## 1 Datenstrukturen





## Abstrakte Datentypen

- Spezifizieren Form und Funktionalität der zu verarbeitenden Daten
- Datenobjekte, Datenfelder, “Sorten”
- Funktionen
- Axiome

## Abstrakte Datentypen

- Spezifizieren Form und Funktionalität der zu verarbeitenden Daten
- Datenobjekte, Datenfelder, “Sorten”
- Funktionen
- Axiome

Zeit: hh:mm:ss



## Abstrakte Datentypen

- Spezifizieren Form und Funktionalität der zu verarbeitenden Daten
- Datenobjekte, Datenfelder
- Funktionen
- Axiome

Add: Zeit×Zeit→Zeit

## Abstrakte Datentypen

- Spezifizieren Form und Funktionalität der zu verarbeitenden Daten
- Datenobjekte, Datenfelder
- Funktionen
- Axiome

$$\begin{aligned}&Add([h_1,m_1,s_1],[h_2,m_2,s_2]) =\\ &[(h_1+h_2+(m_1+m_2+(s_1+s_2)/60)/60) \% 24,\\ &(m_1+m_2+(s_1+s_2)/60) \% 60,\\ &(s_1+s_2)) \% 60]\end{aligned}$$



## Abstrakte Datentypen

- Warum abstrakt?
  - Keine Hinweise darauf, wie die jeweiligen Funktionen implementiert werden.
- Warum Typdefinitionen?
  - Spezifikation / Verifikation
  - Top-down Software-Entwurf



Abstrakt bedeutet, es gibt keine „konkrete“ Implementierung oder Vorschriften.

## Beispiel: Bool

- Wertebereich: { true, false }
- ¬: bool → bool
  ∧: bool × bool → bool
  ∨: bool × bool → bool
- ¬ true = false, ¬ false = true
  x ∧ true = x, x ∧ false = false,
  x ∨ true = true, x ∨ false = x



Zentrale Frage: Wieviele Axiome werden gebraucht, um alle Fälle abzudecken. Das ist einfach
für das gegebene Beispiel, da die Menge der Booleans endlich ist. Was ist mit unendlichen Mengen?

## Beispiel: Bool

- Wertebereich: { true, false }
- ¬: bool → bool
  ∧: bool × bool → bool
  ∨: bool × bool → bool
- ¬ true = false, ¬ false = true

| ∧ | true | false |
|---|---|---|
| true | true | false |
| false | false | false |

| ∨ | true | false |
|---|---|---|
| true | true | true |
| false | true | false |



Wertebereich hat Kardinalität zwei. Deshalb können alle möglichen Fälle für die Axiome einfach manuell geprüft werden.

## Aufzählungstypen

- Endlicher Wertebereich
  - Vollständige Spezifikation durch Tabellen
- Verallgemeinerung
  - Aufzählungstypen: enum
  - z.B. Wochentage, chemischeElemente, ...
  - Kodierung als ganze Zahlen modulo n (implizite Ordnungsrelation)



## Beispiel: Integer

- Wertebereich: $Z = N_+ \cup \{0\} \cup -N_+$
- $+: Z \times Z \rightarrow Z$
  $-: Z \times Z \rightarrow Z$
  $-: Z \rightarrow Z$
  $\times: Z \times Z \rightarrow Z$
  $\div: Z \times Z \rightarrow Z$
  ...

## Beispiel: Integer

$$+(\ \ 0,b) = b$$
$$+(a+1,b) = +(a,b) + 1$$
$$+(a-1,b) = +(a,b) - 1$$

$$-(\ \ b,b) = 0$$
$$-(a+1,b) = -(a,b) + 1$$
$$-(a-1,b) = -(a,b) - 1$$

$$-(b) = -(0,b)$$

$$\times(\ \ 0,b) = 0$$
$$\times(a+1,b) = +(\times(a,b), b)$$
$$\times(a-1,b) = -(\times(a,b), b)$$

$$\div(a,0) = \text{ERROR}$$
$$\div(a,b) = 0 \text{ if } |a|<|b|$$
$$\div(a+b,b) = \div(a,b) + 1$$
$$\div(a-b,b) = \div(a,b) - 1$$



Reduktion auf Trivialfälle. Axiome bestimmen gesamtes Regelwerk für die Arithmetik mit z.B. ganzen Zahlen. Alle Funktionen können auf triviale Elementaroperationen zurückgeführt werden. Intention: Rekursive Definition, bei denen die Platzhalter entweder hoch oder herunter gezählt werden bis Terminalausdruck erreicht wird.



## Beispiel: Integer

count down to 0

count up to 0

$$+(\ \ 0,b) = b$$
$$+(a+1,b) = +(a,b) + 1$$
$$+(a-1,b) = +(a,b) - 1$$

$$-(\ \ b,b) = 0$$
$$-(a+1,b) = -(a,b) + 1$$
$$-(a-1,b) = -(a,b) - 1$$

$$-(b) = -(0,b)$$

$$\times(\ \ 0,b) = 0$$
$$\times(a+1,b) = +(\times(a,b), b)$$
$$\times(a-1,b) = -(\times(a,b), b)$$

$$\div(a,b) = 0 \text{ if } |a|<|b|$$
$$\div(a+b,b) = \div(a,b) + 1$$
$$\div(a-b,b) = \div(a,b) - 1$$

## Beispiel: Integer

$+(\ \ 0,b) = b$
$+(a+1,b) = +(a,b) + 1$
$+(a-1,b) = +(a,b) - 1$

$-(\ \ b,b) = 0$
$-(a+1,b) = -(a,b) + 1$
$-(a-1,b) = -(a,b) - 1$

$-(b) = -(0,b)$

$\times(\ \ 0,b) = 0$
$\times(a+1,b) = +(\times(a,b), b)$
$\times(a-1,b) = -(\times(a,b), b)$
$\div(a,0) = \text{ERROR}$
$\div(a,b) = 0 \text{ if } |a|<|b|$
$\div(a+b,b) = \div(a,b) + 1$
$\div(a-b,b) = \div(a,b) - 1$

count down to b

count up to b



## Beispiel: Integer

$+(\ \ 0,b) = b$
$+(a+1,b) = +(a,b) + 1$
$+(a-1,b) = +(a,b) - 1$

$-(\ \ b,b) = 0$
$-(a+1,b) = -(a,b) + 1$
$-(a-1,b) = -(a,b) - 1$

$-(b) = -(0,b)$

$\times(\ \ 0,b) = 0$
$\times(a+1,b) = +(\times(a,b), b)$
$\times(a-1,b) = -(\times(a,b), b)$
$\div(a,0) = \text{ERROR}$
$\div(a,b) = 0 \text{ if } |a|<|b|$
$\div(a+b,b) = \div(a,b) + 1$
$\div(a-b,b) = \div(a,b) - 1$

## Beispiel: Integer

- noch abstrakter ...
  0
  N = { S(0), S(S(0)), ... }
  −N = { P(0), P(P(0)), ... }
- +( 0,y) = y
  +(S(x),y)= S(+(x,y))
  +(P(x),y)= P(+(x,y))
- ...



Es gibt eine Integerzahl 0. Und dann nur noch Nachfolger (S = Successor) und Vorgänger (P = Predecessor).
Alle Axiome, die vorher vorgestellt wurden, können auch in dieser Form dargestellt werden.

Beispiel: Die Zahl 4 wäre S(S(S(S(0))))
Die Zahl -2 ist P(P(0))

## Skalare Typen

- „Zahlen“ mit 1-dim Wertebereich
  - Char
  - Integer
  - Float, Double
- Achtung: In *realen* Implementierungen haben skalare Typen meistens einen endlichen Wertebereich
  - Integer : $[1-2^{b-1} .. 2^{b-1}]$
  - Double : $[10^{-300} .. 10^{300}]$

Overflow Error



Integer i.d.R. 32 Bit, d.h., der Typ int ist eine (meistens) ausreichende Approximation der ganzen Zahlen, aber halt nur im Bereich 1-2^(b-1) und 2^(b-1). Sollte dieser Bereich überschritten werden, führt dies zum Overflow Error.

## Zusammengesetzte Typen

- Endliche / feste Kombination von skalaren Typen oder zusammengesetzten Typen
  - k-dim Vektoren
  - Adressen-Eintrag
  - ...
- Mehrdimensionaler Wertebereich



## Beispiel: Sequenz

- Wertebereich: $W = \{\} \cup N \cup N^2 \cup N^3 \cup \ldots$
  $\neq 2^N$ !!!
- Create : $\rightarrow W$
  Append : $N \times W \rightarrow W$
  Remove: $N \times W \rightarrow W$



Sequenz schreibt nicht vor, aus wie vielen Elementen sie besteht.

## Beispiel: Sequenz

- Append(x,{$y_1$, ..., $y_n$}) = {$y_1$, ..., $y_n$,x}

Append(z,Append(y,Append(x,Create()))) = {x,y,z}

| | |
|---|---|
| Remove(Create()) | = Create() |
| Remove(x,Append(x,z)) | = z |
| Remove(x,Append(y,z)) | = Append(y,Remove(x,z)) if x ≠ y |



Die Elemente in geschweiften Klammern sind die Sequenz und unterliegen einer festen Ordnung. Create repräsentiert die leere Sequenz. Append(x,s) fügt der Sequenz s das Element x an. Remove(x,s) macht das Gegenteil.

## Lineare Strukturen

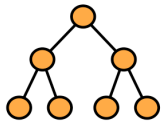
- Beliebige Sequenz von Basisobjekten (skalare / zusammengesetzte Typen) mit variabler Länge
  - Arrays
  - Listen (einfach/doppelt verkettet)
  - Queue (FIFO)
  - Stack (LIFO)
- Unterscheiden sich im Wesentlichen in der Zugriffsfunktionalität



FIFO = First in first out (das Element, das zuerst eingefügt wurde, wird auch zuerst wieder herausgenommen)
LIFO = Last in first out (das Element, das zuletzt eingefügt wurde, wird zuletzt herausgenommen)

## Baumstrukturen

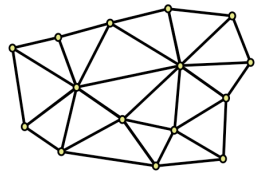
- Hierarchische Strukturen
- Vater/Sohn-Beziehung („der“ Knoten)
- Keine Zyklen
- Eindeutige Vorfahren





## Graphen

- Beliebige topologische Struktur
- Nachbarschaftsbeziehungen zwischen Knoten
- Wichtige Spezialfälle
  - Planare Graphen
  - Gerichtete Graphen
  - Zyklenfreie Graphen
  - ...





Jeder Pfad in einem Baum kann auch als Sequenz aufgefasst werden.
Datenstruktur Baum ist nicht mehr linear.

## Datentypen-Typen

- Aufzählungstypen (bool, enum)
- Skalare Typen (char, int, float, ..)
- Zusammengesetzte Typen (struct, class)
- Lineare Strukturen (list, queue, stack)
- Bäume
- Graphen



## Abstrakte Datentypen

- Spezifizieren Form und Funktionalität der zu verarbeitenden Daten
- Datenobjekte, Datenfelder
- Funktionen
- Axiome



Warum müssen wir abstrakte Datenstrukturen betrachten? Damit wir diese präzise in der Sprache der Mathematik formulieren und spezifizieren können. Diese Klasse der Datenstrukturen eignet sich besonders gut für Korrektheitsbeweise.

## Konkrete Datentypen

- Spezifizieren Form und Funktionalität der zu verarbeitenden Daten
- Datenobjekte, Datenfelder
- Funktionen
- Axiome  Methoden / Algorithmen



## Java Klassen

- Datenobjekte → Attribute
- Funktionen → Methoden-Header (Rückgabetyp, formale Parameter)
- Algorithmen → Methoden-Body
- Überprüfe korrekte Methodenimplementierung anhand der abstrakten Axiome