Imperative Programmierung (IPR)

Kapitel 2: Abstrakte Datentypen

Univ.-Prof. Dr.-Ing. habil. Gero Mühl

Lehrstuhl für Architektur von Anwendungssystemen (AVA)
Fakultät für Informatik und Elektrotechnik (IEF)
Universität Rostock

Universität Rostock

Traditio et Innovatio

AVA

Inhalte

1. Motivation

2. Spezifikation abstrakter Datentypen

3. Realisierung abstrakter Datentypen durch Algebren

4. Ausführbare und vollständige Spezifikationen

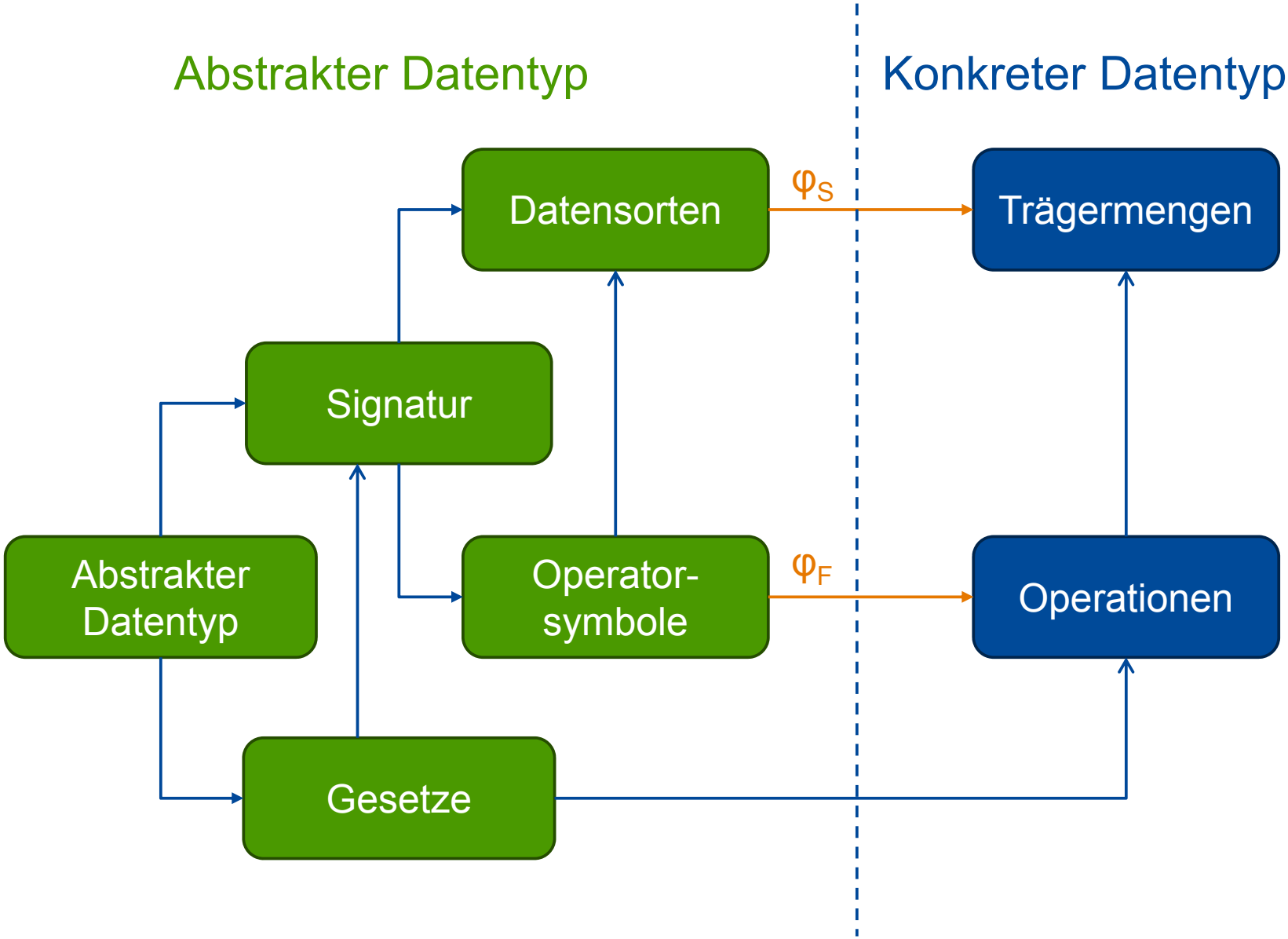
Kapitel 2.1

Motivation

Motivation

- **Wertebereiche** mit den auf ihnen definierten **Operationen** werden in der Informatik **konkreter Datentyp** oder **Datenstruktur** genannt.
- In der Mathematik gibt es hierfür den Begriff der **Algebra**.
- Ein Beispiel für einen konkreten Datentyp sind die positiven ganzen Zahlen mit der Addition und der Multiplikation als Operationen.
- Oft wird ein Datentyp nicht bereits zu Beginn des Entwicklungsprozesses in *allen* Details festgelegt.
- Stattdessen werden zunächst nur die Eigenschaften des Datentyps geeignet spezifiziert.
- Dies führt zu den **abstrakten Datentypen**, die lediglich aus **Signaturen** und **Gesetzen** bestehen.

Zusammenhänge



Kapitel 2.2

Spezifikation abstrakter Datentypen

Abstrakter Datentyp

Definition 1 (Abstrakter Datentyp)

Ein **abstrakter Datentyp** *ist ein Paar* (Σ, E) *bestehend aus einer Signatur* Σ *und einer Menge von Gesetzen* E*.*

- Wir konzentrieren uns zunächst auf die Signatur und behandeln die Gesetze später.

Definition 2 (Signatur)

Eine **Signatur** Σ *ist ein Paar* (S, F) *mit:*

- S *ist eine Menge von* **Sorten**.
- F *ist eine Menge von* **Operatorsymbolen**.
- *Jedes Operatorsymbol hat als Vorbereich das kartesische Produkt einer (potentiell leeren) Untermenge der Sorten und als Zielbereich genau eine der Sorten.*

Beispiel: Boolscher Datentyp

Beispiel 1 (Signatur des Boolschen Datentyps)

- Menge der Sorten: $S = \{b\}$.
- Die Menge der Operatorsymbole F beinhaltet:

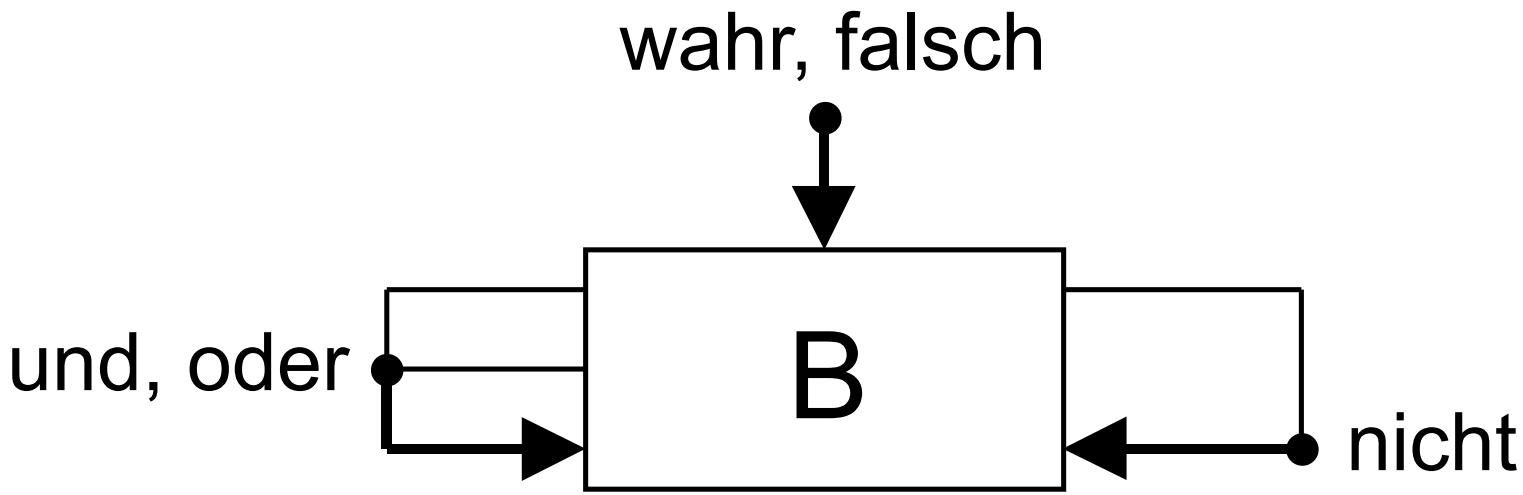
$$
\begin{aligned}
\mathit{wahr} &: \varnothing \longrightarrow b \\
\mathit{falsch} &: \varnothing \longrightarrow b \\
\mathit{nicht} &: b \longrightarrow b \\
\mathit{und} &: b \times b \longrightarrow b \\
\mathit{oder} &: b \times b \longrightarrow b
\end{aligned}
$$

- Zum Beispiel haben die Operatorsymbole *und* und *oder* den Vorbereich $b \times b$ und den Zielbereich b.

- Definitionsbereich $\subseteq$ Vorbereich
- Wertebereich $\subseteq$ Zielbereich

Grafische Darstellung der Signatur

- Beispiel Boolscher Datentyp



Beispiel: Datentyp der natürlichen Zahlen

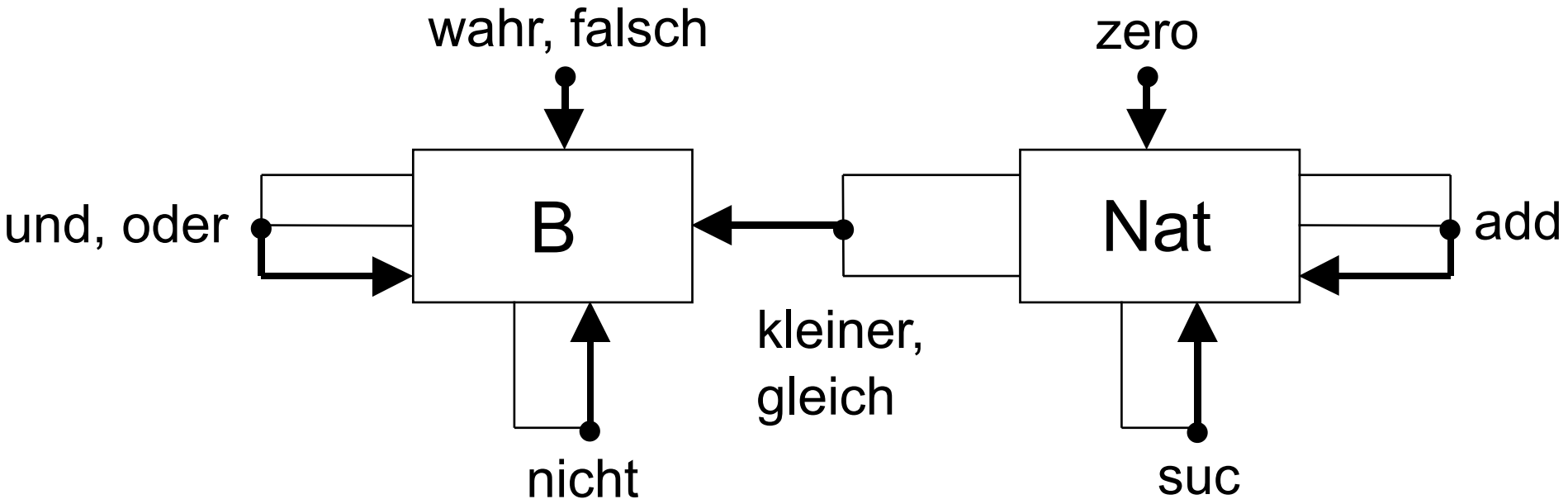
Beispiel 2 (Signatur des Datentyps der natürlichen Zahlen)

- Menge der Sorten: $S = \{b, n\}$.
- Die Menge der Operatorsymbole F beinhaltet neben denen der booleschen Algebra noch folgende Operatorsymbole:

$$
\begin{aligned}
\mathit{zero} &: \varnothing \longrightarrow n \\
\mathit{suc} &: n \longrightarrow n \\
\mathit{add} &: n \times n \longrightarrow n \\
\mathit{mult} &: n \times n \longrightarrow n \\
\mathit{div} &: n \times n \longrightarrow n \\
\mathit{rest} &: n \times n \longrightarrow n
\end{aligned}
\qquad
\begin{aligned}
\mathit{gleich} &: n \times n \longrightarrow b \\
\mathit{ungleich} &: n \times n \longrightarrow b \\
\mathit{kleiner} &: n \times n \longrightarrow b \\
\mathit{groesser} &: n \times n \longrightarrow b \\
\mathit{kleinergleich} &: n \times n \longrightarrow b \\
\mathit{groessergleich} &: n \times n \longrightarrow b \\
\mathit{gerade} &: n \longrightarrow b
\end{aligned}
$$

Grafische Darstellung der Signatur

- Beispiel Datentyp der natürlichen Zahlen



Signatur und Gesetze

- Eine Signatur bestimmt lediglich die Vor- und Zielbereiche der Operatorsymbole.
- Sie legt aber *nicht* fest, welche Eigenschaften die Operatoren haben müssen.
- Daher reicht eine Signatur zur vollständigen Beschreibung der Eigenschaften eines Datentyps nicht aus.
- Hierfür werden zusätzlich **Gesetze** benötigt.
- Die Herausforderung liegt darin, eine Menge von Gesetzen aufzustellen, so dass der Datentyp *alle gewünschten* Eigenschaften hat, aber keine *unnötigen* Einschränkungen gefordert werden.
- Bevor der Begriff Gesetz formal eingeführt wird, müssen zunächst **Terme** einführt werden.

Terme

- Terme werden mit Hilfe der Operatorsymbole auf Basis der Signatur gebildet.
  - $add(suc(suc(zero)), zero)$
  - $und(oder(wahr, falsch), nicht(falsch))$.
- Dabei kommt es auf die genaue Abbildung, die ein Operatorsymbol beschreibt, zunächst gar nicht an.
- Wichtig ist erst mal nur seine **Stelligkeit** (d. h. die Anzahl der Operanden) und die korrekte Einsetzung in den Term.
- Darüber hinaus müssen die Zielbereiche der Unterterme mit den Vorbereichen des jeweiligen Terms identisch sein.

Terme

- Terme können auch Variablen enthalten.
  - add(suc(x),y)
- Deswegen wird für eine Signatur eine Menge von Variablen festgelegt und jeder Variable eine Sorte zugeordnet.
- Die **Menge aller Variablen** X besteht dann aus den disjunkten Mengen der Variablen der einzelnen Sorten X_s, d. h.

$$X = \cup_{s \in S} X_s \text{ mit } s \neq s' \Rightarrow X_s \cap X_{s'} = \emptyset$$

Terme

Definition 3

Sei $\Sigma = (S, F)$ eine Signatur und $X = \cup_s X_s$ ein Menge von Variablen, wobei X_s die Menge der Variablen der Sorte s ist. Dann ist die **Menge der Terme $T_{\Sigma,X}$ über Σ und X** *die kleinste Menge mit den Eigenschaften:*

1. *Alle Variablen sind Terme:*
   Für alle $x \in X_s$ ist $x \in T_{\Sigma,X}$.
2. *Alle nullstelligen Operatorsymbole sind Terme:*
   Für alle Operatorsymbole $f \in F : \varnothing \longrightarrow s$ mit $s \in S$ ist $f \in T_{\Sigma,X}$.
3. *Alle (nicht nullstelligen) Operatorsymbole mit passenden Termen als Argumente sind Terme:*
   Für alle n-stelligen Operatorsymbole $f \in F : s_{i_1} \times \ldots s_{i_n} \longrightarrow s$ und Terme $t_1, \ldots, t_n$ mit $ziel(t_j) = s_{i_j}$ ist $f(t_1, \ldots, t_n) \in T_{\Sigma,X}$.

Beispiele für Terme

Beispiel 3

- Betrachten wir die Boolesche Signatur b mit der Variablenmenge $X_b = \{x_1, x_2\}$.
- Folgende Ausdrücke sind Terme über b und X_b:
  - $\mathit{nicht}(\mathit{wahr})$
  - $\mathit{und}(x_1, \mathit{nicht}(x_2))$
  - $\mathit{oder}(\mathit{wahr}, \mathit{und}(\mathit{nicht}(x_1), \mathit{nicht}(\mathit{nicht}(\mathit{falsch}))))$.
- Folgende Ausdrücke sind *keine* Terme über b und X_b:
  - $\mathit{und}(x_1)$ — zu wenige Argumente bei *und*
  - $\mathit{oder}(x_1, x_2, \mathit{nicht}(\mathit{wahr}))$ — zu viele Argumente bei *oder*
  - $\mathit{und}(\mathit{nicht}(x_1), x_3)$ — $x_3 \notin X_b$

Beispiele für Terme

Beispiel 4

- Betrachten wir die Signatur der natürliche Zahlen n mit der Variablenmenge $X = X_b \cup X_n$ mit $X_b = \{x_1\}$ und $X_n = \{y_1, y_2, y_3\}$.
- Folgende Ausdrücke sind Terme über n und X:
  - $kleiner(y_1, y_2)$
  - $add(y_1, y_2)$
  - $add(suc(suc(y_1)), zero)$
  - $gleich(mult(add(y_1, y_2), y_3), add(y_1, y_2))$
  - $oder(nicht(x_1), kleinergleich(suc(zero), add(y_1, y_2)))$
- Folgende Ausdrücke sind *keine* Terme über n und X:
  - $nicht(y_1)$ — $y_1 \notin X_b$
  - $kleiner(x_1, suc(zero))$ — $x_1 \notin X_n$
  - $add(y_1, y_2, y_3)$ — zu viele Argumente bei *add*
  - $nicht(add(y_1, y_2))$ — Zielbereich des Unterterms passt nicht

Präfix- vs. Infix- vs. Postfix-Notation von Termen

- Die angegebene Definition von Termen führt zur sogenannten **Polnischen Notation** (auch **Präfixnotation**).
- Ein in der üblichen **Infixnotation** formulierter Term

$$(3 * a - 7) * (4 + b)$$

hat in Polnischer Notation folgende Darstellung:

$$* \; - \; * \, 3 \; a \; 7 \; + \; 4 \; b$$

- Es gibt auch die **Umgekehrte Polnische Notation** (auch **Postfixnotation**). Für obiges Beispiel lautet diese:

$$3 \; a \; * \; 7 \; - \; 4 \; b \; + \; *$$

- Hinweis: Nur bei der Infix-Notation sind Klammern notwendig.

Darstellung von Termen als Bäume

- Terme lassen sich als Bäume darstellen.

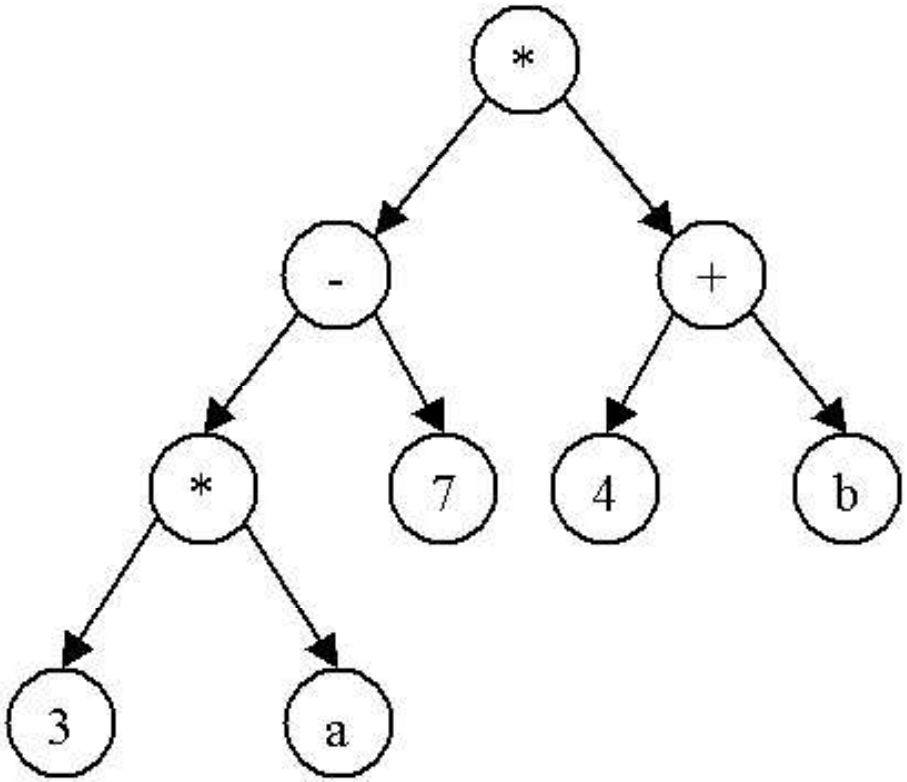
Definition 4 (Kantorovic-Bäume)

1. *Sei $x \in X_s$ eine Variable. Der dazugehörige Kantorovic-Baum ist der einzelne Knoten x.*
2. *Sei f ein 0-stelliges Operatorsymbol. Der dazugehörige Kantorovic-Baum ist der einzelne Knoten f.*
3. *Sei $f(t_1, \ldots, t_n)$ ein Term. Der dazugehörige Kantorovic-Baum besteht aus dem Elternknoten f, der n Kinderknoten hat, wobei der i-te Kinderknoten der Kantorovic-Baum des Terms t_i ist.*

Darstellung von Termen als Bäume

Beispiel 5

- Kantorovic-Baum des Terms $mult(sub(mult(3, a), 7), add(4, b))$



Gesetze

Definition 5 (Menge der potentiellen Gesetze)

Gegeben sei eine Signatur $\Sigma = (S, F)$, eine Variablenmenge X und die zugehörige Menge von Termen $T_{\Sigma,X}$. Ferner bezeichne $b \in S$ die boolesche Sorte. Die **Menge der Gesetze $G_{\Sigma,X}$ über Σ und X** *ist die kleinste Menge mit folgenden Eigenschaften:*

1. *Jeder Term mit der booleschen Sorte als Zielsorte ist ein Gesetz.*
   Falls $t \in T_{\Sigma,X}$ und $ziel(t) = b$, dann ist $t \in G_{\Sigma,X}$.
2. *Die Gleichsetzung zweier Terme mit gleicher Zielsorte ist ein Gesetz.*
   Falls $t_1, t_2 \in T_{\Sigma,X}$ und $ziel(t_1) = ziel(t_2)$, dann ist $(t_1 = t_2) \in G_{\Sigma,X}$.
3. *Die logische Negation eines Gesetzes ist ein Gesetz.*
   Falls $g \in G_{\Sigma,X}$, dann ist $not(g_1) \in G_{\Sigma,X}$.

Gesetze

Definition 5 (Menge der potentiellen Gesetze)

4. *Die logische Verknüpfung zweier Gesetze mit einem der Operatorsymbole $\wedge$, $\vee$, $\Rightarrow$ und $=$ ist ein Gesetz.*
   Falls $g_1, g_2 \in G_{\Sigma,X}$, dann sind $(g_1 \wedge g_2)$, $(g_1 \vee g_2)$, $(g_1 \Rightarrow g_2)$ und $(g_1 = g_2)$ in $G_{\Sigma,X}$.
5. *Die Quantifizierung eines Gesetzes über einer freien Variable (mittels Existenzquantor oder Allquantor) ist wieder ein Gesetz.*
   Falls $g \in G_{\Sigma,X}$ und $x \in X$, dann sind $(\exists\, x : g)$ und $(\forall\, x : g)$ in $G_{\Sigma,X}$.

- Enthält die Signatur die boolesche Sorte nicht, entfällt (1).

Beispiel: Boolescher Datentyp

Beispiel 6 (Exemplarische Gesetze)

- Seien $x, y \in X_b$:

$$\begin{aligned} \mathit{nicht}(\mathit{wahr}) &= \mathit{falsch} \\ \mathit{nicht}(\mathit{falsch}) &= \mathit{wahr} \\ \mathit{nicht}(\mathit{nicht}(x)) &= x \end{aligned}$$

$$\begin{aligned} \mathit{und}(\mathit{wahr}, x) &= x \\ \mathit{und}(\mathit{falsch}, x) &= \mathit{falsch} \\ \mathit{und}(x, y) &= \mathit{und}(y, x) \end{aligned}$$

$$\begin{aligned} \mathit{oder}(\mathit{wahr}, x) &= \mathit{wahr} \\ \mathit{oder}(\mathit{falsch}, x) &= x \\ \mathit{oder}(x, y) &= \mathit{oder}(y, x) \end{aligned}$$

Beispiel: Datentyp der natürlichen Zahlen

Beispiel 7 (Exemplarische Gesetze)

- Seien $x, y \in X_n$:

$$
\begin{aligned}
add(x, zero) &= x \\
add(x, suc(y)) &= suc(add(x, y)) \\
\\
mult(x, zero) &= zero \\
mult(x, suc(y)) &= add(mult(x, y), x) \\
\\
gleich(zero, zero) &= wahr \\
gleich(zero, suc(x)) &= falsch \\
gleich(suc(x), zero) &= falsch \\
gleich(suc(x), suc(y)) &= gleich(x, y)
\end{aligned}
$$

Beispiel: Datentyp der natürlichen Zahlen

Beispiel 8 (Einige Gesetze)

- Seien $x, y \in X_n$:

$$\begin{aligned}
kleiner(x, zero) &= falsch \\
kleiner(zero, suc(x)) &= wahr \\
kleiner(suc(x), suc(y)) &= kleiner(x, y) \\
\\
ungleich(x, y) &= nicht(gleich(x, y)) \\
kleinergleich(x, y) &= oder(kleiner(x, y), gleich(x, y)) \\
groesser(x, y) &= nicht(kleinergleich(x, y)) \\
groessergleich &= nicht(kleiner(x, y))
\end{aligned}$$

Kapitel 2.3

Realisierung abstrakter Datentypen durch Algebren

Realisierung abstrakter Datentypen

- Wurde ein abstrakter Datentyp vollständig spezifiziert, kann er durch einen konkreten Datentyp realisiert werden.
- Der konkrete Datentyp muss sämtliche Gesetze des abstrakten Datentyps erfüllen.
- Er wird dann auch **Modell des abstrakten Datentyps** genannt.
- Um dies formaler zu fassen, werden zunächst **Algebren** als konkrete Datentypen formal definiert.
- Im Anschluss wird der Begriff der **Interpretation** eingeführt, mit dem die Einhaltung der Gesetze nachgewiesen werden kann.

Algebra

Definition 6 (Algebra)

Eine **Algebra** *ist ein Paar* $(M_1, \ldots, M_n; f_1, \ldots, f_m)$ *bestehend aus einer Menge von* **Trägermengen** M_i *und einer Menge von* **Operationen** f_j. *Der Vorbereich jeder Operation ist das (potentiell leere) kartesische Produkt einer Teilmenge der Trägermengen, während der Zielbereich jeder Operation genau eine Trägermenge ist.*

- Algebren mit genau einer Trägermenge sind **homogen**.
- Algebren mit zwei oder mehr unterschiedlichen Trägermengen werden **heterogen** genannt.

Beispiel Algebra

Beispiel 9 (Eine Boolesche Algebra)

$$M = \{0, 1\}$$

$$f_{wahr} = 1$$
$$f_{falsch} = 0$$
$$f_{nicht} = \{0 \mapsto 1, 1 \mapsto 0\}$$
$$f_{und} = \{(0,0) \mapsto 0, (0,1) \mapsto 0, (1,0) \mapsto 0, (1,1) \mapsto 1\})$$
$$f_{oder} = \{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 1, (1,1) \mapsto 1\})$$

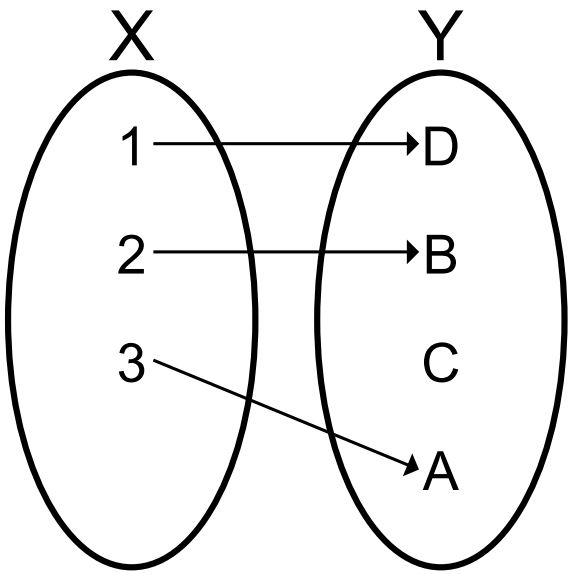
- f_{wahr} und f_{falsch} sind nullstellige Operationen mit Zielbereich M.
- f_{nicht} ist eine einstellige Operation über M mit Zielbereich M.
- f_{und} und f_{oder} sind zweistellige Operationen über $M \times M$ mit Zielbereich M.
- Diese Algebra ist homogen, da M die einzige Trägermenge ist.

Interpretation

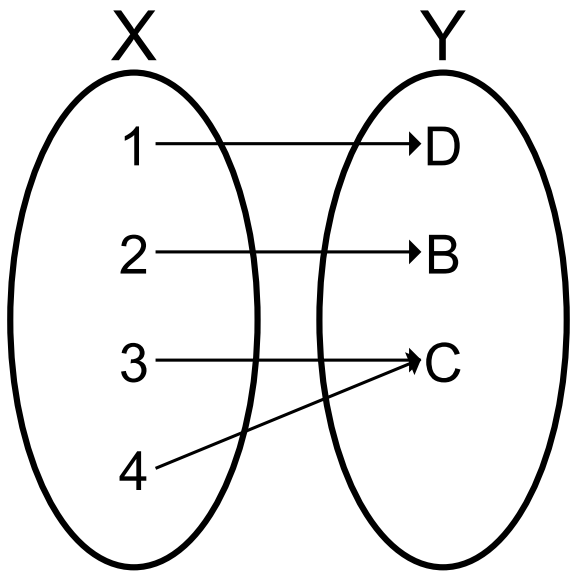
- Ist die als Beispiel gegebene Boolesche Algebra ein Modell des abstrakten Booleschen Datentyps?
- Intuitiv ja, aber um dies formal feststellen zu können, wird der Begriff der Interpretation eingeführt.
- Eine **Interpretation** bildet jede Sorte auf eine Trägermenge und jedes Operatorsymbol auf eine Operation ab.
- Diese Abbildung wird dann so erweitert, dass Terme des abstrakten Datentyps auf Terme über der Algebra abgebildet werden.
- Hierdurch können die Gesetze des abstrakten Datentyps in Gesetze über der Algebra überführt und deren Gültigkeit nachgewiesen werden.

Eigenschaften von Abbildungen

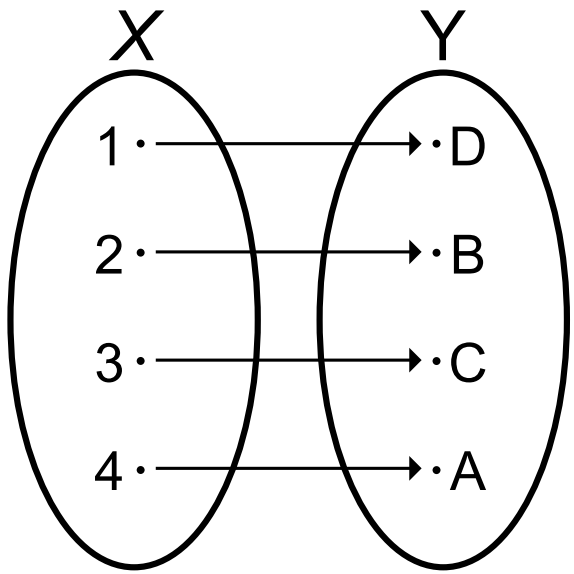
- Injektivität



- Surjektivität



- Bijektivität



- Injektivität: Für jedes $y \in Y$ gibt es *höchstens* ein $(x, y) \in R$.
  Daraus folgt $| X | \leq | Y |$.
- Surjektivität: Für jedes $y \in Y$ gibt es *mindestens* ein $(x, y) \in R$.
  Daraus folgt $| X | \geq | Y |$.
- Bijektivität: Für jedes $y \in Y$ gibt es *genau* ein $(x, y) \in R$.
  Daraus folgt $| X | = | Y |$.

Interpretation

Definition 7 (Interpretation)

Seien $\Sigma = (S, F)$ eine Signatur und $A = (M_1, M_2, \ldots; g_1, g_2, \ldots)$ eine Algebra. Eine Abbildung $\varphi : \Sigma \longrightarrow A$, die aus zwei Abbildungen $\varphi_S : S \longrightarrow \{M_1, M_2, \ldots\}$ und $\varphi_F : F \longrightarrow \{g_1, g_2, \ldots\}$ besteht, heißt **Interpretation von Σ in A** *falls gilt:*

1. *φ_S und φ_F sind total und surjektiv.*
2. *Für jedes Operatorsymbol $f \in F$ gilt:*

$$f : s_{i_1} \times \ldots \times s_{i_r} \longrightarrow s \;\Rightarrow\; \varphi_F(f) : \varphi_S(s_{i_1}) \times \ldots \varphi_S(s_{i_r}) \longrightarrow \varphi_S(s)$$

- Die zweite Bedingung verlangt, dass sich
  1. die Stelligkeit der Operatorsymbole auf die Operationen überträgt und
  2. die Trägermenge eines Arguments bzw. des Ergebnisses einer Operation aus der Sorte des entsprechenden Arguments bzw. der Zielsorte des zugeordneten Operatorsymbols ergibt.

Interpretation

Beispiel 10 (Interpretation des Booleschen Datentyps unter einer Booleschen Algebra)

$$\varphi_S = \{b \mapsto M\}$$
$$\varphi_F = \{wahr \mapsto f_{wahr}, falsch \mapsto f_{falsch}, nicht \mapsto f_{nicht},$$
$$und \mapsto f_{und}, oder \mapsto f_{oder}\}$$

- Damit wird z. B. $nicht : b \longrightarrow b$ abgebildet auf $f_{nicht} : M \longrightarrow M$
- Sowie $und : b \times b \longrightarrow b$ auf $f_{und} : M \times M \longrightarrow M$.

- Stellt die obige Interpretation die Gültigkeit der Gesetze des abstrakten Booleschen Datentyps sicher?
- Wie können wir dies feststellen?

Gültigkeit von Gesetzen unter einer Interpretation

- Um die Gültigkeit von Gesetzen unter einer Interpretation zu überprüfen, wird die Interpretation φ zu einer Abbildung φ' fortgesetzt.
- Sei x eine Variable. Dann gilt

$$\varphi'(x) = x$$

- Sei f ein nullstelliges Operatorsymbol. Dann gilt

$$\varphi'(f) = \varphi_F(f)$$

- Sei $f(t_1, \ldots, t_n)$ ein Term. Dann gilt

$$\varphi'(f(t_1, \ldots, t_n)) = \varphi_F(f)(\varphi'(t_1), \ldots, \varphi'(t_n))$$

Gültigkeit von Gesetzen unter einer Interpretation

Beispiel 11

Betrachten wir die Signatur des Datentyps der natürlichen Zahlen und die übliche Algebra der natürlichen Zahlen mit einer geeigneten Interpretation.

- Betrachten wir das Gesetz $add(x, zero) = x$.
- Der Term $add(x, zero)$ wird auf x abgebildet, da $add(x, zero) = x + 0 = x$.
- Ebenso wird der Term x auf x abgebildet. □
- Betrachten wir das Gesetz $mult(x, suc(y)) = add(mult(x, y), x)$.
- Der Term $mult(x, suc(y))$ wird auf $x \cdot (y + 1)$ abgebildet. Durch Ausmultiplizieren ergibt sich hieraus $x \cdot y + x$.
- Der Term $add(mult(x, y), x)$ wird auch auf $x \cdot y + x$ abgebildet. □

Gültigkeit von Gesetzen unter einer Interpretation

Beispiel 12 (Gültigkeit des De Morgan'schen Gesetzes)

- Betrachten wir die Signatur des Booleschen Datentyps und die folgende Algebra $A = (M; f_{wahr}, f_{falsch}, f_{und}, f_{oder}, f_{nicht})$ mit:

$$
\begin{aligned}
M &= \{0, 1\} \\
f_{wahr} &= 1 \\
f_{falsch} &= 0 \\
f_{und}(x, y) &= x \cdot y \\
f_{oder}(x, y) &= x + y - x \cdot y \\
f_{nicht}(x) &= 1 - x
\end{aligned}
$$

- Und die Interpretation aus Beispiel 10.

Gültigkeit von Gesetzen unter einer Interpretation

Beispiel 13 (Gültigkeit des De Morgan'schen Gesetzes)

- Das Gesetz

$$\mathit{nicht}(\mathit{und}(x, y)) = \mathit{oder}(\mathit{nicht}(x), \mathit{nicht}(y))$$

1. wird dann abgebildet auf:

$$1 - (x \cdot y)$$

2.

$$\begin{aligned}
&(1 - x) + (1 - y) - (1 - x) \cdot (1 - y) \\
&= (1 - x) + (1 - y) - (1 - y) + x \cdot (1 - y) \\
&= (1 - x) + x \cdot (1 - y) \\
&= 1 - (x \cdot y)
\end{aligned}$$

Modell eines abstrakten Datentyps

Definition 8

Eine Algebra A heißt **Modell** *eines abstrakten Datentyps (Σ, E), wenn es eine Interpretation $\varphi : \Sigma \longrightarrow A$ gibt, unter der alle Gesetze in E gültig sind.*

- Jetzt stellt sich die Frage, wie viele Modelle zu einem abstrakten Datentyp existieren und worin sich diese unterscheiden.
- Um diese Frage näher zu beleuchten, wird der Begriff der **abstrakten Algebra** eingeführt, der auf der **Isomorphie** von Algebren basiert.

Beispiel für isomorphe Algebren

Beispiel 14

- Algebra 1

$$A_1 = (M_1 = \{0, 1\}, \\ f_1 = \{(0,0) \mapsto 0, (0,1) \mapsto 0, (1,0) \mapsto 0, (1,1) \mapsto 1\})$$

- Algebra 2

$$A_2 = (M_2 = \{F, T\}, \\ f_2 = \{(F,F) \mapsto F, (F,T) \mapsto F, (T,F) \mapsto F, (T,T) \mapsto T\})$$

- Beide Algebren (A_1 und A_2) können als einfache boolesche Algebra mit *UND*-Verknüpfung aufgefasst werden.
- Was unterscheidet beide Algebren dann?

Isomorphismus / Isomorphe Algebren

Definition 9 (Isomorphe Algebren)

Zwei Algebren sind **isomorph**, *wenn bezüglich dieser beiden Algebren ein Isomorphismus existiert.*

- Für Isomorphie müssen bijektive Abbildungen $\phi_{i_1}, \ldots, \phi_{i_{k+1}}$ existieren, mit deren Hilfe sich die Operationen der beiden Algebren für alle Werte von $a_1, \ldots a_k$ wechselseitig ersetzen können:

$$\phi_{i_{k+1}}\big(f_i(a_1, \ldots, a_k)\big) = g_i(\phi_{i_1}(a_1), \ldots, \phi_{i_k}(a_k))$$

- Die Abbildung ϕ_i übersetzt also Werte der Trägermenge M_i in Werte der Trägermenge N_i und zwar derart, dass es egal ist, ob
  1. mit den ursprünglichen Werten die Operationen f_i berechnet und dann das Ergebnis transformiert wird oder
  2. mit den transformierten Werten die Operation g_i berechnet wird.

Isomorphismus: Allgemeines Beispiel

Beispiel 15

- Betrachten wir zwei Algebren

  $A = (f_1; M_1, M_2, M_3)$ und $B = (g_1; N_1, N_2, N_3)$

  mit $f_1 : M_1 \times M_2 \longrightarrow M_3$ und $g_1 : N_1 \times N_2 \longrightarrow N_3$.

- Gelten die folgenden zwei Bedingungen
  1. ϕ_1, ϕ_2 und ϕ_3 sind bijektiv und
  2. $\forall (a_1, a_2) \in M_1 \times M_2$ gilt $\phi_3(f_1(a_1, a_2)) = g_1(\phi_1(a_1), \phi_2(a_2))$

  mit geeignetem $\phi_1 : M_1 \longrightarrow N_1$, $\phi_2 : M_2 \longrightarrow N_2$ und $\phi_3 : M_3 \longrightarrow N_3$.

- Dann ist $\phi = \{\phi_1, \phi_2, \phi_3\}$ ein Isomorphismus bezüglich A und B.

Isomorphismus: Beispiel

Beispiel 16

- Wir betrachten die beiden Algbebren A_1 und A_2 aus Beispiel 14 und definieren die bijektive Abbildung $\phi = \{0 \mapsto F, 1 \mapsto T\}$.
- Dann gilt:
  - $\phi\big(f_1(0,0)\big) = \phi(0) = F, \quad f_2\big(\phi(0), \phi(0)\big) = f_2(F,F) = F$
  - $\phi\big(f_1(0,1)\big) = \phi(0) = F, \quad f_2\big(\phi(0), \phi(1)\big) = f_2(F,T) = F$
  - $\phi\big(f_1(1,0)\big) = \phi(0) = F, \quad f_2\big(\phi(1), \phi(0)\big) = f_2(T,F) = F$
  - $\phi\big(f_1(1,1)\big) = \phi(1) = T, \quad f_2\big(\phi(1), \phi(1)\big) = f_2(T,T) = T$
- Damit ist ϕ ein Isomorphismus bezüglich A_1 und A_2.
- Damit sind A_1 und A_2 isomorph. $\square$
- Da ϕ bijektiv ist, gilt z. B. auch $\phi^{-1}(f_2(T,T)) = \phi^{-1}(T) = 1$ und $f_1(\phi^{-1}(T), \phi^{-1}(T)) = f_1(1,1) = 1$. Für die anderen drei Kombination von Werten sind die Ergebnisse auch gleich.

Monomorphe und Polymorphe Datentypen

Definition 10 (Abstrakte Algebra)

*Die **abstrakte Algebra** zu einer Algebra A ist die Menge der zu A isomorphen Algebren (einschließlich A selbst).*

Definition 11 (Monomorpher Datentyp)

*Ein abstrakter Datentyp, für den nur isomorphe Algebren möglich sind, wird **monomorph** genannt.*

Definition 12 (Polymorpher Datentyp)

*Ein abstrakter Datentyp, für den nicht nur isomorphe Algebren möglich sind, wird **polymorph** genannt.*

- Ein monomorpher Datentyp ist bezüglich seiner Realisierungsmöglichkeiten eingeschränkt.

Polymorphe Datentypen: Beispiel

Beispiel 17

- Betrachten wir jetzt die Algebra A_3:

$$A_3 = (M_3 = \{0, 1, \ldots\},$$
$$f_3 = \{(0,0) \mapsto 0, (0,1) \mapsto 0, \ldots, (x,y) \mapsto x * y\})$$

- Auch A_3 kann (wie A_1 und A_2) als einfache boolesche Algebra mit *UND*-Verknüpfung aufgefasst werden.

- Weiterhin erfüllen A_1, A_2 und A_3 das Gesetz

$$f(zero, x) = f(x, zero) = zero$$

sofern für $A_1, A_3 : zero = 0$, für $A_2 : zero = F$ definiert wird.

Polymorphe Datentypen: Beispiel

Beispiel 18

- Allerdings ist A_3 nicht isomorph zu A_1 und A_2.
- Dies folgt direkt aus der Tatsache, dass A_3 echt mehr Elemente als A_1 und A_2 enthält.
- Daher kann keine bijektive Abbildung zwischen A_3 und A_1 (oder auch A_2) existieren.
- Ein abstrakter Datentyp, der *nur* dieses eine Gesetz enthält, ist also offensichtlich polymorph.

Kapitel 2.4

Ausführbare und vollständige Spezifikationen

Motivation

- Ein abstrakter Datentyp wird durch Angabe einer Signatur und einer Menge von Gesetzen spezifiziert, während die Umsetzung des ADTs dann durch einen konkreten Datentyp erfolgt.
- Hierfür ist nachzuweisen, dass der konkrete Datentyp alle Gesetze des abstrakten Datentyps erfüllt; dies fällt oft nicht leicht.
- Hilfreich wäre daher die Möglichkeit, direkt aus der Spezifikation eine lauffähige Implementierung des konkreten Datentyps abzuleiten.
- Dies ist mit der funktionalen Programmiersprache Haskell möglich.
- Hierfür werden die Gesetze in eine Form gebracht, die es erlaubt, jeden Term mittels **Termreduktion** schrittweise zu vereinfachen und schließlich in eine nicht weiter vereinfachbare **Normalform** zu bringen.
- Die generierte Implementierung ist zwar meist ineffizient, eignet sich aber etwa zum Testen einer anderen Implementierung.

Anforderungen an eine ausführbare Spezifikation

- Die Umwandlung der Gesetze in die ausführbare Form bzw. die Aufstellung der Gesetze in dieser Form sollte möglichst einfach sein.
- Aus jedem Term sollte nach endlich vielen Reduktionen ein Term entstehen, der keine Ersetzungen mehr erlaubt (Terminierung).
- Ausgehend von einem Term sollte jede mögliche Folge von Reduktionen am Ende zum gleichen Term führen (Determiniertheit).
- Alle berechenbaren Funktionen sollten spezifiziert werden können (Turing-Vollständigkeit).

- Wie sieht eine ausführbare Spezifikation jetzt aus?
- Hierfür definieren wir zunächst, was Termreduktion genau bedeutet.

Termreduktion

Definition 13 (Termreduktion)

Von **Termreduktion** *wird gesprochen, wenn die Gesetze in der Form* $T_1 = T_2$ *vorliegen und nur von links nach rechts angewendet werden.*

- Ein Gesetz der Form $T_1 = T_2$ kann daher so verstanden werden, dass überall dort, wo ein Term der Art T_1 steht, stattdessen ein Term der Art T_2 geschrieben werden kann.
- Passt also ein Term oder Unterterm auf die linke Seite eines Gesetzes, so kann dieser durch den Term auf der rechten Seite ersetzt werden.
- Da sich dies rein mit textueller Ersetzung durchführen lässt, werden Gesetze in diesem Kontext auch **Termersetzungsregeln** genannt.

Termreduktion

Beispiel 19 (Algebra der natürlichen Zahlen)

- Die Gesetze der natürlichen Zahlen sind bereits in der gewünschten Form $T_1 = T_2$ und werden von links nach rechts angewendet.
- Das Gesetz

$$add(suc(x), y) = suc(add(x, y))$$

kann daher folgendermaßen gedeutet werden:

Ein Term der Form

$$add(suc(x), y)$$

kann durch den Term

$$suc(add(x, y))$$

ersetzt werden. Hierbei können x und y beliebige Terme mit dem Zielbereich der natürlichen Zahlen sein.

Termreduktion

Beispiel 20 (Algebra der natürlichen Zahlen)

- Auf Basis des Gesetzes

$$add(suc(x), y) = suc(add(x, y))$$

- kann zum Beispiel der Term

$$add(suc(\underbrace{suc(zero)}_{x}), \underbrace{zero}_{y})$$

durch folgenden Term ersetzt werden

$$suc(add(\underbrace{suc(zero)}_{x}, \underbrace{zero}_{y})).$$

Termersetzungssysteme

Definition 14 (Termersetzungssystem)

Ein **Termersetzungssystem (TES)** *besteht aus einer Menge von Termersetzungsregeln.*

- Die Klasse der Termersetzungssysteme ist Turing-vollständig.
- Nach einer Folge von Termersetzungen *kann* schließlich ein Term auftreten, für den keine weitere Termersetzungen mehr möglich sind.
- In der Regel ist dies genau das Ziel und die Regeln sind so aufzustellen, das dies sichergestellt ist.

Definition 15 (Irreduzible Terme)

Ein Term T heißt **irreduzibel**, *wenn kein Teilterm von T auf die linke Seite eines der Gesetze passt. Ansonsten ist der Term* **reduzibel**.

Konvergente Termersetzungssysteme

Definition 16 (Eigenschaften von Termersetzungssystemen)

1. *Ein TES heißt* **terminierend**, *wenn es keine unendlichen Ketten von Ersetzungen erlaubt. Damit brechen für jeden Term die möglichen Ersetzungen nach endlich vielen Schritten ab.*
2. *Es heißt* **konfluent**, *wenn es zwei Terme, die durch Ersetzungen aus demselben Term hervorgegangen sind, durch weitere Ersetzungen schließlich in jedem Fall wieder auf den gleichen Term zusammenführt.*
3. *Es ist* **konvergent**, *wenn es terminierend und konfluent ist.*

- In einem terminierenden Termersetzungssystem steht am Ende jeder Ersetzungskette (nach endlich vielen Schritten) ein irreduzibler Term. Dies stellt die Existenz einer Normalform sicher.
- In einem konfluenten Termersetzungssystem ist es egal, in welcher Reihenfolge mögliche Ersetzungen durchgeführt werden. Dies stellt die Eindeutigkeit der Normalform sicher.

Konvergente Termersetzungssysteme

- In einem konvergenten Termersetzungssystem hat jeder Term (ohne Variablen) eine eindeutige **Normalform**, die am Ende der für diesen Term möglichen Kette von Ersetzungen steht.
- Ein konvergentes Termersetzungssystem wird daher auch als **vollständig** bezeichnet.
- Vollständigkeit ist wichtig, damit sich alle Implementierungen der Spezifikation gleich verhalten und um Ausführbarkeit sicherzustellen.
- Vollständigkeit impliziert Monomorphie des abstrakten Datentyps.
- In diesem Fall sind die normalisierten Terme isomorph zu den Elementen der Trägermengen der Algebra.
- Das heißt, bei Vollständigkeit lässt sich jeder Term auf ein Element einer eingebrachten Trägermenge reduzieren.
- Terme, die sich auf die gleiche Normalform bringen lassen, heißen **gleichbedeutend**: $t_1 \equiv t_2$.

Konvergente Termersetzungssysteme

Beispiel 21 (Konfluenz)

- Die Algebra der natürlichen Zahlen ist konfluent.
- Beim Term

$$add(\underbrace{add(suc(zero), zero)}_{x}, \underbrace{add(zero, suc(zero))}_{y})$$

  kann entweder zuerst die innere Summe x oder aber zuerst die innere Summe y reduziert werden.
- Dies fürt auf die beiden Terme $add(suc(zero), add(zero, suc(zero)))$ bzw. $add(add(suc(zero), zero), suc(zero))$.
- In jedem Fall ist aber ausgehend von diesen beiden Termen das Ergebnis immer der irreduzible Term $suc(suc(zero))$.
- Dieser repräsentiert die natürliche Zahl 2.

Konvergente Termersetzungssysteme

Beispiel 22 (Terminierung)

- Die Algebra der natürlichen Zahlen ist nicht nur konfluent, sondern auch terminierend und damit konvergent.
- Bei geeigneter Wahl der Gesetze hat jede natürliche Zahl die Normalform $suc^*(zero)$.
- Beispiel

$$
\begin{aligned}
2 * 2 = & \\
& = mult(suc(suc(zero)), suc(suc(zero))) \\
& = add(mult(suc(suc(zero)), suc(zero)), suc(suc(zero))) \\
& = add(suc(suc(zero)), suc(suc(zero))) \\
& = suc(add(suc(suc(zero), suc(zero)))) \\
& = suc(suc(add(suc(suc(zero)), zero))) \\
& = suc(suc(suc(suc(zero)))) = 4
\end{aligned}
$$

Konstruktoren und Projektoren

Definition 17 (Konstruktoren)

Operatoren, die als Ergebnis ein Element der definierten Sorte zurückliefern, werden **Konstruktoren** *genannt.*

Definition 18 (Projektoren)

Operatoren, die als Ergebnis ein Element einer anderen Sorte als der definierten Sorte zurückliefern, werden **Projektoren** *genannt.*

- Konstruktoren erzeugen Elemente der definierten Sorte.
- Projektoren liefern Informationen über Elemente der definierten Sorte.

Beispiel 23 (Projektor)

- In der Algebra der natürlichen Zahlen sind beispielsweise die Vergleichsoperationen wie $f_{kleiner}$ Projektoren, da diese als Ergebnis keine natürliche Zahl, sondern einen boolschen Wert liefern.

Aufteilung in Haupt- und Hilfskonstruktoren

- Die Konstruktoren werden in zwei disjunkte Teilmengen aufgeteilt.
- **Hauptkonstruktoren** sind die Konstruktoren, die in der Normalform der Terme vorkommen.
  - Hauptkonstruktoren werden also benötigt, um alle Elemente der definierten Sorte (in der Normalform) zu erzeugen.
  - Dies wird auch **Erzeugungsprinzip** genannt.
- **Hilfskonstruktoren** sind die Konstruktoren, die in der Normalform der Terme *nicht* vorkommen.
  - Hilfskonstruktoren sind für die Erzeugung der Elemente nicht notwendig und können durch Hauptkonstruktoren ersetzt werden.
  - Hierfür müssen die Hilfskonstruktoren auf der linken Seite geeigneter Gesetzen vorkommen.
- Im Folgenden wird der Begriff Konstruktor im Sinne von Hauptkonstruktor genutzt.

Konstruktoren: Beispiel

Beispiel 24 (Haupt- und Hilfskonstruktoren)

- Betrachten wir den Datentyp der natürlichen Zahlen, jetzt in der für Spezifikationssprachen üblichen Notation:

$$[M]$$

$$\begin{array}{|l} \mathit{zero} : M \\ \mathit{suc} : M \longrightarrow M \\ \mathit{add} : M \times M \longrightarrow M \\ \hline \mathit{add}(x, \mathit{zero}) = x \\ \mathit{add}(x, \mathit{suc}(y)) = \mathit{suc}(\mathit{add}(x, y)) \end{array}$$

- Die (Haupt-)Konstruktoren dieses Datentyps sind *zero* und *suc*. Mit ihnen lassen sich alle Elemente des Datentyps erzeugen.
- *add* ist lediglich ein Hilfskonstruktor, da *add* durch *zero* und *suc* ersetzt und so aus jedem Term eliminiert werden kann.

Vollständige Spezifikationen

Definition 19 (Hauptfunktor)

In einem Term $f(x, y, \ldots)$ ist f der **Hauptfunktor.**

Beispiel 25 (Hauptfunktor)

- Im Term $add(suc(zero), suc(zero))$ ist add der Hauptfunktor.

Definition 20 (Konstruktorterm)

Ein **Konstruktorterm** *ist ein Term, der ausschließlich Konstruktoren als Funktoren verwendet.*

Beispiel 26 (Konstruktorterm)

- Der Term $suc(suc(zero))$ ist ein Konstruktorterm, während der Term $add(zero, zero)$ kein Konstruktorterm ist.

Ausführbarkeit einer Spezifikation

Definition 21 (Ausführbare Spezifikation)

Ein Termersetzungssystem ist **ausführbar**, *wenn die folgenden Bedingungen erfüllt sind:*

1. *Alle Gesetze haben die Form* $T_1 = T_2$.
2. *Konstruktoren kommen nicht als Hauptfunktor auf der linken Seite von Gesetzen vor.*
3. *Alle Argumente des Hauptfunktors auf der linken Seite eines Gesetzes sind Konstruktorterme.*
4. *Die linke Seite eines Gesetzes darf nicht nur eine Variable sein.*
5. *Auf der rechten Seite eines Gesetzes dürfen nur Variablen vorkommen, die auch auf der linken Seite des Gesetzes vorkommen.*
6. *Auf jeden Term sollte nur ein Gesetz angewendet werden können.*

- Ein vollständiges und ausführbares Termersetzungssystem erfüllt die ursprünglichen Anforderungen an eine Spezifikation.

Ausführbarkeit einer Spezifikation

Beispiel 27 (Anforderungen an die Gesetze)

- Betrachtet wird der Datentyp der natürlichen Zahlen mit *zero* und *suc* als Konstruktor.
- Folgendes Gesetz erfüllt die Anforderungen, da $suc(y)$ ein Konstruktorterm ist:

$$add(x, suc(y)) = suc(add(x, y))$$

- Hier werden die Anforderungen *nicht* erfüllt, da $add(x, y)$ *kein* Konstruktorterm ist:

$$groesser(x, add(x, y)) = false$$

- Hier auch nicht, da das Gesetz *nicht* in der Form $T_1 = T_2$ vorliegt:

$$add(x, y) = x \Rightarrow y = zero$$

Ausführbarkeit einer Spezifikation

Beispiel 28 (Anforderungen an die Gesetze)

- Folgendes Gesetz erfüllt die Anforderungen nicht, da die linke Seite nur aus der Variablen x besteht:

$$x = add(x, zero).$$

- Hier werden die Anforderungen nicht erfüllt, da die Variable y nur auf der rechten, aber nicht auf der linken Seite vorkommt:

$$add(x, suc(zero)) = add(x, y).$$

Beispiel in Haskell: Boolesche Algebra

Beispiel 29 (Signatur und Gesetze)

```
module Bool where

data Boolean = Wahr | Falsch
               deriving Show

und       :: (Boolean, Boolean) -> Boolean
oder      :: (Boolean, Boolean) -> Boolean
nicht     :: Boolean -> Boolean

und(Wahr,x)     = x
und(Falsch,x)   = Falsch
oder(Falsch,x)  = x
oder(Wahr,x)    = Wahr
nicht(Wahr)     = Falsch
nicht(Falsch)   = Wahr
```

Beispiel in Haskell: Natürliche Zahlen

Beispiel 30 (Signatur)

```
module Nat where

data Nat = Zero | Suc Nat
     deriving Show

add  :: (Nat , Nat) -> Nat
sub  :: (Nat , Nat) -> Nat
mult :: (Nat , Nat) -> Nat
div  :: (Nat , Nat) -> Nat
pot  :: (Nat , Nat) -> Nat
fak  :: Nat -> Nat
```

Beispiel in Haskell: Natürliche Zahlen

Beispiel 31 (Gesetze)

```
add(x, Zero) = x
add(x, Suc(y)) =
    Suc(add(x, y))

sub(x, Zero) = x
sub(Suc(x),Suc(y)) =
    sub(x, y)

mult(x, Zero) = Zero
mult(x, Suc(y)) =
    add(x, mult(x, y))
```

```
div(Zero, x) = Zero
div(x, y) =
    add(Suc(Zero),
        div(sub(x, y), y))

pot(x, Zero) = Suc(Zero)
pot(x, Suc(y)) =
    mult(x, pot(x, y))

fak(Zero) = Suc(Zero)
fak(Suc(x)) =
    mult(Suc(x), fak(x))
```

Exemplarische Fragen zur Lernkontrolle

Beispiel 32

1. Worin unterscheiden sich ein abstrakter und ein konkreter Datentyp?
2. Was ist eine Signatur?
3. Was ist ein Term?
4. Worin unterscheiden sich die Präfix-, die Infix- und die Postfix-Notation von Termen?
5. Wie lassen sich Terme als Bäume darstellen?
6. Wie werden Gesetze auf Basis von Termen definiert?

Exemplarische Fragen zur Lernkontrolle

Beispiel 33

7. Was ist eine Algebra?
8. Was wird unter einer Interpretation eines abstrakten Datentyps in einer Algebra verstanden?
9. Wie wird nachgewiesen, dass eine Algebra ein Modell eines abstrakten Datentyps ist?
10. Was versteht man unter der abstrakten Algebra zu einer Algebra?
11. Worin besteht der Unterschied zwischen einem monomorphen und einem polymorphen Datentyp?

Exemplarische Fragen zur Lernkontrolle

Beispiel 34

12. Wann spricht man von Termreduktion?
13. Was ist ein Termersetzungsystem und wann ist es terminierend, konfluent oder konvergent?
14. Wann ist ein Term irreduzibel?
15. Worin unterscheiden sich Konstruktoren und Projektoren!
16. Worin unterscheiden sich Hauptkonstruktoren und Hilfskonstruktoren?
17. Erläutern Sie das Erzeugungsprinzip!
18. Was ist ein Konstruktorterm?
19. Wann ist ein Termersetzungsystem ausführbar?
20. Wie kann mittels eines vollständigen und ausführbaren Termersetzungssystems eine lauffähige Spezifikation erstellt werden?

Vielen Dank für Ihre Aufmerksamkeit!

Univ.-Prof. Dr.-Ing. Gero Mühl

gero.muehl@uni-rostock.de
https://www.ava.uni-rostock.de