$.
- nichtleere Iteration:
  Ist γ eine BNF-Satzform, dann auch $(\gamma)^+$.
- Option:
  Ist γ eine BNF-Satzform, dann auch $[\gamma]$.

Bedeutung (Semantik) der BNF

- Eine Zeichenreihe w ist *herleitbar* mit einer BNF Syntaxdefinition, wenn man w aus α_S durch eine oder mehrere der folgenden Ersetzungsoperationen erhalten kann:
  - Eine syntaktische Variable α, für die die Regel $\alpha ::= \gamma_1 | \ldots | \gamma_n$ vorhanden ist, darf durch eines der γ_i $(1 \leq i \leq n)$ ersetzt werden.
  - Ein Vorkommen von $(\gamma_1 | \ldots | \gamma_n)$ darf man durch eines der γ_i $(1 \leq i \leq n)$ ersetzen.
  - Ein Vorkommen von $(\gamma)^*$ darf man durch $\underbrace{(\gamma)(\gamma)\ldots(\gamma)}_{n-\text{mal}}$ mit einem beliebigen $(n \geq 0)$ ersetzen.
  - Ein Vorkommen von $(\gamma)^+$ darf man durch $\underbrace{(\gamma)(\gamma)\ldots(\gamma)}_{n-\text{mal}}$ mit einem beliebigen $(n > 0)$ ersetzen.
  - Ein Vorkommen von (γ) darf man durch γ ersetzen falls γ nicht von der Form $\gamma_1 | \ldots | \gamma_n$ ist.
  - Ein Vorkommen von $[\gamma]$ darf man durch (γ) ersetzen, oder ersatzlos streichen.
- Wir schreiben auch $\alpha_S \rightarrow w$.

Beispiele

- BNF Syntaxdefinition für natürliche Zahlen:

  $\langle Dezimalzahl \rangle ::= 0 \mid \langle Nichtnulldarstellung \rangle$

  $\langle Nichtnulldarstellung \rangle ::= \langle Nichtnullziffer \rangle \ (0 \mid \langle Nichtnullziffer \rangle)^*$

  $\langle Nichtnullziffer \rangle ::= 1|2|3|4|5|6|7|8|9$

  Die syntaktische Variable $\langle Dezimalzahl \rangle$ dient als Startsymbol α_S.

- Ableitung der Zeichenreihe 308 aus $\alpha_S = \langle Dezimalzahl \rangle$ ist:

  $$\begin{aligned}\langle Dezimalzahl \rangle &\rightarrow \langle Nichtnulldarstellung \rangle \\ &\rightarrow \langle Nichtnullziffer \rangle \ (0 \mid \langle Nichtnullziffer \rangle)^* \\ &\rightarrow \langle Nichtnullziffer \rangle \ (0 \mid \langle Nichtnullziffer \rangle)(0 \mid \langle Nichtnullziffer \rangle) \\ &\rightarrow \langle Nichtnullziffer \rangle \ 0 \ \langle Nichtnullziffer \rangle \\ &\rightarrow 308\end{aligned}$$

Beispiele

- BNF Syntaxdefinition für ganze Zahlen ("$-$" und die Dezimalziffern des Alphabets $\mathcal{A}_{10}$ sind Terminalzeichen):

  $\langle GanzeZahl \rangle ::= [-] \langle Dezimalzahl \rangle$

  Die syntaktische Variable $\langle GanzeZahl \rangle$ dient hier als Startsymbol α_S.

- BNF Syntaxdefinition für Gleitpunktzahlen ("$-$", "E" und die Dezimalziffern des Alphabets $\mathcal{A}_{10}$ sind Terminalzeichen):

  $\langle Gleitpunktzahl \rangle ::= [-] \langle Mantisse \rangle (E \langle GanzeZahl \rangle)$

  $\langle Mantisse \rangle ::= \langle Dezimalzahl \rangle . \langle Dezimalstellen \rangle$

  $\langle Dezimalstellen \rangle ::= 0 | \langle Nichtnullstellen \rangle$

  $\langle Nichtnullstellen \rangle ::= (0 | \langle Nichtnullziffer \rangle)^* \langle Nichtnullziffer \rangle$

  Die syntaktische Variable $\langle Gleitpunktzahl \rangle$ dient hier als Startsymbol α_S.

Diskussion

- Mit Hilfe der BNF kann man nicht nur die Syntax (Darstellung) von Daten formal definieren.
- Viele Programmiersprachen benützen BNF Syntaxdefinitionen auch für die eindeutige Definition ihrer Syntax, d.h. der Spracheelemente, die erlaubt sind und der Sprache an sich.
- Manche Eigenschaften von Darstellungen oder Sprachen kann man in BNF allerdings nicht darstellen. Diese werden dann typischerweise zusätzlich "verbal" angegeben, d.h. sie werden als Zusatzanforderungen notiert. Diese zusätzlichen Angaben heißen *Kontextbedingungen*.

Überblick

2. Daten und Algorithmen
2.1 Zeichenreihen
2.2 Datendarstellung durch Zeichenreihen
2.3 Syntaxdefinitionen
2.4 Algorithmen

Eigenschaften von Algorithmen

- Nach dem wir die erste der auf Folie 23 genannten Eigenschaften, eindeutige Datendarstellung, diskutiert haben, widmen wir uns im folgenden den restlichen drei Eigenschaften von Algorithmen.
- Neben diesen Forderungen gibt es noch weitere interessante Eigenschaften, die Algorithmen haben können.
- Wir werden all diese Eigenschaften an einem Beispiel erläutern.

Eigenschaften von Algorithmen

Die restlichen drei Eigenschaften von Algorithmen auf Folie 23 sind

1. Präzise, endliche Beschreibung.
2. Effektiver Verarbeitungsschritt.
3. Elementarer Verarbeitungsschritt.

Dazu kommen folgende Eigenschaften:

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 betrachten 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.
- Wir legen folgende Datendarstellung fest:
  - r: als natürliche Zahl in Dezimaldarstellung.
  - w: als Folge von Werten 1, 2 oder 5.
- Wir benutzen dabei die Bezeichnung $()$ für die *leere Folge* und die Funktion $\circ$ zur Konkatenation zweier Folgen wie oben. Desweiteren sagen wir auch “nimm x zu w hinzu” für die Operation $w \circ x$.

Erster Lösungsansatz

- Idee:
  Ausgehend von r sukzessive um 1,2 und 5 hochzählen (unter Hinzunahme der entsprechenden Münzen/Scheine) bis man bei 100 angelangt ist. Dabei möglichst schnell eine durch 5 teilbare Zahl erreichen, um möglichst wenige Münzen/Scheine auszugeben.
- Algorithmus *Wechselgeld 1*

  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.

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	$r = 90$	$w = (2, 2, 5)$	
	$r = 95$	$w = (2, 2, 5, 5)$	
	$r = 100$	$w = (2, 2, 5, 5, 5)$	

- 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 *Wechselgeld 2*

  Führe folgende Schritte der Reihe nach aus:

  1. Setze $w = ()$.
  2. Solange $r < 100$: Führe jeweils (wahlweise) einen der folgenden Schritte aus:
     1. Falls die letzte Ziffer von r eine 2,4,7 oder 9 ist, dann erhöhe r um 1 und nimm 1 zu w hinzu.
     2. 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.
     3. 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.

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	$r = 100$	$w = (2, 2, 5, 5, 5)$	

- 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:
  - Algorithmus 2 arbeitet nach dem selben Grundprinzip wie Algorithmus 1, läßt aber gewisse "Freiheiten" bei der Auswahl des nächsten Schrittes beim wiederholt auszuführenden Schritt 2.
    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 $w = (2, 5, 2, 5, 5)$.
  - Bemerkung: 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.
- Die Wahl der Datendarstellung kann also Einfluss auf Eigenschaften von Algorithmen haben.
- Bemerkung: Würden wir r nicht in Dezimal- sondern z.B. in Binärdarstellung modellieren, wären beide Algorithmen in der angegebenen Form sinnlos.

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 die selbe Darstellung gebracht werden, die auch für r benutzt wird.
- Algorithmus *Wechselgeld 3*

  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.

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

- 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.
- Eine triviale Lösung 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.
- Allgemein wird man eine kompakte Darstellung suchen, in der die zu bestimmende Abbildung in geeigneter Weise aus einfachen (elementar auswertbaren) Abbildungen zusammengesetzt ist.

Vorbereitungen

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

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

- DIV berechnet das Ergebnis der ganzzahlige Division zweier natürlicher Zahlen, z.B. $DIV(7,3) = 2$.
- 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 $r < l$ und $k = q \cdot l + r$. Dann ist

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

Zweiter Lösungsansatz

- Idee:
  Ähnlich wie Algorithmus 3 teilen wir $100 - r$ 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 Elemente 2 und r_2 die Anzahl der Elemente 1 in $h(r)$.
- Algorithmus *Wechselgeld 4*

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

  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.

Zweiter Lösungsansatz

- Bemerkung:
  In dieser Schreibweise kann z.B. $(5_1, \ldots 5_0)$ auftreten, was bedeuten soll, dass die Folge kein Element 5 enthält.
- Beispielanwendung $r = 81$:

$$\begin{aligned} DIV(100 - r, 5) &= DIV(19, 5) = 3 \\ MOD(100 - r, 5) &= MOD(19, 5) = 4 \\ DIV(MOD(100 - r, 5), 2) &= DIV(4, 2) = 2 \\ MOD(MOD(100 - r, 5), 2) &= MOD(4, 2) = 0 \end{aligned}$$

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

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

Zweiter 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" sondern beschreibt den "funktionalen Zusammenhang" zwischen r und w durch ineinander eingesetzte Anwendungen gewisser Basisoperationen.

Variante des zweiten 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 Mengen) ist das Prinzip der *Rekursion*, das wir im nächsten Abschnitt noch etwas genauer untersuchen, hier aber schon einmal informell einführen.
- Kernidee einer rekursiven Definition einer Abbildung f mit definitionsbereich $\mathbb{N}_0$ ist es, für einen oder mehrere *Basisfälle* die Abbildung explizit anzugeben (typischerweise $f(0)$) und den allgemeinen *Rekursionsfall* $f(n)$ auf den Fall $f(n-1)$ zurückzuführen.

Variante des zweiten Lösungsansatzes

- Idee:
  Auch im Fall der Abbildung $h : \mathbb{N}_0 \rightarrow$ "Folgen über 1,2,5" kann man sehr leicht eine rekursive Definition finden, die die Operationen *DIV* und *MOD* nicht mehr verwendet.
- Algorithmus *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 zweiten 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 zweiten 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.
- 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.

Zweite Variante des zweiten Lösungsansatzes

- Algorithmus *Wechselgeld 6*

  Setze $w = ()$ und führe anschließend folgenden in Abhängigkeit der Eingabegröße 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.

Diskussion

- Grundlegende algorithmische Konzepte sind (z.T. abhängig vom verwendeten Programmierparadigma):
  - 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.

Unterschiedliche Programmierparadigmen

- *Imperative Algorithmen*:
  - Imperative Algorithmen spezifizieren die Abfolge von Aktionen, die typischerweise bestimmte Größen verändern.
  - Diese Auffassung spiegelt auch die technischen Möglichkeiten von Rechneranlagen wider, die Schritt für Schritt bestimmte grundlegende Aktionen ausführen können.
- *Funktionale Algorithmen*:
  - Funktionale Algorithmen spezifizieren eine auswertbare Darstellung des funktionalen Zusammenhangs zwischen Ein- und Ergebniswerten.
  - Diese Auffassung spiegelt ein höheres Abstraktionsniveau wider: Obwohl intern (d.h. zur Auswertung der entsprechenden Abbildung) wiederum bestimmte Schritte ausgeführt werden, ist die eigentliche Spezifikation des Algorithmus als Abbildung praktisch losgelöst von den technischen Möglichkeiten der Rechenanlage.
- *Objektorientierte Algorithmen*:
  - Objektorientierte Algorithmen spezifizieren eine höhere Abstraktionsebene zur Darstellung von komplexen Eingabe- und Ergebnisdaten sowie Zwischenzuständen während der Berechnung.
  - Zur eigentlichen Lösung der gegebenen Aufgabe können sie sowohl die funktionale als auch die imperative Auffassung verwenden.