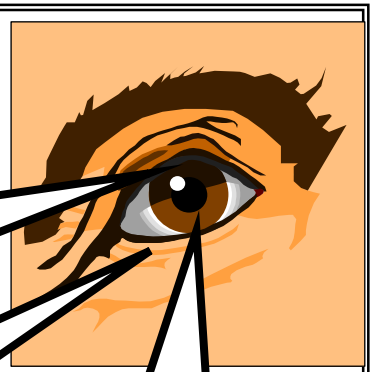
Syntax, Semantik, Spezifikation

Grundlagen der Informatik

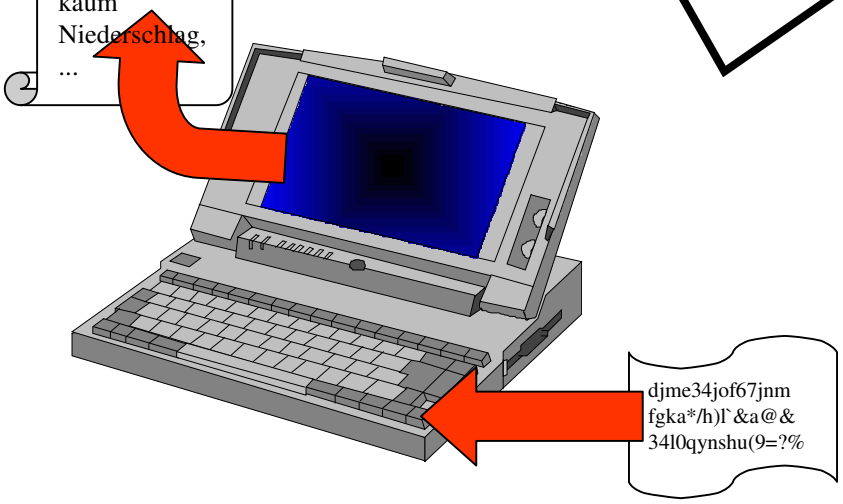
R. Hartwig

ProcedureHeading = PROCEDURE ident
[FormalParameters].
Block = {declaration}[BEGIN StatementSequence] END.
Declaration = CONST {ConstantDeclaration „;“} |
TYPE {TypeDeclaration „;“} |
VAR {VarableDeclaration „;“} | **SYNTAX**



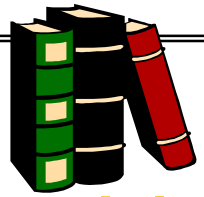
SEMANTIK

Das Wetter in Leipzig am 15. Oktober 2001: sonnig, z.t. bewölkt, 19°C, kaum Niederschlag, ...




```
// klasse1.cpp -- eine Klasse benutzen
# include <iostream.h>
// Klassendefinition
class softball
{
private:
    char vorname[15];
    char[nachname[15];
    unsigned schlaege;
    unsigned treffer;
    unsigned rbis;
    float durchs;
    float berechne_durchs();
public:
    void alles_setzen();
    void aktual();
    void zeige_stat();
};
```

SPEZIFIKATION



Literatur

Ehrich, H.-D., M. Gogolla, U. W. Lipeck: ***Algebraische Spezifikation abstrakter Datentypen*** . B. G. Teubner Stuttgart, 1989

Ehrig, H., B. Mahr: ***Fundamentals of Algebraic Specification*** . Springer-Verlag, 1985

Grätzer, G.: ***Universal Algebra***. Springer-Verlag, 1979

Klaeren, H.A.: ***Algebraische Spezifikation***. Springer-Verlag, 1983

Loeckx, J., H.-D. Ehrich, M. Wolf: ***Specification of Abstract Data Types***. John Wiley & Sons and B. G. Teubner, 1996

Lugowski, H.: ***Grundzüge der Universellen Algebra***. Teubner-Verlag, 1976

Vorlesungsskripten zu „***Algebraische Grundlagen der Informatik***“ und „***Algebraische Spezifikation***“

Inhalt

1. Einführung
2. Heterogene Algebren
3. Algebraische Hülle und Homomorphie
4. Termalgebren
5. Syntax und Semantik
6. Kongruenzen
7. Gleichungskalkül
8. Abstrakte Datentypen
9. Spezifikationen
10. Initialität
11. Gleichungskalkül und Induktion
12. Erweiterung von Gleichungsspezifikationen
13. Finale Semantik und beobachtbares Verhalten
14. Implementation von Spezifikationen

Kapitel 1

Einführung

Guten Tag,
darf ich Ihnen mal eine
Frage stellen?

Wat jibt´s denn?
Mit Computa kenn´
ick mir aus!

Was ist ein „Datentyp“?

integers,
reals, arrays,
records, stacks,
files, characters,
...

Definition erforderlich!

Beachte:
Nutzerdefinierte Datentypen
in vielen Programmiersprachen

„Ein Datentyp legt die Menge der Werte fest, die eine Variable annehmen kann."

(*Pascal* User Manual)

?

stacks* = *queues

alles

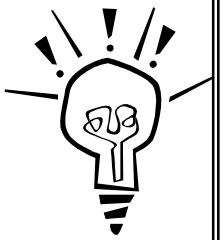
sequences

Na ja, na ja, ...

Zugriffsmechanismen:

filo vs. ***fifo***

Datentypen sind Algebren ?



Datentyp:

- Menge von Objekten
- Operationen über diesen Objekten

= Algebra

Algebra und Informatik

Mathematik	Informatik
Algebra	z.B.: `real array` benötigt
= universelle Algebra	`real`, `array`, `integer`
= algebraische Struktur	----------
----------	Algebra
• Trägermenge	=> Begriff erweitern !
• Operationenfamilie	----------
	• mehrere Trägermengen
	• Operationenfamilie

Algebra und Informatik

Homogene oder einsortige Algebra

- Gruppe
- Monoid
- Ring
- Boolesche Algebra
- Verband

Heterogene oder mehrsortige Algebra

- Datentyp
- Vektorraum
- endlicher Automat
- Syntax **einer Programmiersprache**
- Semantik - “ -

Algebra und Informatik

Problem:

Partialität vieler Operationen:

Selektion, Division, Überlauf/Unterlauf,

Lesen aus leerem Keller,

Aufruf nichtdeklarierter Prozedur

mehrsortige, partielle Algebren

Theorie kompliziert

partielle Operation ⟹ `error` totale Operation

Beispiel

Datentyp „Zeichenkeller“ (stacks of characters):

- (mindestens) drei Trägermengen

  Char = {a, b, c, ..., z}

  Stack = Char *

  Bool = {**`true`**, **`false`**}

- vier Operationen zweckmäßig

  `push` : Char × Stack → Stack

  `top` : Stack → Char

  `pop` : Stack → Stack

  `empty?` : Stack → Bool

Beispiel (Forts.)

Definition der Operationen:

Sei ε - leerer Keller

sx - Füllung (Verkettung) des Kellers *s* mit Zeichen *x*

$$
\begin{aligned}
push(x,s) &= sx \\
top(sx) &= x \\
pop(sx) &= s \\
empty?(s) &= \begin{cases} true & , \text{falls } s = \varepsilon \\ false & , \text{falls } s \neq \varepsilon \end{cases} \\
top(\varepsilon) &= ? \\
pop(\varepsilon) &= ?
\end{aligned}
$$

$\Rightarrow$ Einführung von error-Elementen

Beispiel (Forts.)

Definition der Operationen:

Sei ε - leerer Keller

sx - Füllung (Verkettung) des Kellers *s* mit Zeichen *x*

```
push(x,s) = sx
top(sx)   = x
pop(sx)   = s

top(ε)    =  char-err
pop(ε)    =  stack-err

              { true     , falls s = ε
empty?(s) = { bool-err, falls s = stack-err
              { false    , sonst
```

Beispiel (Forts.)

Trägermengen erweitern:

$\underline{\text{Char}}_p = \{a, b, c, ..., z\}$

$\underline{\text{Char}} = \underline{\text{Char}}_p \cup \{\text{char-err}\}$

$\underline{\text{Stack}} = \underline{\text{Char}}_p^* \cup \{\text{stack-err}\}$

$\underline{\text{Bool}} = \{\texttt{true}, \texttt{false}\} \cup \{\text{bool-err}\}$

Operationen - „strikt“ vervollständigen

$\texttt{push} : \underline{\text{Char}} \times \underline{\text{Stack}} \rightarrow \underline{\text{Stack}}$

$\texttt{top} : \underline{\text{Stack}} \rightarrow \underline{\text{Char}}$

$\texttt{pop} : \underline{\text{Stack}} \rightarrow \underline{\text{Stack}}$

$\texttt{empty?} : \underline{\text{Stack}} \rightarrow \underline{\text{Bool}}$

Beispiel (Forts.)

- Erweiterung um Boolesche Operationen oft nützlich:

  vel : $\underline{\text{Bool}} \times \underline{\text{Bool}} \to \underline{\text{Bool}}$

  et : $\underline{\text{Bool}} \times \underline{\text{Bool}} \to \underline{\text{Bool}}$

  non : $\underline{\text{Bool}} \to \underline{\text{Bool}}$

  (alle strikt vervollständigt)

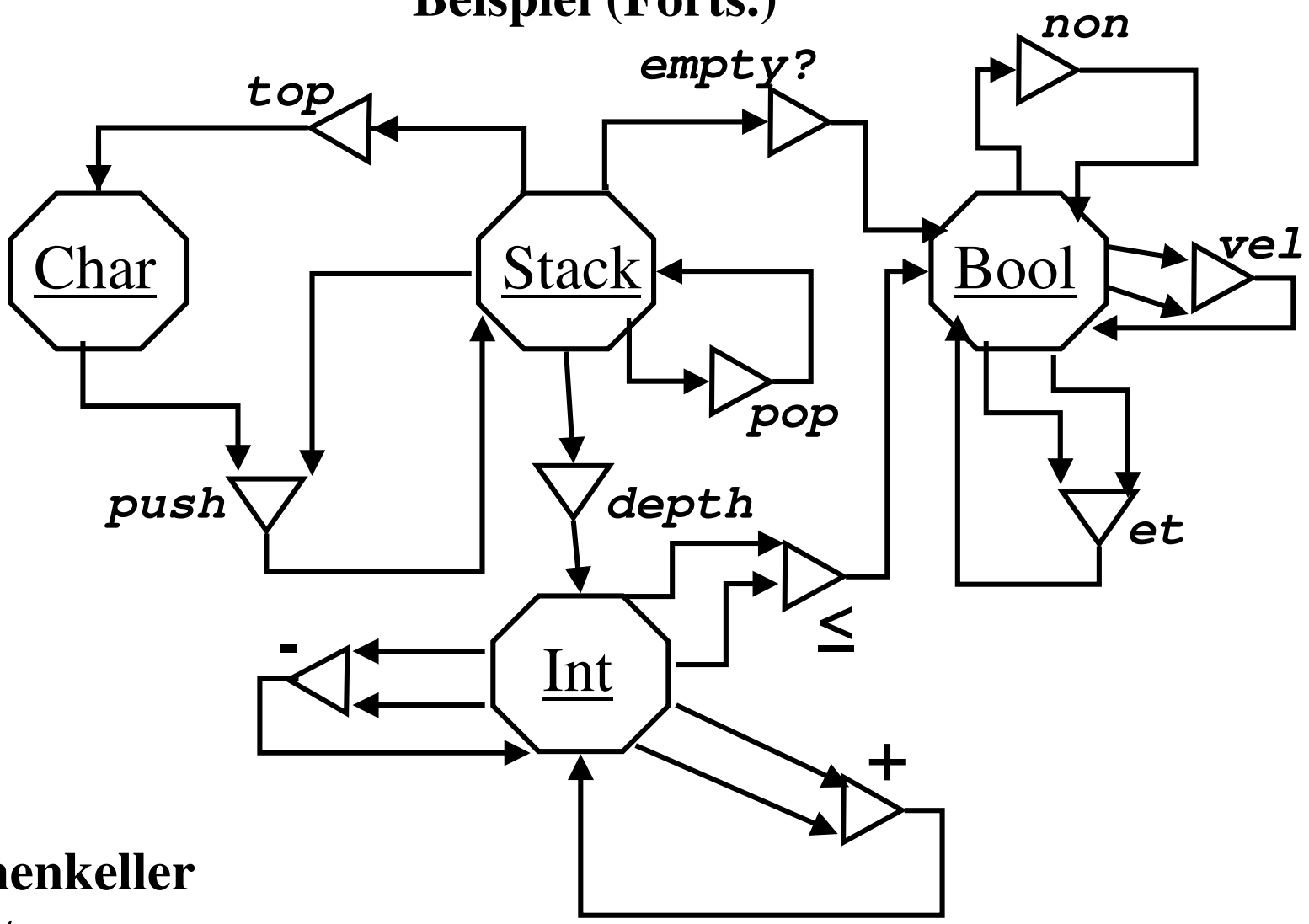
- Ergänzung des Datentyps „Zeichenkeller“ um neue Sorte $\underline{\text{Int}}$ möglich, um z.B. die „Kellertiefe“ zu erfassen:

  depth : $\underline{\text{Stack}} \to \underline{\text{Int}}$

- Dann evtl. zweckmäßig, Rechnen über $\underline{\text{Int}}$ hinzuzunehmen:
  arithmetische (**+, -**) und Vergleichsoperationen $\underline{<}$ usw.

⇨ Algebra mit vier Trägermengen und Operationen zwischen ihnen

Beispiel (Forts.)



Zeichenkeller

Signatur

Heterogene Algebren

Signatur Σ : $\Sigma = (S, \Omega, \alpha)$

Mengen

$S \neq \emptyset, S \cap \Omega = \emptyset$

$\alpha : \Omega \rightarrow S^+$

S heißt Menge der ***Sorten****(namen)*,

Ω Menge der ***Operatoren***,

α ***Sortigkeit*** oder ***Arität***

Signatur Σ , $\Sigma = (S, \Omega, \alpha)$, $\omega \in \Omega$ gegeben.

$\Rightarrow$ $\alpha(\omega) \in S^+$

$\alpha(\omega) = (s_1, s_2, \ldots, s_n, s_{n+1})$

$w = (s_1, s_2, \ldots, s_n)$ heißt ***Eingang*** des Operators ω,
kurz $i_\alpha(\omega)$.

s_{n+1} heißt ***Ausgang*** des Operators ω,
kurz $o_\alpha(\omega)$.

n heißt ***Stelligkeit*** des Operators ω.

$\Rightarrow i_\alpha(\omega) \in S^*$, $o_\alpha(\omega) \in S$,

$\alpha(\omega) = (i_\alpha(\omega), o_\alpha(\omega)) = (w, s)$

$\alpha : \Omega \rightarrow S^* \times S$

Bequeme Schreibweisen:

$A = (A_s)_{s \in S}$ und $B = (B_s)_{s \in S}$ seien Mengenfamilien.
A ***Teilfamilie*** von B, $A \subseteq B$, gdw. $A_s \subseteq B_s$ für alle $s \in S$.

Analog wird *Durchschnitt* und *Vereinigung* von Mengenfamilien definiert.
$a \in A$ ist Abkürzung für $\exists s(s \in S \wedge a \in A_s)$.

<u>*Verallgemeinerung des Kreuzprodukts:*</u>
für $w \in S^*$

$$A^w = \begin{cases} A_{s_1} \times A_{s_2} \times \ldots \times A_{s_n}, & \text{falls } w = (s_1, s_2, \ldots, s_n) \\ \{\varepsilon\}, & \text{falls } w = \varepsilon \end{cases}$$

Bequeme Schreibweisen (Forts.):

Analog zu A^w für $w \in S^*$ wird auch für Abbildungsfamilien $h = (h_s)_{s \in S}$ die Schreibweise h^w eingeführt:

Wenn

$$h : A \to B\,, \quad (h_s : A_s \to B_s)_{s \in S}\,,$$

dann bei $w = (s_1, s_2, \ldots, s_n)$:

$$h^w : A^w \to B^w$$

vermöge

$$h^w(a_1, a_2, \ldots, a_n) = (h_{s_1}(a_1),\ h_{s_2}(a_2), \ldots,\ h_{s_n}(a_n))$$

Signatur $\Sigma = (S, \Omega, \alpha)$ gegeben.

Bilde für alle $w \in S^*$, $s \in S$ die Mengen

$$\Sigma_{w,s} = \{\omega \mid \omega \in \Omega \wedge \alpha(\omega) = (w, s)\}$$

Man beachte, daß bei Anwendungen fast alle $\Sigma_{w,s}$ leer sind.

Manchmal heißt dann auch die Familie

$$\Sigma = (\Sigma_{w,s})_{w \in S^*, s \in S}$$

eine ***S-sortige Signatur.***

Die Elemente von S sind wieder die ***Sorten***, die von $\Sigma_{w,s}$ ***Operatoren*** mit ***Eingang*** w, ***Ausgang*** s und ***Arität*** (w,s).

$\Rightarrow \alpha(\omega) = (w, s)$ ist gleichbedeutend mit $\omega \in \Sigma_{w,s}$

$\omega \in \Sigma$ steht als Abkürzung für

$$\exists s \exists w (s \in S \wedge w \in S^* \wedge \omega \in \Sigma_{w,s}).$$

Zum Begriff der „heterogenen Algebra“:

Signatur Σ mit $\Sigma = (S, \Omega, \alpha)$ gegeben.

Heterogene Algebra mit Signatur Σ,
kurz ***Σ-Algebra***:

$$\mathcal{A} = [(A_s)_{s \in S}, (f_\omega)_{\omega \in \Omega}]$$

Mengenfamilie — Funktionenfamilie

mit: wenn $\alpha(\omega) = (w, s)$, so

$$f_\omega : A^w \rightarrow A_s$$

Σ-Algebra $\mathcal{A} = [(A_s)_{s \in S}, (f_\omega)_{\omega \in \Omega}]$ gegeben.

A_s heißen ***Trägermengen*** von $\mathcal{A}$,
Trägermenge der Sorte s,
paarweise Disjunktheit ist nicht (!) gefordert,
f_ω heißen ***Operationen*** von $\mathcal{A}$.
Wenn $\alpha(\omega) = (s)$, d.h. $i_\alpha(\omega) = \varepsilon$, so heißt
f_ω ***Konstante der Sorte*** s.

$\mathcal{A} = [(A_s)_{s \in S}, (f_\omega)_{\omega \in \Omega}]$ Σ-Algebra

$\Rightarrow \quad f_\omega : A^{i\alpha(\omega)} \rightarrow A_{o\alpha(\omega)}$

Homogene Algebra

$$\mathcal{A} = [\, A \;, (f_\omega)_{\,\omega \in \Omega}]$$

ist Spezialfall der heterogenen Algebren,
und zwar mit einsortiger Signatur!

Wenn bei $\Sigma = (S, \Omega, \alpha)$ die Sortenmenge einelementig ist, $S = \{s\}$, so wird Familie $(A_s)_{s \in S}$ zu einer einzigen Trägermenge A,
die Arität $\alpha(\omega) = (s, s, ..., s)$ gibt nur noch die Stellenzahl an, also äquivalent zu $\tau : \Omega \rightarrow \underline{\mathrm{Nz}}$.
$\tau : \Omega \rightarrow \underline{\mathrm{Nz}}$ beinhaltet alle notwendigen „Signatur“-Informationen und ersetzt damit Σ.

τ heißt ***Typ*** der homogenen Algebra $\mathcal{A}$

Beispiele für Algebren

1. MEALY-Automat:

$$\mathcal{A} = [\, X,\ Y,\ Z\ ;\ \delta,\ \lambda\,]$$

wie üblich definierter endlicher Automat:

X - Eingabealphabet,

Y - Ausgabealphabet,

Z - Zustandsmenge,

$\delta : Z \times X \to Z$ Überführungsfunktion,

$\lambda : Z \times X \to Y$ Ausgabefunktion.

$\mathcal{A}$ ist eine Σ-Algebra $\mathcal{A} = [(A_s)_{s \in S},\ (f_\omega)_{\ \omega \in \Omega}]$ mit der Signatur $\Sigma = (S, \Omega, \alpha)$ mit z.B.

$$S = \{x, y, z\},\ \Omega = \{1, 2\},$$

$$\alpha(1) = (z, x, z),\ \alpha(2) = (z, x, y).$$

Dann ist $A_x = X,\ A_y = Y,\ A_z = Z$ und $f_1 = \delta,\ f_2 = \lambda.$

Beispiele für Algebren

2. MOORE-Automat:

$$\mathcal{A} = [\, X,\ Y,\ Z\ ;\ \delta,\ \mu\,]$$

kann analog als eine Σ-Algebra

$$\mathcal{A} = [(A_s)_{s \in S},\ (f_\omega)_{\ \omega \in\ \Omega}]$$

mit z.B. der Signatur

$$\Sigma = (S,\ \Omega,\ \alpha\ ) \quad \text{mit}$$

$$S = \{x,\ y,\ z\}, \quad \Omega = \{\, \delta,\ \mu\, \},$$

$$\alpha\,(\delta) = (z,\ x\ ,\ z), \quad \alpha\,(\mu) = (z,\ y)$$

absolut willkürlich !

aufgefaßt werden.

Beachte hier: $f_\delta = \delta,\ f_\mu = \mu$

Beispiele für Algebren

3. ROBIN-SCOTT-Automat (initialer Automat):

$$\mathcal{A} = [\, Z, X, z_0, F, \delta \,]$$

mit $\delta : Z \times X \to Z,\ z_0 \in Z,\ F \subseteq Z$

kann als eine Σ-Algebra $\mathcal{A} = [(A_s)_{s \in S}, (f_\omega)_{\omega \in \Omega}]$ aufgefaßt werden mit $S = \{z, x\}$, $\Omega = \{\omega_0, \omega_\delta\} \cup \{\omega_t \mid t \in F\}$,

$\alpha(\omega_0) = (z)$, $\alpha(\omega_t) = (z)$ für alle $t \in F$, $\alpha(\omega_\delta) = (z,x,z)$.

$\Rightarrow$ $A_z = Z$, $A_x = X$ und

$f_{\omega_0} = z_0$, $f_{\omega_t} = t$ für alle $t \in F$, (Initial-, Finalzustände)

Konstanten !

$f_{\omega_\delta} = \delta$ (Überführungsfunktion)

Beispiele für Algebren

4. Homogene Algebren:

- $[\ \wp(M)\ ;\ \cup\ ,\ \cap\ ]$

  Potenzmenge über einer Menge M mit Vereinigung und Durchschnitt

- $[\ \{W, F\}\ ;\ \wedge\ ,\ \vee\ ]$

  Menge der Wahrheitswerte mit Konjunktion und Disjunktion

- $[\ \underline{\mathbf{Nz}}\ ;\ +\ ,\ *\ ]$

  Menge der natürlichen Zahlen mit Addition und Multiplikation

- alle drei Algebren vom **Typ (2,2).**

Sie heißen deshalb *ähnlich* (**!**).

Beispiele für Algebren

weiter zu
4. Homogene Algebren:

- [**Nz** ; 0 , ']
  Menge der natürlichen Zahlen mit der Konstanten **0**
  und der Nachfolgerfunktion **'**

- homogene Algebra vom **Typ (0,1).**
Das ist die allgemein bekannte PEANO-Algebra der natürlichen Zahlen.

Beispiele für Algebren

5. Aussagenlogische Ausdrucks-Algebra:

- [**AUS** ; *neg, con, alt*]

Menge aller „pfeilfreien“ aussagenlogischen Ausdrücke mit *neg* als der syntaktischen Operation der Bildung negierter Ausdrücke, *con* bzw. *alt* den entsprechenden syntaktischen Operationen der Bildung von Konjunktionen bzw. Alternativen.

Für $A, B \in$ **AUS** gilt z.B.

$$neg(A) = \neg A$$
$$con(A, B) = (A \wedge B)$$
$$alt(A, B) = (A \vee B)$$

- homogene Algebra vom **Typ (1, 2, 2).**
- **bemerkenswerte Eigenschaften!**

Beispiele für Algebren

6. Semiotisches Quadrupel:

- $\underline{\boldsymbol{S}} = [\, W, X ;\ \Upsilon , \bullet \,]$

X ist ein Alphabet, W ist die Wortmenge über diesem Alphabet, Υ das leere Wort, und $\bullet$ die Verkettung eines Wortes mit einem Zeichen. (!)

$\underline{\boldsymbol{S}}$ ist heterogene Algebra mit der Signatur $\Sigma = (S, \Omega, \alpha)$ mit

$S = \{w, z\}, \quad \Omega = \{\varepsilon, \kappa\},$

$\alpha(\varepsilon) = (w), \quad \alpha(\kappa) = (w,z,w).$

$\Rightarrow \quad A_w = W, \ A_z = X$ und

$f_\varepsilon = \Upsilon , \qquad f_\kappa = \bullet : W \times X \mapsto W$

(Υ Konstante der Sorte w)

Beispiele für Algebren

7. Syntax einer Programmiersprache
   kann man stets als heterogene Algebra auffassen!

Einfache Beispielsprache:

- <Ide> - Menge der ***Identifier*** sei beliebig gegeben
- ***Ausdrücke*** (expressions)

  <Exp> ::= 0 | 1 | *true* | *false* | <Ide> | ¬ <Exp> |
  (<Exp> = <Exp>) | (<Exp> + <Exp>)
- ***Anweisungen*** (commands)

  <Com> ::= <Ide> := <Exp> | <Com> ; <Com> |
  if <Exp> *then* <Com> *else* <Com> *fi*

7. Syntax einer Programmiersprache (Forts.):

Man erhält drei Trägermengen

Ide, **Exp**, **Com** .

Zu jeder Einzelregel obiger BNF gehört eine Operation der Algebra, hinzu kommt für jeden Identifier eine Konstante:

Beispiel: Operationen, die sich aus Regeln für <Com> ergeben:

$$f_{:=}(I, E) = I := E$$
$$f_{if}(E, C_1, C_2) = \underline{if}\ E\ \underline{then}\ C_1\ \underline{else}\ C_2\ \underline{fi}$$
$$f_{;}(C_1, C_2) = C_1;C_2$$

7. Syntax einer Programmiersprache (Forts.):

Die Operationen der Algebra lassen sich aus der BNF ablesen:

$f_0 = \underline{0}$

$f_1 = \underline{1}$

$f_{true} = \underline{true}$

$f_{false} = \underline{false}$

} ***Konstanten!***

$f_I = I$ für alle $I \in$ **Ide**

$f_{\neg}(E) = \neg E$ für $E \in$ **Exp**

$f_{=}(E_1, E_2) = (E_1 = E_2)$ für $E_1, E_2 \in$ **Exp**

$f_{+}(E_1, E_2) = (E_1 + E_2)$ für $E_1, E_2 \in$ **Exp**

$f_{:=}(I, E) = I := E$ für $I \in$ **Ide**, $E \in$ **Exp**

$f_{if}(E, C_1, C_2) =$ $\underline{if}\ E\ \underline{then}\ C_1\ \underline{else}\ C_2\ \underline{fi}$

für $E \in$ **Exp**, $C_1, C_2 \in$ **Com**

$f_{;}(C_1, C_2) = C_1;C_2$ für $C_1, C_2 \in$ **Com**

7. Syntax einer Programmiersprache (Forts.):

Die Signatur dieser Algebra kann man auch leicht aus folgenden Angaben ablesen:

$0 : \mapsto \mathbf{Exp}$

$1 : \mapsto \mathbf{Exp}$

$\mathit{true} : \mapsto \mathbf{Exp}$

$\mathit{false} : \mapsto \mathbf{Exp}$

$I : \mapsto \mathbf{Exp}$ für alle $I \in \mathbf{Ide}$

$\neg : \mathbf{Exp} \mapsto \mathbf{Exp}$

$= : \mathbf{Exp} \times \mathbf{Exp} \mapsto \mathbf{Exp}$

$+ : \mathbf{Exp} \times \mathbf{Exp} \mapsto \mathbf{Exp}$

$:= : \mathbf{Ide} \times \mathbf{Exp} \mapsto \mathbf{Com}$

$\mathit{if} : \mathbf{Exp} \times \mathbf{Com} \times \mathbf{Com} \mapsto \mathbf{Com}$

$; : \mathbf{Com} \times \mathbf{Com} \mapsto \mathbf{Com}$

Algebraische Hülle und Homomorphie

$\mathcal{A} = [A , F]$ sei Σ-Algebra.

Eine Teilfamilie $B \subseteq A$ heißt ***abgeschlossen bezüglich*** der Operation f_ω aus F gdw. mit

$$(a_1, a_2 , ..., a_n) \in B^{i\alpha(\omega)}$$

stets auch

$$f_\omega (a_1, a_2 , ..., a_n) \in B_{o\alpha(\omega)}$$

gilt.

B heißt ***abgeschlossen gegenüber*** F bzw. ***abgeschlossen in*** $\mathcal{A}$ genau dann, wenn B abgeschlossen bezüglich jeder Operation f_ω aus F ist.

$\mathcal{A} = [A, F]$ sei Σ-Algebra.
Eine Teilfamilie $B \subseteq A$ heißt ***algebraische Hülle*** der Teilfamilie $C \subseteq A$, kurz

$$B = [C]_{\mathcal{A}} \quad \text{oder} \quad B = [C]$$

gdw.

$B = \bigcap \{ A' \mid C \subseteq A' \subseteq A$ und A' abgeschlossen in $\mathcal{A} \}$.

- Die algebraische Hülle von C ist die kleinste abgeschlossene Teilfamilie, die C umfaßt.
- Ist C abgeschlossen gegenüber F, so ist $[C]_{\mathcal{A}} = C$

Für die algebraische Hülle gelten die **Hülleneigenschaften**:

- $B \subseteq [B]_{\mathcal{A}}$ *Einbettung*
- Wenn $B \subseteq C$, so $[B]_{\mathcal{A}} \subseteq [C]_{\mathcal{A}}$. *Monotonie*
- $[\,[B]_{\mathcal{A}}\,]_{\mathcal{A}} \subseteq [B]_{\mathcal{A}}$ *Abgeschlossenheit*

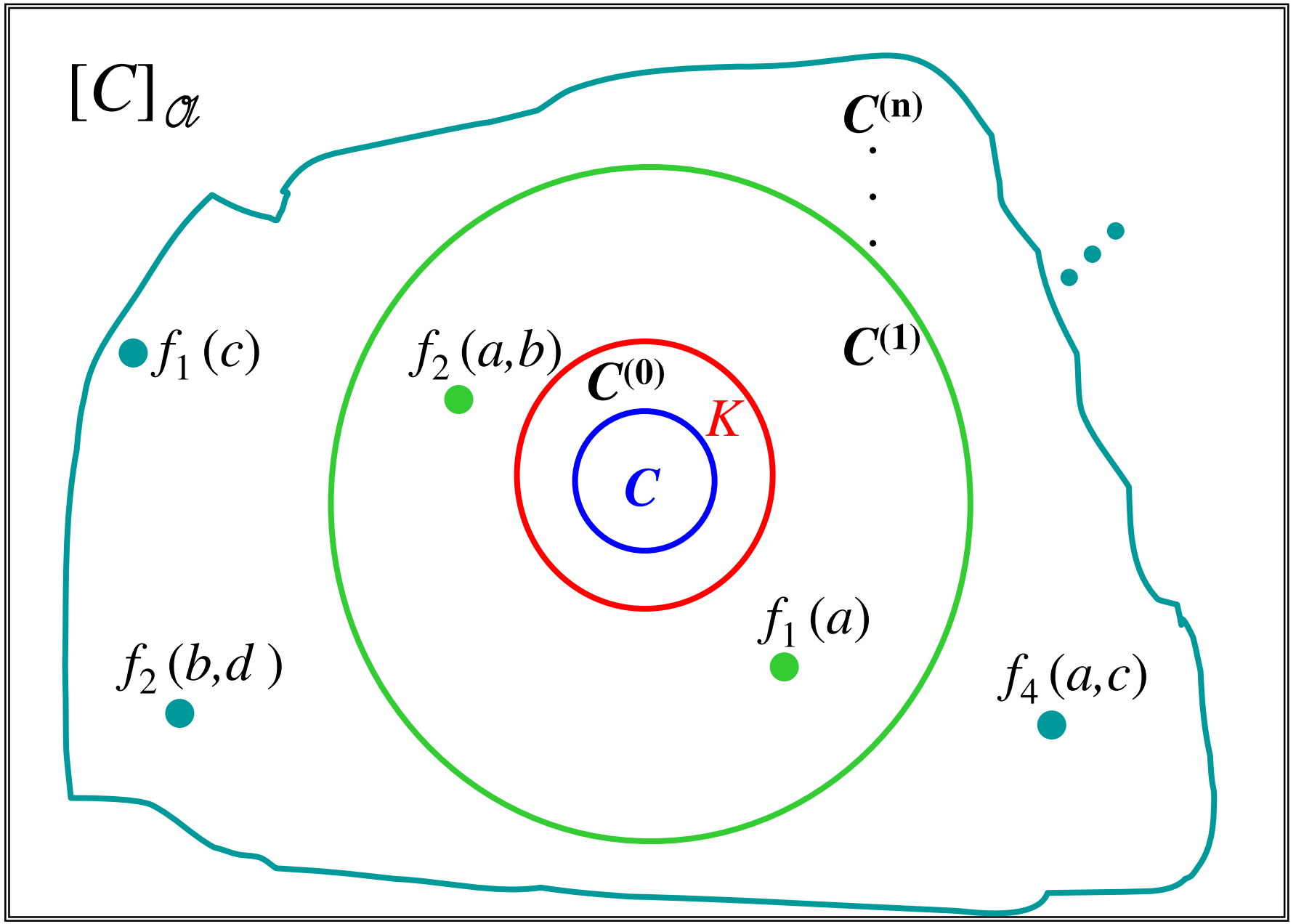
Für eine Σ-Algebra $\mathcal{A} = [A, F] = [(A_s)_{s \in S}, (f_\omega)_{\omega \in \Omega}]$
bezeichne im folgenden $K = (K_s)_{s \in S}$ mit

$$K_s = \{f_\omega \mid \omega \in \Omega \wedge \alpha(\omega) = (s)\}$$

die Familie der Konstanten der Sorte s.
$\underline{\varnothing} = (\varnothing)_{s \in S}$ bezeichne die S-sortige Familie leerer Mengen.

Hülle von $\varnothing$:

- $[\underline{\varnothing}]_{\mathcal{A}} = [K]_{\mathcal{A}}$
- $[\underline{\varnothing}]_{\mathcal{A}} = \underline{\varnothing}$ gdw. es in Ω keinen nullstelligen Operator gibt.


$[C]_{\mathcal{A}}$
$C^{(n)}$
$C^{(1)}$
$C^{(0)}$
K
C
$f_1(c)$
$f_2(a,b)$
$f_1(a)$
$f_2(b,d)$
$f_4(a,c)$

$\mathcal{A} = [A, F]$ sei Σ-Algebra.
Die Anwendung des von $F = (f_\omega)_{\omega \in \Omega}$ erzeugten
BAIRE- ***Operators*** $\underline{F}$ auf die Teilfamilie $B \subseteq A$
ist definiert als

$$\underline{F}(B) = \left(\{ f_\omega(b_1, b_2, \ldots, b_n) \mid \omega \in \Omega \wedge \alpha(\omega) = (w, s) \wedge w \neq \varepsilon \wedge (b_1, b_2, \ldots, b_n) \in B^w \} \right)_{s \in S}$$

• Der BAIRE-Operator beschreibt also die einmalige Anwendung aller in F vorkommenden Operationen auf alle in deren Vorbereich liegenden Argumentfolgen, die mit den Elementen der Familie B gebildet werden können.

Darstellung der algebraischen Hülle $[C]_{\mathcal{A}}$:

Die algebraische Hülle kann induktiv wie folgt erzeugt werden:

- $C^{(0)} = C \cup K$
- $C^{(i+1)} = C^{(i)} \cup \underline{F}(C^{(i)})$ für alle $i \geq 0$
- $[C]_{\mathcal{A}} = \bigcup_{i \in Nz} C^{(i)}$

Beweisidee:

Setze $B = \bigcup_{i \in Nz} C^{(i)}$. $\Rightarrow$ $C \subseteq B \subseteq A$

- Zeige B ist abgeschlossen in $\mathcal{A}$. $\Rightarrow$ $[C]_{\mathcal{A}} \subseteq B$
- Zeige induktiv für alle i: $C^{(i)} \subseteq [C]_{\mathcal{A}}$. $\Rightarrow$ $B \subseteq [C]_{\mathcal{A}}$

$\mathcal{A} = [A, (f_\omega)_{\omega \in \Omega}]$ und $\mathcal{L} = [B, (f'_\omega)_{\omega \in \Omega}]$ seien zwei Σ-Algebren.

Dann heißt $\mathcal{L}$ eine ***Unteralgebra*** von $\mathcal{A}$ gdw.

$B \subseteq A$ und für alle $(b_1, b_2, ..., b_n) \in B^w$ gilt:

$$f'_\omega(b_1, b_2, ..., b_n) = f_\omega(b_1, b_2, ..., b_n) .$$

1. $\mathcal{A}$ sei Σ-Algebra. Dann ist $\mathcal{L} = [B, (f'_\omega)_{\omega \in \Omega}]$ Unteralgebra von $\mathcal{A}$ gdw.

(a) $B \subseteq A$,

(b) für alle $\omega \in \Omega$ ist $f'_\omega = f_\omega |_{B^{i\alpha(\omega)}}$, *(Einschränkung)*

(c) B ist abgeschlossen gegenüber $(f'_\omega)_{\omega \in \Omega}$.

2. Zu jedem $B \subseteq A$ gibt es damit höchstens eine Unteralgebra $\mathcal{L}$ von $\mathcal{A}$.

Hülle und Unteralgebra :

Es sei $\mathcal{A}$ eine Σ-Algebra und $X \subseteq A$. Dann ist $\mathcal{L} = [\, [X]_{\mathcal{A}} \,,\, (f_\omega)_{\omega \in \Omega}\,]$ Unteralgebra von $\mathcal{A}$.

$\mathcal{L} = [\, [X]_{\mathcal{A}} \,,\, (f_\omega)_{\omega \in \Omega}\,]$ heißt die ***von der Familie X in $\mathcal{A}$ erzeugte Unteralgebra***; die Mengenfamilie X heißt ein ***Erzeugendensystem*** von $\mathcal{L}$.

1. X ist genau dann ein Erzeugendensystem der Algebra $\mathcal{A} = [A\,,\,F\,]$, wenn $[X]_{\mathcal{A}} = A$ ist .

2. Eine Algebra kann auch ein leeres Erzeugendensystem besitzen.

Beispiele für Erzeugendensysteme

1. $\mathcal{A}$ = [**AUS** ; *neg, con, alt*] - die Algebra der pfeilfreien aussagenlogischen Ausdrücke besitzt als Erzeugendensystem die Menge *V* der aussagenlogischen Variablen, denn es ist

$$[V]_{\mathcal{A}} = \mathbf{AUS}$$

   - vergleiche dazu die Konstruktion der algebraischen Hülle einerseits und die aus der Logik bekannte Ausdrucksdefinition

2. $\mathcal{G}$ = [**G** ; *0*, **+**] - die Algebra der ganzen Zahlen mit der Konstanten 0 und der Addition **+** besitzt als Erzeugendensystem { -1, 1} (und jede umfassende Menge).

Beispiele für Erzeugendensysteme (Forts.)

noch zu 2. $\mathcal{G} = [\ \underline{\mathbf{G}}\ ;\ 0,\ +\ ]$:

- zum Begriff der Unteralgebra betrachte man die Algebra $\mathcal{N}$ der natürlichen Zahlen $\mathcal{N} = [\ \underline{\mathbf{Nz}}\ ;\ 0,\ +\ ]$ wieder mit 0 und der Addition **+** .

  $\mathcal{N}$ ist selbstverständlich eine Unteralgebra von $\mathcal{G}$.

  $\mathcal{G}$ ist bekanntermaßen eine Gruppe, d.h. aber nicht, daß die Unteralgebra $\mathcal{N}$ eine Untergruppe von $\mathcal{G}$ ist!

3. Suche ein minimales Erzeugendensystem für das "Semiotische Quadrupel" $\underline{\boldsymbol{S}} = [\ W, X\ ;\ \Upsilon\ ,\ \bullet\ ]$!

   Zur Definition vgl. das Beispiel 6. für Algebren.

Prinzip der algebraischen Induktion :

Es sei $\mathcal{A} = [A , F]$ eine Σ-Algebra und X ein Erzeugendensystem von $\mathcal{A}$. Wenn dann für eine Teilfamilie B von A folgendes gilt:

- $X \subseteq B$,
- B ist abgeschlossen gegenüber F,

dann ist $B = A$.

Der Beweis ist eine einfache ÜA.

Beweisverfahren (algebraische Induktion):

$H(a)$ sei für alle $a \in A$ zu zeigen.

X sei Erzeugendensystem von $\mathcal{A}$.

2 Beweisschritte:

(IA) Zeige $H(x)$ für alle $x \in X$.

(IS) Zeige für beliebige $\omega \in \Omega$ (es sei $\alpha(\omega) = (w, s)$)

und beliebige $(a_1, a_2, ..., a_n) \in A^w$:

Wenn $H(a_1) \wedge H(a_2) \wedge ... \wedge H(a_n)$,

so auch $H(f_\omega(a_1, a_2, ..., a_n))$.

Für die Rechtfertigung betrachte im letzten Satz $B = (B_s)_{s \in S}$ mit

$$B_s = \{ a \mid a \in A_s \wedge H(a) \} .$$

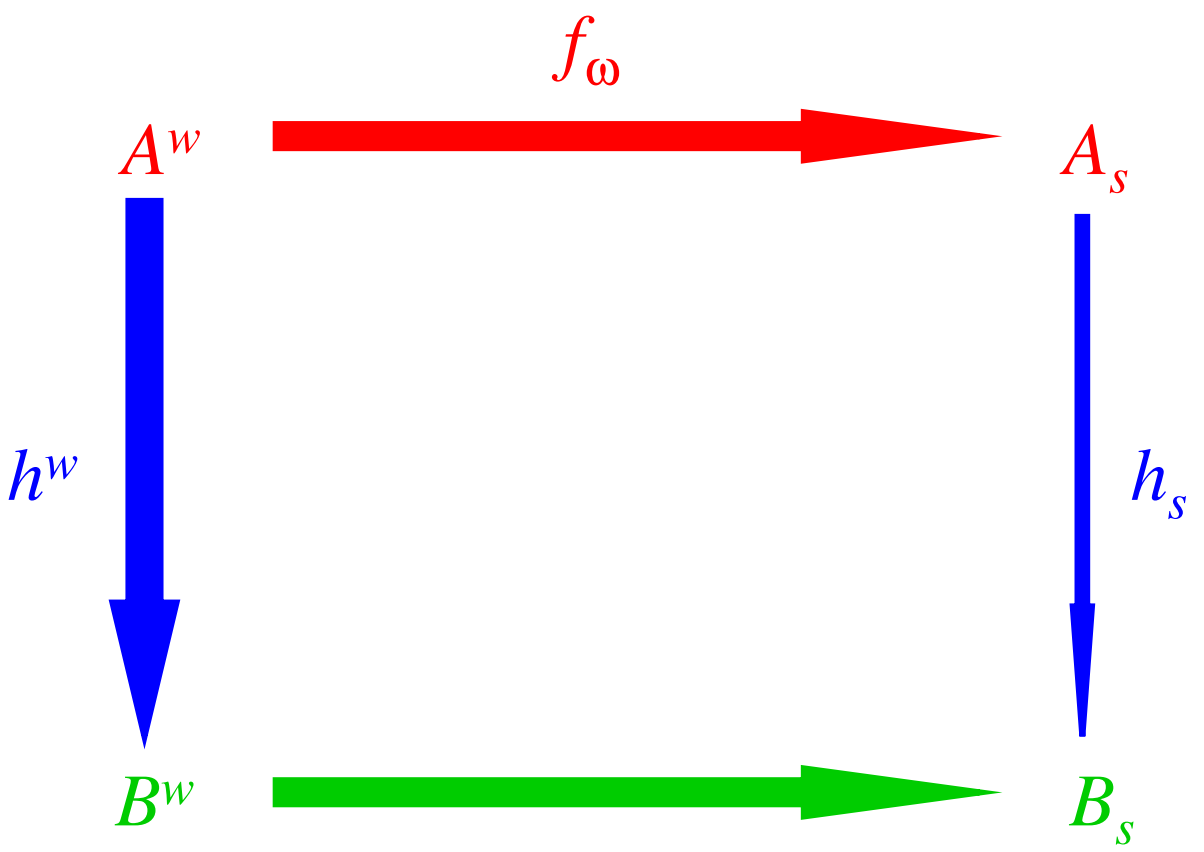
$\mathcal{A} = [A, (f_\omega)_{\omega \in \Omega}]$ und $\mathcal{L} = [B, (g_\omega)_{\omega \in \Omega}]$ seien zwei Σ-Algebren.

Ein ***Σ-Homomorphismus*** (kurz Homomorphismus) von $\mathcal{A}$ in $\mathcal{L}$ ist eine eindeutige Abbildung(sfamilie)

$$h : A \mapsto B,$$

so daß für alle $\omega \in \Omega$ mit $\alpha(\omega) = (s_1, s_2, ..., s_n, s)$ und für alle $(a_1, a_2, ..., a_n) \in A_{s_1} \times A_{s_2} \times ... \times A_{s_n}$ gilt:

$$h_s f_\omega(a_1, a_2, ..., a_n) = g_\omega(h_{s_1} a_1, h_{s_2} a_2, ..., h_{s_n} a_n) .$$



$$h_s f_\omega(a_1,a_2,\ldots,a_n) = g_\omega(h_{s_1}a_1,\ h_{s_2}a_2,\ \ldots,\ h_{s_n}a_n)$$
$$= g_\omega(b_1,\ b_2,\ \ldots,\ b_n)$$

Spezialfälle von Homomorphismen von $\mathcal{A}$ in $\mathcal{L}$:

- ***Einbettung:*** eineindeutiger Homom. von $\mathcal{A}$ in $\mathcal{L}$
- ***Isomorphismus:*** eineind. Homom. von $\mathcal{A}$ auf $\mathcal{L}$
  $\mathcal{A}$ und $\mathcal{L}$ heißen dann ***isomorph***: $\mathcal{A} \cong \mathcal{L}$
- ***Epimorphismus:*** Homomorphismus von $\mathcal{A}$ auf $\mathcal{L}$
  $\mathcal{L}$ heißt dann ***homomorphes Bild*** von $\mathcal{A}$
- ***Endomorphismus:*** Homomorphismus von $\mathcal{A}$ in $\mathcal{A}$

Ein Homomorphismus bildet die Konstanten von $\mathcal{A}$ auf die von $\mathcal{L}$ ab.

Denn: Für alle $\omega \in \Omega$ mit $i_\alpha(\omega) = \varepsilon$, $o_\alpha(\omega) = s$ folgt

$$h_s f_\omega = g_\omega.$$

Nacheinanderausführung von Homomorphismen:

$\mathcal{A}$, $\mathcal{B}$ und $\mathcal{C}$ seien drei Σ-Algebren,
h sei ein Homomorphismus von $\mathcal{A}$ in $\mathcal{B}$, g sei Homomorphismus von $\mathcal{B}$ in $\mathcal{C}$.
Dann ist $g \circ h$ Homomorphismus von $\mathcal{A}$ in $\mathcal{C}$.

Homomorphes Bild der Hülle:

Sei h ein Homomorphismus von $\mathcal{A}$ in $\mathcal{B}$. Weiter sei $X \subseteq A$.
Dann ist

$$h([X]_{\mathcal{A}}) = [h(X)]_{\mathcal{B}}$$

Eindeutigkeit der homomorphen Fortsetzung:

Es seien $\mathcal{A} = [A, F]$ und $\mathcal{B} = [B, G]$ zwei Σ-Algebren, X sei Erzeugendensystem von $\mathcal{A}$, und φ_0 eine eindeutige Abbildungsfamilie von X in B.
Dann gibt es höchstens einen Σ-Homomorphismus φ von $\mathcal{A}$ in $\mathcal{B}$, der die gegebene Abbildung φ_0 fortsetzt.

Beweisidee:

Gäbe es zwei Homomorphismen φ und ψ, die φ_0 fortsetzen:

$$\varphi|_X = \psi|_X = \varphi_0 \ .$$

Betrachte nun die Mengenfamilie

$$M = (M_s)_{s \in S} \text{ mit } M_s = \{ a \mid a \in A_s \wedge \varphi_s(a) = \psi_s(a) \}$$

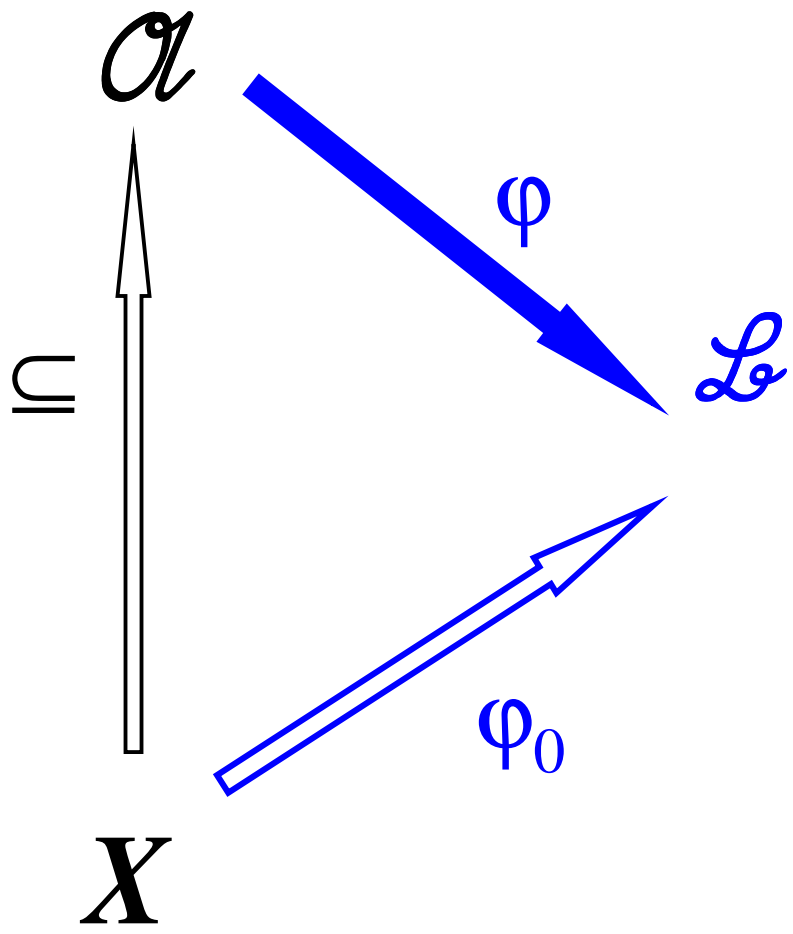
von Mengen bildgleicher Elemente und zeige durch algebraische Induktion, daß $M = A$ ist.

Termalgebren

Definition "Freie Algebra"

Die Σ-Algebra $\mathcal{A} = [A, F]$ heißt ***frei mit dem freien Erzeugendensystem*** X ***(über der Klasse aller Σ-Algebren)*** gdw.
X Erzeugendensystem von $\mathcal{A}$ ist, so daß für jede Σ-Algebra $\mathcal{L} = [B, G]$ und jede eindeutige Abbildungsfamilie φ_0 von X in B genau ein Σ-Homomorphismus φ von $\mathcal{A}$ in $\mathcal{L}$ existiert, der φ_0 fortsetzt.

$\mathcal{A}$ frei über X:



Isomorphie freier Algebren:

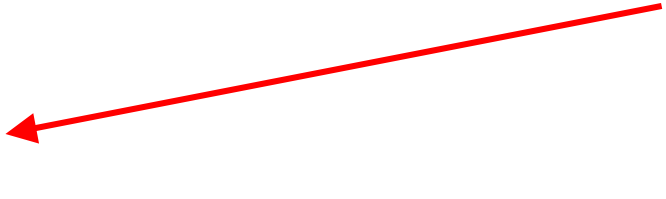
$\mathcal{A} = [A, F]$ und $\mathcal{B} = [B, G]$ seien zwei freie Σ-Algebren mit den freien Erzeugendensystemen X bzw. Y. Dann gilt: $\mathcal{A}$ und $\mathcal{B}$ sind isomorph, ***wenn*** ihre freien Erzeugendensysteme X bzw. Y sortenweise gleichmächtig sind.

Beweisidee: Wegen der Gleichmächtigkeit von X und Y gibt es eine 1-1-Abbildung ψ von X auf Y, die man wegen der Freiheit von $\mathcal{A}$ zu einem Homomorphismus ψ' von $\mathcal{A}$ in $\mathcal{B}$ fortsetzen kann. Die inverse Abbildung ψ^{-1} kann analog wegen der Freiheit von $\mathcal{B}$ zu einem Homomorphismus $(\psi^{-1})'$ von $\mathcal{B}$ in $\mathcal{A}$ fortgesetzt werden. Um $\mathcal{A} \cong \mathcal{B}$ zu sehen, überzeuge man sich davon, daß beides Epimorphismen sind und zueinander invers.

$\neg(p \wedge (r \vee \neg q))$

keine Variable, zu $im(neg)$: $(p \wedge (r \vee \neg q))$

keine Variable, zu $im(con)$: $(p \wedge (r \vee \neg q))$

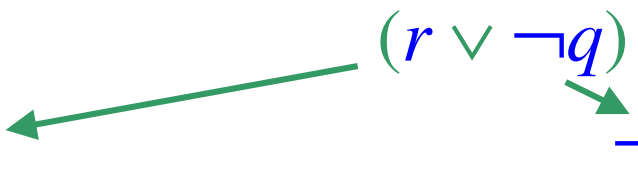


p

Variable

$(r \vee \neg q)$

keine Variable, zu $im(alt)$



r

Variable

$\neg q$

keine Variable, zu $im(neg)$

q

Variable

Eine Σ-Algebra $\mathcal{A} = [(E_s)_{s \in S}, (f_\omega)_{\omega \in \Omega}]$ heißt ***Σ-Ausdrucksalgebra (Σ-Termalgebra) über dem Alphabet*** X oder auch ***Σ-PEANO-Algebra über der PEANO-Basis*** X genau dann, wenn

$$X = (X_s)_{s \in S}$$

und für alle $s \in S$ ist $X_s \subseteq E_s$,

und die sogenannten ***verallgemeinerten PEANO-Axiome*** gelten.

Die Elemente von E_s heißen dann ***Terme*** oder ***Ausdrücke*** der Sorte s, die von X_s heißen ***Variablen*** der Sorte s.

Die ***verallgemeinerten*** PEANO-***Axiome*** sind die folgenden Bedingungen:

(P1) $\forall\omega\forall s\forall v\ (\omega\in\Omega \wedge o_\alpha(\omega) = s \wedge v \in E^{i\alpha(\omega)} \rightarrow f_\omega(v) \notin X)$

(P2) $\forall\omega_1\forall\omega_2\forall s\forall v\forall b\ (\omega_1,\omega_2\in\Omega \wedge o_\alpha(\omega_1)=o_\alpha(\omega_2)= s \wedge v \in E^{i\alpha(\omega_1)} \wedge b \in E^{i\alpha(\omega_2)} \wedge f_{\omega_1}(v) = f_{\omega_2}(b) \rightarrow \omega_1 = \omega_2 \wedge v = b)$

(P3) $[X]_\ell = E$

Fallunterscheidungssatz für Termalgebren:

Ist $\mathcal{E}$ eine Σ-Termalgebra über X, so gilt für alle $e \in E$:
Es gibt ein $s \in S$ und es ist entweder $e \in X_s$ oder es existiert genau ein $\omega \in \Omega$ mit $o_\alpha(\omega) = s$ und genau ein Ausdruckstupel $(e_1, e_2, ..., e_n) \in E^{i_\alpha(\omega)}$ mit
$e = f_\omega(e_1, e_2, ..., e_n)$.

Beweis:

Wegen (P3) und dem Darstellungssatz für die algebraische Hülle ergibt sich die Behauptung trivialerweise aus den Bedingungen (P1) und (P2).

Folgerung:

Ist $\mathcal{E}$ Σ-Termalgebra über X, so gilt für alle $s \in S$:

$$X_s = E_s \setminus \bigcup_{\substack{\omega \in \Omega \\ o_\alpha(\omega) = s}} im\,(f_\omega)\ .$$

Mit der Angabe einer Σ-Termalgebra $\mathcal{E} = [E, F]$ ist damit zugleich ihr Alphabet X festgelegt.

Es sei $\mathcal{A} = [A, F]$ eine Σ-Algebra und $\mathcal{E} = [E, G]$ eine Σ-Ausdrucksalgebra über dem Alphabet X. Jede eindeutige Abbildung $\varphi: X \mapsto A$ heißt ***$\mathcal{A}$-Belegung*** von X. Eine eindeutige Abbildung $\varphi^*: E \mapsto A$, die für alle $s \in S$ folgenden Gleichungen

1. $\varphi_s^*(x) = \varphi_s(x)$ für alle $x \in X_s$ und
2. $\varphi_s^*(g_\omega(e_1, e_2, ..., e_n)) = f_\omega(\varphi^{*w}(e_1, e_2, ..., e_n))$
   alle $\omega \in \Omega$ mit $\alpha(\omega) = (w, s)$ und alle $(e_1, e_2, ..., e_n) \in E^w$

genügt, heißt ***homomorphe*** (oder ***natürliche***) ***Fortsetzung*** der $\mathcal{A}$-Belegung von X.

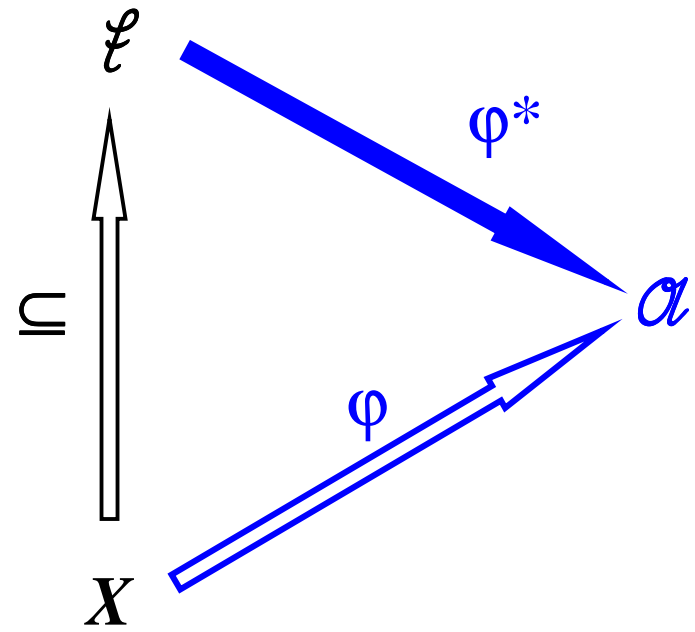
Die Voraussetzung $\mathcal{E} = [E, F]$ Σ-Termalgebra ist wesentlich.

Fortsetzungssatz für Termalgebren:

Es sei $\mathcal{A} = [A, F]$ eine Σ-Algebra und $\mathcal{E} = [E, G]$ eine Σ-Ausdrucksalgebra über dem Alphabet X. Dann existiert zu jeder $\mathcal{A}$-Belegung φ von X genau ein Homomorphismus φ^* von $\mathcal{E}$ in $\mathcal{A}$, der über X mit φ übereinstimmt. Dieser ist die homomorphe Fortsetzung von φ.

Beweis:

Die Existenz einer homomorphen Fortsetzung φ^* ergibt sich aus dem Fallunterscheidungssatz sofort. Außerdem ist φ^* wegen 2. Gleichung Homomorphismus von $\mathcal{E}$ in $\mathcal{A}$. Die Eindeutigkeit folgt aus dem Satz von der Eindeutigkeit der homomorphen Fortsetzung.



Folgerung:

Jede Σ-PEANO-Algebra über X ist also freie Algebra mit dem freien Erzeugendensystem X über der Klasse aller Σ-Algebren.

Isomorphie von PEANO-Algebren:

Zwei Σ-PEANO-Algebren $\mathcal{A} = [A, F]$ und $\mathcal{B} = [B, G]$ sind isomorph ***genau dann, wenn*** ihre PEANO-Basen X bzw. Y sortenweise gleichmächtig sind.

Beweis:

Einfach wegen dem vorigen Satz und dem Satz über die Isomorphie freier Algebren, die umgekehrte Richtung ergibt sich leicht aus der Folgerung zum Fallunterscheidungssatz für Termalgebren.

Sei $\Sigma = (S, \Omega, \alpha)$ Signatur und sei $X = (X_s)_{s \in S}$ eine disjunkte Mengenfamilie, die auch zu Ω disjunkt ist.
Die Mengen $T_{i,s}$, $i \in \underline{Nz}$, $s \in S$ seien folgende Mengen endlicher Folgen:

$$T_{0,s} = X_s \cup \{ \omega \mid \omega \in \Omega \wedge \alpha(\omega) = (s) \},$$

$$T_{i+1,s} = T_{i,s} \cup \{ \omega t_1 t_2 ,\ldots t_n \mid \omega \in \Omega \wedge \exists w(\alpha(\omega) = (w,s) \wedge (t_1, t_2 ,\ldots, t_n) \in T_i^w \}.$$

Weiter bezeichne

$$T_s = \bigcup_{i \in Nz} T_{i,s}$$

und

$$T_\Sigma(X) = (T_\Sigma(X)_s)_{s \in S} = (T_s)_{s \in S} .$$

Bezeichnet nun f_ω für jedes $\omega \in \Omega$ mit $\alpha(\omega) = (w,s)$ die Funktion

$$f_\omega : T_\Sigma(X)^w \mapsto T_\Sigma(X)_s$$

vermöge

$$f_\omega (t_1,t_2 ,...,t_n) = \omega t_1 t_2 ,... t_n$$

für alle $(t_1,t_2 ,...,t_n) \in T_\Sigma(X)^w$ mit $w \neq \varepsilon$

und $$f_\omega (\varepsilon) = \omega ,$$

dann ist

$$\underline{T_\Sigma(X)} = [\, T_\Sigma(X) , (f_\omega)_{\omega \in \Omega} \,]$$

eine Σ-Algebra und heißt ***Standardtermalgebra*** $\underline{T_\Sigma(X)}$ der Signatur Σ über dem ***Variablensystem*** (dem Alphabet) X.

Rechtfertigung der Bezeichnung:
Die Standardtermalgebra $\underline{T_\Sigma(X)}$ über X ist eine PEANO-Algebra über X.

Beweis: durch Nachweis der verallgemeinerten PEANO-Axiome

Folgerung:
Jede Σ-PEANO-Algebra über X ist isomorph zur Standardtermalgebra $\underline{T_\Sigma(X)}$.

In der Definition von $\underline{T_\Sigma(X)}$ kann X auch als S-sortige Familie leerer Mengen $X = \underline{\emptyset} = (\emptyset)_{s \in S}$ gewählt werden.

Definition "Grundterme"

Es wird gesetzt:

$$T_\Sigma = T_\Sigma(\emptyset) \text{ und } \underline{T_\Sigma} = \underline{T_\Sigma(\emptyset)}\,.$$

Die Elemente von T_Σ heißen (Standard-)***Grundterme***.

Initialität:

Für jede **Σ**-Algebra $\mathcal{A}$ gibt es genau einen Homomorphismus

$$h_{\mathcal{A}} : \underline{T_\Sigma} \mapsto \mathcal{A}\,.$$

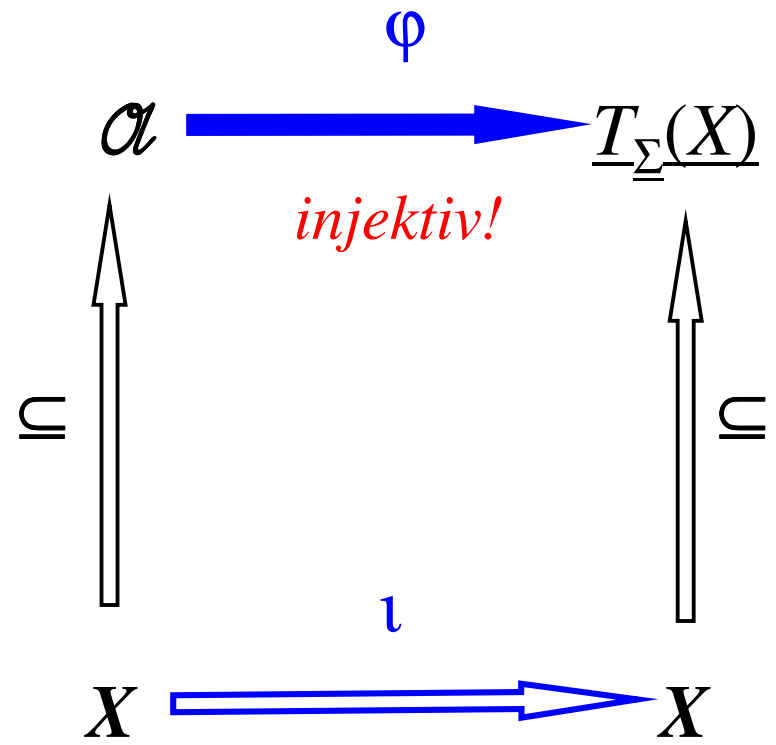
"frei ist PEANOsch":

Ist die Σ-Algebra $\mathcal{A} = [A, F]$ frei über der Klasse aller Σ-Algebren mit dem freien Erzeugendensystem X, welches zu Ω disjunkt ist, so ist $\mathcal{A}$ eine Σ-PEANO-Algebra über der PEANO-Basis X.

Beweis: durch Nachweis der verallgemeinerten PEANO-Axiome

Folgerungen:

1. Ist $\mathcal{A}$ eine freie Algebra mit einem freien Erzeugendensystem X, so ist dieses eindeutig bestimmt.
2. Freie Algebren $\mathcal{A}$ und $\mathcal{L}$ sind isomorph **genau dann, wenn** ihre freien Erzeugendensysteme sortenweise gleichmächtig sind.



Jede freie Σ-Algebra $\mathcal{A}$, frei über X, ist isomorph zur Standardtermalgebra $\underline{T_\Sigma(X)}$ vermöge eines X festlassenden Isomorphismus.

Zusammenfassung

Begriff	Charakterisierungen	
PEANO-Algebra	innere	analytisch
freie Algebra	äußere	algebraisch
Standard-termalgebra	konstruktiv	synthetisch

Äquivalenz der drei Charakterisierungen für Termalgebren:

Für jede Σ-Algebra $\mathcal{A} = [A, F]$ und beliebige Teilfamilien $X \subseteq A$ sind folgende Aussagen äquivalent:

1. $\mathcal{A}$ ist Σ-PEANO-Algebra über der PEANO-Basis X.
2. $\mathcal{A}$ ist frei in der Klasse aller Σ-Algebren mit dem freien Erzeugendensystem X.
3. $\mathcal{A}$ ist isomorph zu $\underline{T_\Sigma(X)}$ vermöge eines X festlassenden Isomorphismus.

Beweis:

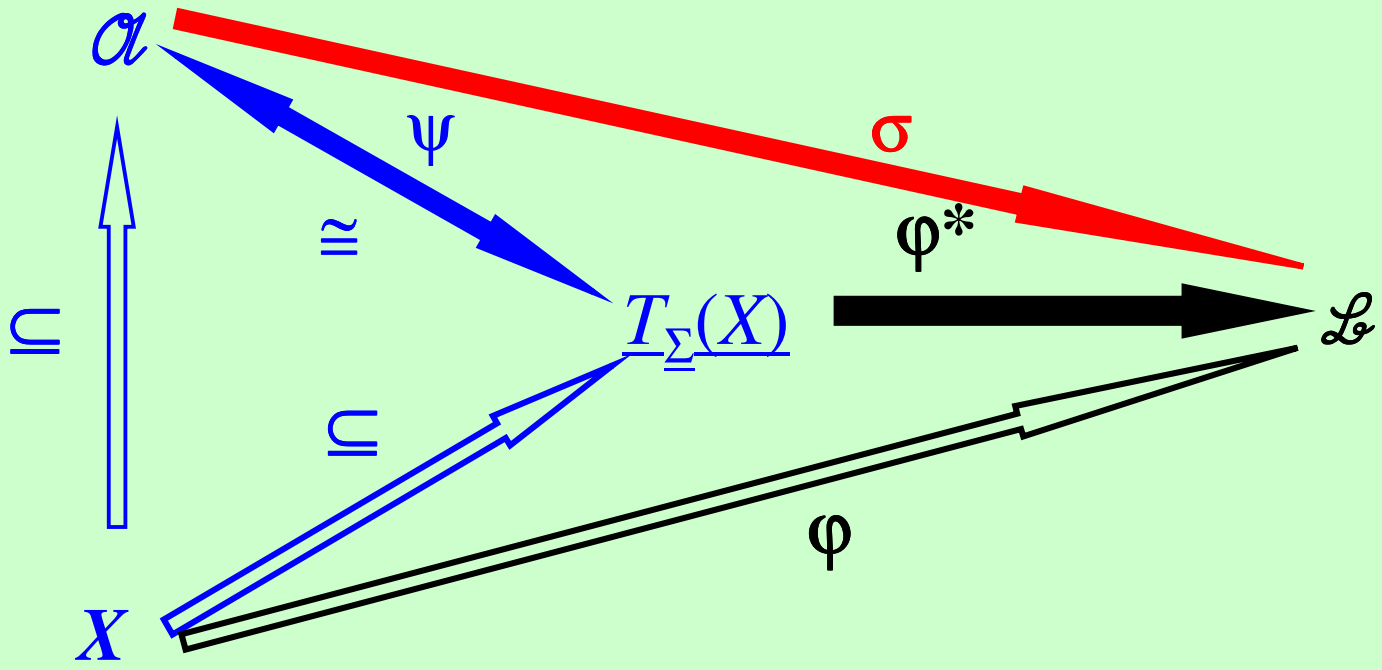
(1) $\rightarrow$ (2) : Folgerung zum Fortsetzungssatz (Folie 11)

(2) $\rightarrow$ (1) : **"frei ist PEANOsch"** (Folie 17)

(1) $\rightarrow$ (3) : Folgerung zu $\underline{T_\Sigma(X)}$ ist PEANO-Algebra (Folie 15)

(3) $\rightarrow$ (2) : wird im folgenden noch gezeigt !

(3) → (2) :



$\sigma = \varphi^* \circ \psi$ Homomorphismus von $\mathcal{A}$ in $\mathcal{L}$ mit

$\sigma|_X = (\varphi^* \circ \psi)|_X = \varphi^* \circ \psi|_X = \varphi^* \circ \iota_X = \varphi^*|_X = \varphi$,

also ist σ homomorphe Fortsetzung von φ.

Beispiele für Ausdrucksalgebren

1. $\mathcal{A}$ = [**AUS** ; *neg, con, alt*] - die Algebra der pfeilfreien aussagenlogischen Ausdrücke ist ein klassisches Beispiel für eine homogene Ausdrucksalgebra (Namensgeber!)
   - vergleiche dazu die Überlegungen zur Syntaxanalyse und die aus der Logik bekannten Sätze über die Ausdrücke

2. $\mathcal{N}$ = [**Nz** ; **0 , suc**] - „die“ (?) PEANO-Algebra der natürlichen Zahlen mit der Konstanten **0** und der Nachfolgerfunktion **suc**
   - die PEANO-Axiome sind ein Spezialfall der verallgemeinerten PEANO-Axiome, folglich genügt $\mathcal{N}$ den Bedingungen (P1) - (P3), PEANO-Basis ist hier $\varnothing$

Beispiele für Ausdrucksalgebren (Forts.)

2. $\mathcal{N}$ = [**Nz** ; **0 , suc**] - ist PEANO-Algebra unabhängig von der „konkreten“ Gestalt der natürlichen Zahlen:
   - **Nz** = {*0,1,2, ...*} ,
   - **Nz** = { I, II, III, IIII, ...} ,
   - **Nz** = Menge der Äquivalenzklassen gleichmächtiger endlicher Mengen,
   - **Nz** = { **O**, **L**, **LO**, **LL**, **LOO**, ...} ,
   - **Nz** = Menge der Ordinalzahlen,

     • • •

   $\mathcal{N}$ ist homogene Algebra vom Typ τ mit

$$\Omega = \{O, ' \},\ \tau(O) = 0,\ \tau(') = 1.$$

Beispiele für Ausdrucksalgebren (Forts.)

zu 2.: Σ bezeichne wie üblich die sich ergebende Signatur.

(a) Terme der Standardtermalgebra $\underline{T_\Sigma}$ sind z.B.

O, 'O, "O, "'O, ...

(b) die Termalgebra mit den (entsprechenden) Termen

O, O', O", O"', ...

ist natürlich isomorph zu $\underline{T_\Sigma}$; genauso wie

(c) $\mathcal{N}$ = [**Nz** ; **0 , suc**] .

(d) die folgenden Ausdrücke gehören zu einer Ausdrucksalgebra derselben Signatur über der Variablenmenge $X = \{ x_1, x_2, x_3, ... \}$:

x_1, x_2, x_{27}, O, O"""", x_{359}"""", ...

Beispiele für Ausdrucksalgebren (Forts.)

3. Algebra arithmetischer blauer Terme:

Zeichenvorrat $\Gamma = \{\ 0, +,), (, x_1, x_2, x_3, \ldots\ \}$ sei gegeben.

Setze $X = \{\ x_1, x_2, x_3, \ldots\ \}$, $\Omega = \{0, +\}$ und bilde die homogene Algebra $\mathcal{E} = [E; f_0, f_+]$ vom Typ (0,2) mit

$$f_0 = 0, \quad f_+(e_1, e_2) = (e_1 + e_2) \text{ für bel. } e_1, e_2 \in E$$

(a) Ausdrücke (d.h. Elemente von E) sind z.B.

$$0,\ ((x_1 + 0) + x_5),\ (x_1 + (0 + x_3)),\ (x_{34} + x_{87}),\ \ldots$$

(b) wenn man die Definition von f_+ abändert in

$$f_+(e_1, e_2) = +e_1e_2\ ,$$

so erhält man analog die Ausdrücke bzw. Terme

$$0,\ ++x_10x_5\ ,\ +x_1+0x_3\ ,\ +x_{34}x_{87}\ ,\ \ldots$$

Beispiele für Ausdrucksalgebren (Forts.)

4. Homogene Termalgebra über $X = \{p, q, r\}$ mit $\Omega = \{\wedge , \vee\}$ vom Typ (2,2) :

Terme:

(a) bei Benutzung der üblichen *Folgenschreibweise*

- vergleiche dazu die Def. von $T_\Sigma(X)$

(p) , (∧, (∧, q, r), p) , (∧, (∨, p, r), (∧, p, p)) , ...

(b) bei Verwendung von *Zeichenreihenschreibweisen*:

Präfixschreibweise: p, ∧∧qrp, ∧∨pr∧pp

Infixschreibweise: p, ((q ∧ r) ∧ p) , ((p ∨ r) ∧ (p ∧ p))

Postfixschreibweise: p, qr∧p∧ , pr∨pp∧∧

Funktionsschreibweise: p, ∧(∧(q, r), p) , ∧(∨(p, r), ∧(p, p))

Beispiele für Ausdrucksalgebren (Forts.)

5. Zum MEALY-Automaten :

$$\Sigma = (S, \Omega, \alpha)\ ,\ \ S = \{x, y, z\},\ \Omega = \{1, 2\},$$
$$\alpha(1) = (z, x, z),\ \alpha(2) = (z, x, y).$$

Bilde Σ-Standardtermalgebra über $V = (V_s)_{s \in S}$ mit

$V_x = \{ x_1, x_2, x_3, ... \},\ V_y = \{ y_1, y_2, y_3, ... \},\ V_z = \{ z_1, z_2, z_3 \}.$

Zu $\underline{T_\Sigma(V)}$ gehören z.B. folgende Terme:

in T_x: $x_1,\ x_2,\ x_3, ...$ keine weiteren !

in T_y: $y_1,\ y_2,\ y_3, ...$

$2z_1x_1,\ 2z_1x_2,\ 2z_1x_3, ...$

$2z_2x_1,\ 2z_2x_2,\ 2z_2x_3, ...$

$2z_3x_1,\ 2z_3x_2,\ 2z_3x_3, ...$

zur Fortsetzung weitere "Zustandsterme" aus T_z nötig

Beispiele für Ausdrucksalgebren (Forts.)

weiter zu 5:

in T_z: $z_1,\ z_2,\ z_3$

$1z_1x_1,\ 1z_1x_2,\ 1z_1x_3,\ \ldots$

$1z_2x_1,\ 1z_2x_2,\ 1z_2x_3,\ \ldots$

$1z_3x_1,\ 1z_3x_2,\ 1z_3x_3,\ \ldots$

zur Fortsetzung diese schon gebildeten "Zustandsterme" aus T_z verwenden

$11\underbrace{z_1x_1}_{\in T_z}x_1,\ 11z_1x_1x_2,\ 11z_1x_1x_3,\ \ldots$

$\bullet\ \bullet\ \bullet$

zu T_y kommen nun dazu: $21z_1x_1x_1,\ 21z_1x_1x_2,\ 21z_1x_1x_3,\ \ldots$

$21z_1x_2x_1,\ 21z_1x_2x_2,\ 21z_1x_2x_3,\ \ldots$

$\bullet\ \bullet\ \bullet$

Beispiele für Ausdrucksalgebren (Forts.)

6. Zum Datentyp „Keller“ :

$\Sigma = (S, \Omega, \alpha)$ mit $S = \{d, s\}$, $\Omega = \{\blacklozenge, \#, \Lambda, \text{pop}, \text{push}, \text{top}\}$,

d für date, s für stack, $\blacklozenge$ - date error, # - stack error, Λ - leerer Keller

$$\alpha(\blacklozenge) = (d),\ \alpha(\#) = \alpha(\Lambda) = (s),$$

$$\alpha(\text{push}) = (d,s,s),\ \alpha(\text{pop}) = (s,s),\ \alpha(\text{top}) = (s,d).$$

Bilde Σ-Ausdrucksalgebra $\underline{E_\Sigma(X)}$ über $X = [X_d, X_s]$ mit

$$X_d = \{x_1, x_2, x_3, \ldots\},\ X_s = \varnothing$$

in schwacher Abwandlung der Definition von $\underline{T_\Sigma(X)}$:

in E_s z.B.: $\#, \Lambda, \text{push}(x_i\text{push}(x_j\Lambda)), \text{pop}(\text{push}(x_j\Lambda)), \ldots$

für beliebige i und j

in E_d z.B.: $x_1,\ x_2,\ \blacklozenge, \text{top}(\Lambda),\ \text{top}(\text{push}(x_1\text{push}(x_{27}\Lambda)),$

$\text{top}(\text{push}(x_7\Lambda)),\ \ldots$

7. Beispiel

Zuordung der syntaktischen Algebra zu kontextfreier Grammatik

$G = (N, T, P, S)$ - gegebene kontextfreie Grammatik

N - Alphabet der Nichtterminale,

T - Alphabet der Terminalzeichen,

P - Menge der Produktionen : $P \subseteq N \times (T \cup N)^*$

S - Startsymbol: $S \in N$

Für jedes $A \in N$ setze $L_A = \{ w \mid A \rightarrow^* w \wedge w \in T^* \}$.

Es ist dann $L(G) = L_S$.

Zu Regel $p \in P$ der Gestalt

$A \rightarrow u_1 A_1 u_2 A_2 ... u_n A_n u_{n+1}$ mit $A, A_i \in N$, $u_j \in T^*$

definiere Operation $f_p : L_{A_1} \times L_{A_2} \times ... \times L_{A_n} \mapsto L_A$ mit

$$f_p(x_1, x_2, ..., x_n) = u_1 x_1 u_2 x_2 ... u_n x_n u_{n+1}$$

7. Beispiel (Forts.)

Die sich so ergebende Algebra wird mit $\underline{\mathrm{SYN}}_G$ bezeichnet:

$$\underline{\mathrm{SYN}}_G = [(L_A)_{A \in N}, (f_p)_{p \in P}]$$

Sie heißt die zu G gehörige ***syntaktische Algebra***.

Deren Signatur Σ_G ist ebenfalls aus der Grammatik erkennbar:

$$\Sigma_G = (N, P, \alpha) ,$$

wobei

$$\alpha (p) = (A_1, A_2, \ldots , A_n , A)$$

zu setzen ist, falls die Regel p die Gestalt

$$A \rightarrow u_1 A_1 u_2 A_2 \ldots u_n A_n u_{n+1} \qquad \text{mit } A, A_i \in N , u_j \in T^*$$

besitzt.

7. Beispiel (Forts.)

Man erkennt, daß $\underline{\mathrm{SYN}_G}$ die verallgemeinerten PEANO-Axiome erfüllt, falls <u>alle</u> Sprachen L_A, nicht nur $L(G)$ selbst, bzgl. G *eindeutige* Sprachen sind (im Sinne der Theorie formaler Sprachen!).

Im Falle der „grammatikalischen“ Eindeutigkeit ist also $\underline{\mathrm{SYN}_G}$ eine Ausdrucks- bzw. PEANO-Algebra.

Deren eindeutig bestimmtes Erzeugendensystem X ist dann das System leerer Mengen wegen

$$X_A = L_A \setminus \bigcup_{l(p) = A} im\,(f_p) = \varnothing\,.$$

$l(p)$ bezeichnet hier die linke Seite der Regel p.

7. Beispiel (Forts.)

Die Signatur Σ_G, die durch die kontextfreie Grammatik G festgelegt ist, bestimmt auch die zugehörige initiale (variablenfreie) Standardtermalgebra $\underline{T_{\Sigma_G}}$, kurz $\underline{T_G}$.
Entsprechend der allgemeinen Definition von $\underline{T_\Sigma}$ erhält man hier als Elemente der Trägermenge $T_{G,A}$ von $\underline{T_G}$ alle Folgen von Regeln, die zu vollständigen Linksableitungen in G, beginnend mit A gehören. Vergleiche dazu die Erläuterungen am Beispiel
Da zu jeder Linksableitung eineindeutig ein Ableitungsbaum gehört, sagt man, daß die initiale Termalgebra der Signatur Σ_G die Algebra der *Ableitungsbäume* ist.
Diese Algebra $\underline{T_G}$ heißt deshalb auch ***abstrakte Syntax***.

7. Beispiel (Forts.)

Beispiel

$G = (\{A, B\}, \{ +,), (, a, b \}, P, A)$

mit $P = \{ A \to (A+B),\ A \to a,\ B \to b \}.$

Also ist $\Sigma_G = (\{A, B\}, P,\ \alpha).$

Regelnamen	*Regeln*	$\alpha(p) = (w,s)$	
+	$A \to (A+B)$	(AB, A)	
a	$A \to a$	(A)	} Konstanten!
b	$B \to b$	(B)	

zu $T_{G,B}$ nur: $B \to b$ als Konstante

zu $T_{G,A}$: Wie sehen die „Standardterme“ der Sorte A aus?

Es gibt zwei Operatoren mit Ausgang A !

7. Beispiel (Forts.)

Beispiel (Forts.)

zu $T_{G,A}$ z.B. : $A \to a$ als Konstante

$A \to (A+B)\ A \to a\ B \to b$

$\omega \qquad t_1 \qquad t_2$

$A \to (A+B)\ \underbrace{A \to (A+B)\ A \to a\ B \to b}\ B \to b$

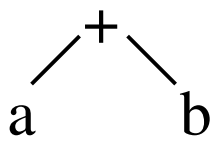
$\omega \qquad\qquad t_1 \qquad\qquad t_2$

Umbenennung der Operatoren ergibt folgende Schreibweise derselben

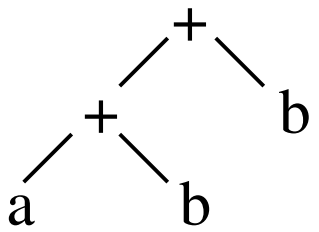
Terme: a • a

+ ab

++abb



Ableitungsbäume



Kapitel 5

Syntax und Semantik

Die Syntax einer formalen Sprache kann als eine i.a. mehrsortige Algebra angesehen werden:

$$\underline{\text{SYN}} = [(L_s)_{s \in S}, (f_\omega)_{\omega \in \Omega}]$$

<u>SYN</u> wird ***syntaktische Algebra*** genannt.

L_s - Teilsprachen, Menge der syntaktischen Objekte der Sorte s, grammatikalische Einheit, Elemente gleichen „Typs“

f_ω - syntaktische Operationen, bildet aus syntaktischen Objekten neue syntaktische Objekte

Vergleiche Beispiele vorn: logische Ausdrücke, arithmetische Terme, Typ-2- Sprachen

~~~~~~~~~~~~~~

? Grammatik ⇔ Orthographie ?
~~~~~~~~~~~~~~

Zwei Niveaustufen der Syntax

- Ebene der „Sprachstruktur“ = Grammatik der Sprache
- Ebene der „Rechtschreibung“ oder Orthographie

Beispiel Entwurf einer Programmiersprache
Entscheidung über **Varianten** „bedingter Anweisungen“ zu treffen:

- „Ja-Nein-Verzweigung“:
  eine Bedingung, zwei Anweisungen
- „unvollständige bedingte Anweisung“:
  eine Bedingung, eine Anweisung
- „n-zweigige Fallunterscheidung“ (DIJKSTRAsche *guarded commands*):
  beliebige Anzahl Paare Bedingung - Anweisung

= rein grammatikalische bzw. strukturelle Ebene

Varianten für Notationen der „Ja-Nein-Verzweigung“:

- Cond ::= **if** Exp **then** Com **else** Com **fi**
- Cond ::= **case** Exp **true** Com **false** Com
- Cond ::= Exp → Com , Com
- Cond ::= (Exp | Com | Com)

= Rechtschreibungsebene

Die üblichen Methoden der Syntaxdefinition von Programmiersprachen wie BACKUS-NAUR-Form, Syntaxdiagramm, CHOMSKY-Grammatik erlauben es **nicht**, die strukturelle Ebene ***unabhängig*** von der Orthographie zu definieren.

Abstrakte Syntax

Vergleiche vorn 7. Beispiel

Durch die Signatur Σ der syntaktischen Algebra SYN wird die initiale Termalgebra (= irgendwie ausgezeichnete variablenfreie Standardtermalgebra = Algebra der Ableitungsbäume) bestimmt.

Diese heißt auch ***abstrakte Syntax*** ABS.

Sie „trägt“ die Bedeutung der Sprachelemente.

Die abstrakte Syntax repräsentiert also die strukturelle Ebene der Syntax.

Die orthographische Ebene, die in SYN ausgedrückt wird, ist von erheblicher pragmatischer Bedeutung.

Semantik

FREGEsches Prinzip: Gottlob Frege (1848-1925)

Die Bedeutung des Ganzen ist eine Funktion der Bedeutung seiner Teile.

$$\mathbf{M}\ [\ g\ (t_1, t_2\ , \ldots, t_n)\ ] = \varphi\ (\ \mathbf{M}\ [t_1], \mathbf{M}\ [t_2], \ldots, \mathbf{M}\ [t_n]\ )$$

Bedeutung von — Konstruktion eines Ganzen

Teilkonstrukte

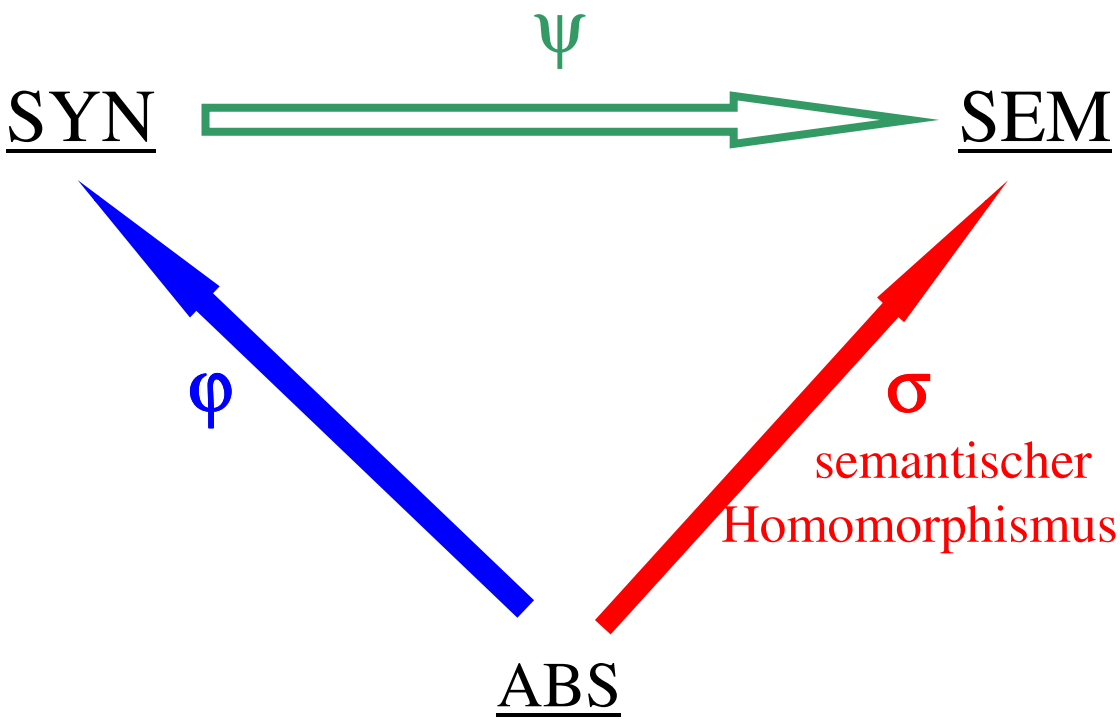
g entsprechende semantische Zusammensetzungs-funktion

Bedeutung der Teile

Entsprechend den Eigenschaften der Termalgebren (Initialität !) existiert zu jeder beliebigen Algebra gleicher Signatur genau ein Homomorphismus von der abstrakten Syntax in die gegebene Algebra.

- Kerngedanke der „Algebraischen Semantik“.

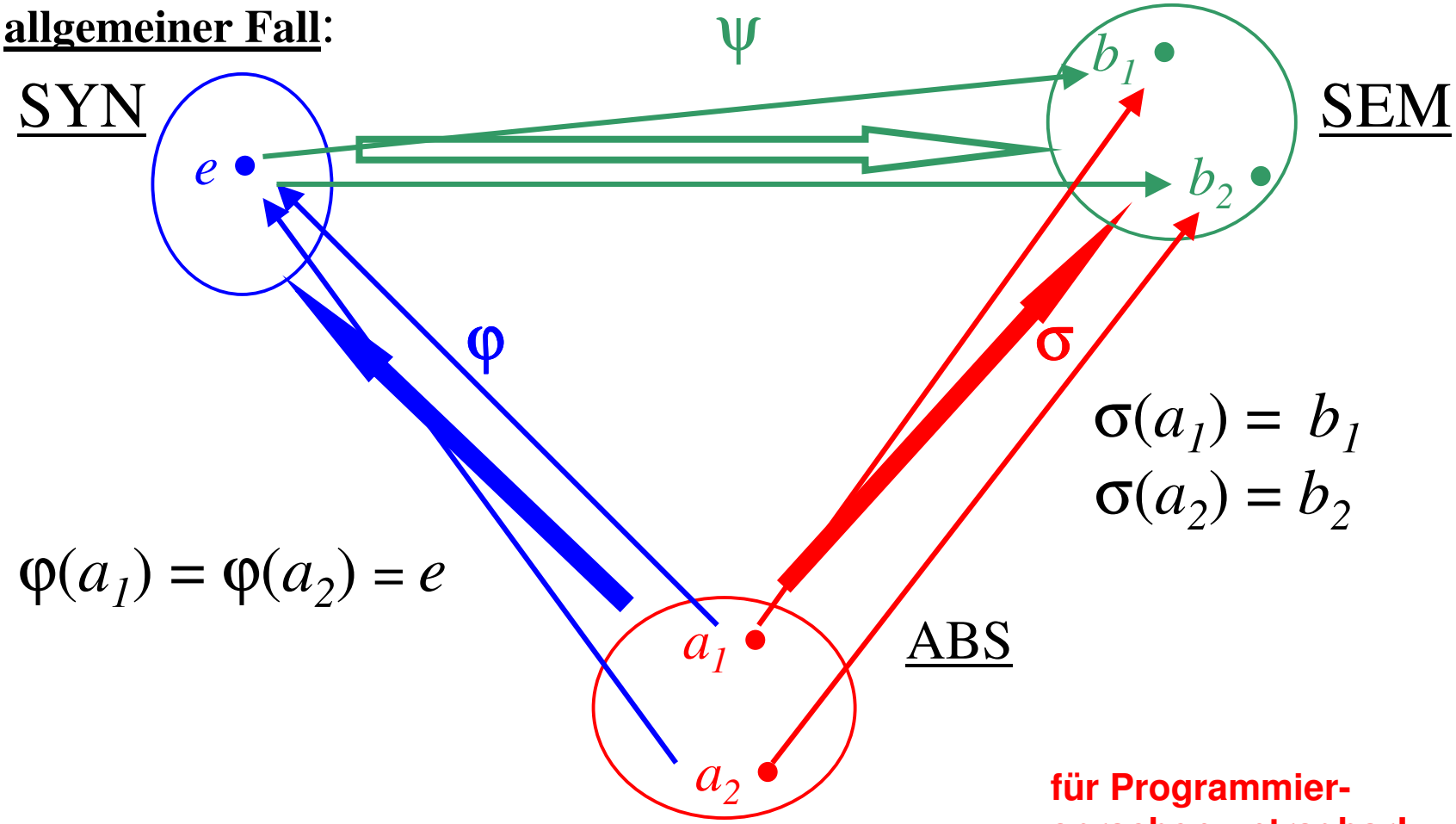
Jede Algebra mit derselben Signatur wie die syntaktische Algebra SYN (und die abstrakte Syntax ABS) kann damit als ***semantische Algebra*** SEM fungieren.
Der durch SEM eindeutig festgelegte Homomorphismus von ABS in SEM heißt ***semantischer Homomorphismus*** und vermittelt die ***Semantik*** der Sprache.
Durch ihn werden den syntaktischen Objekten aus SYN Bedeutungen zugeordnet.



ψ vermittelt die Semantik! Es ist $\psi = \sigma \circ \varphi^{-1}$.
φ^{-1} sollte Funktion, d.h. ***eindeutig*** sein, damit ψ eindeutig ist.
Bei Eindeutigkeit der Sprache ist auch SYN PEANO-Algebra und damit isomorph zu ABS, d.h. φ^{-1} auch Homomorphismus.

$\Rightarrow$ **Rolle der Rechtschreibung!**

allgemeiner Fall:



$\sigma(a_1) = b_1$

$\sigma(a_2) = b_2$

$\varphi(a_1) = \varphi(a_2) = e$

für Programmier-sprachen untragbar!

Mehrdeutigkeit $\psi(e) = b_1$ und $\psi(e) = b_2$

Rolle der Rechtschreibung:

- ***Pragmatische Aspekte:***

  Verständlichkeit (für den Menschen)
  durch Lesbarkeit, Übersichtlichkeit,
  Anknüpfen an Erfahrungen

- ***Sicherung der syntaktischen Eindeutigkeit***:

  geeigneter Einsatz von Klammern,
  Trennzeichen usw.

Semantische Eindeutigkeit:

Einzige Ausnahme der Zulassung von Mehrdeutigkeit!

Wenn in SEM für $\psi(e) = b_1$ und $\psi(e) = b_2$ stets gilt: $b_1 = b_2$

Beispiel: Assoziativität der Verkettung von Anweisungen

$$A_1 ; A_2 ; \ \bullet\bullet\bullet \ ; A_n$$

Fortsetzung des 7. Beispiels
Kontextfreie Grammatik, syntaktische Algebra, abstrakte Syntax

- $\underline{T_G}$ war die *abstrakte Syntax* der durch die kontextfreie Grammatik G gegebenen Signatur Σ_G, sie ist initial und zugleich die *Algebra der Ableitungsbäume*.
- Jede weitere Σ_G-Algebra kann als *semantische Algebra* dienen.
- Zu dieser Algebra muß dann ein eindeutig bestimmter *Homomorphismus* in diese existieren.
- Wir konstruieren nun im folgenden zu gegebener Signatur Σ_G eine konkrete Σ_G-Algebra und betrachten den zugehörigen *semantischen Homomorphismus*.

Fortsetzung des 7. Beispiels

Als Trägermengen wähle (alle gleich!)

T^* - Wortmenge über Terminalalphabet

und die Operationen f_p seien wie oben über $(L_A)_{A \in N}$ definiert, aber nun erweitert auf ganz T^*.

D.h., zu Regel $p \in P$ der Gestalt

$A \rightarrow u_1 A_1 u_2 A_2 \ldots u_n A_n u_{n+1}$ mit $A, A_i \in N$, $u_j \in T^*$

definiere Operation $f_p : T^* \times T^* \times \ldots \times T^* \mapsto T^*$ mit

$$f_p(x_1, x_2, \ldots, x_n) = u_1 x_1 u_2 x_2 \ldots u_n x_n u_{n+1} .$$

Die sich ergebende Algebra sei mit $\underline{D}_G$ bezeichnet:

$$\underline{D}_G = [(T^*)_{A \in N}, (f_p)_{p \in P}].$$

Fortsetzung des 7. Beispiels

$$\underline{D}_G = [(T^*)_{A \in N}, (f_p)_{p \in P}].$$

Von $\underline{T}_G$ in $\underline{D}_G$ existiert der eindeutig bestimmte Homomorphismus

$$h_D : \underline{T}_G \mapsto \underline{D}_G .$$

Er ordnet jedem „Ableitungsbaum“ aus $\underline{T}_G$ eine Zeichenreihe aus T^* zu, und zwar einem Ableitungsbaum der Sorte A_i (Wurzel ist A_i) das durch die zugehörige Linksableitung erzeugte Wort aus, d.h. das abgeleitete Wort aus L_{A_i} .

Als homomorphes Bild $h_D(\underline{T}_G)$ erhalten wir also gerade $\underline{\mathrm{SYN}}_G$.

Die „Zeichenreihenerzeugung“ kann also auch als *Semantik* der durch die Grammatik G festgelegten abstrakten Syntax $\underline{T}_G$ angesehen werden!

Fortsetzung des 7. Beispiels

Beispiel (vergleiche vorn): $T = \{ +,), (, a, b \}$ Terminalzeichen rot!

Regelnamen	*Regeln*	*Operationen* f_p	$\alpha(p) = (w,s)$
+	$A \rightarrow (A+B)$	$f_+(x_1, x_2) = (x_1+x_2)$	(AB, A)
a	$A \rightarrow a$	$f_a = a$	(A)
b	$B \rightarrow b$	$f_b = b$	(B)

Betrachte aus $T_{G,A}$: ++abb

$$
\begin{aligned}
h_{D,A}(++abb) &= f_+(h_{D,A}(+ab), h_{D,B}(b)) \\
&= f_+(f_+(h_{D,A}(a), h_{D,B}(b)), f_b) \\
&= f_+(f_+(f_a, f_b), b) \\
&= f_+(f_+(a, b), b) \\
&= f_+((a+b), b) \\
&= ((a+b)+b)
\end{aligned}
$$

Die „Algebraische Semantik“

- Methode der Semantikdefinition
- oft auch **denotationale Semantik** genannt

Durch geeignete Beschreibung der gemeinten ***semantischen Algebra*** SEM wird die beabsichtigte Semantik festgelegt.

Die sog. denotationale Semantik entwickelt u.a. spezifische Begriffe und Hilfsmittel zur Beschreibung der semantischen Operationen und Bereiche.
Durch geeignete Wahl der semantischen Bereiche („vollständige Halbordnungen“) wird die Existenz benötigter semantischer Konstruktionen (z.B. Fixpunkte) zur Beherrschung „komplizierterer“ Programmkonstrukte (z.B. Iterationen) gesichert.

Beispiel zur „algebraischen Semantik“

Vergleiche vorn Beispiel für heterogene Algebra: einfache Programmiersprache

$$\underline{\text{SYN}} = [\textbf{Ide}, \textbf{Exp}, \textbf{Com}; (f_p)_{p \in P}]$$

P ist dabei die Menge der Namen, die den Produktionsregeln der definierenden BNF zugeordnet wurden. siehe vorn

Wir setzen nun

$$\underline{\text{SEM}} = [\textbf{Ide}, \textbf{Ed}, \textbf{Cd}; (g_p)_{p \in P}].$$

Ide erste Trägermenge in $\underline{\text{SYN}}$ wie in $\underline{\text{SEM}}$,
d. h. die „Identifier“ sollen sich selbst bezeichnen!

Ed steht für „*expression denotations*“

Cd steht für „*command denotations*“

g_p - semantische Operationen

Zur Definition der Mengen **Ed**, **Cd** und der Funktionen g_p werden zunächst Hilfsoperationen und Hilfsbereiche eingeführt:

Beispiel zur „algebraischen Semantik“ (Forts.)

Wertemengen:

$\mathbf{V} = \mathbf{Nz} \cup \mathbf{Bool}$, $\mathbf{Nz} = \{0,1,2, \ldots\}$, $\mathbf{Bool} = \{\mathrm{T}, \mathrm{F}\}$

Hilfsfunktionen:

$+$ - Addition in **Nz**, $\neg$ - Negation in **Bool**,

auf **V** erweiterte Funktionen:

$$plus: \mathbf{V} \times \mathbf{V} \mapsto \mathbf{V} \quad \text{mit} \quad plus(v_1,v_2) = \begin{cases} v_1 + v_2, \text{ falls } v_1,v_2 \in \mathbf{Nz} \\ \underline{\text{err}} \quad \text{sonst} \end{cases}$$

$$non: \mathbf{V} \mapsto \mathbf{V} \quad \text{mit} \quad non(v) = \begin{cases} \neg v, \text{ falls } v \in \mathbf{Bool} \\ \underline{\text{err}} \quad \text{sonst} \end{cases}$$

! Generell alle Funktionen bzgl. $\underline{\text{err}}$ ***strikt erweitern***!

Beispiel zur „algebraischen Semantik“ (Forts.)

Es wird festgelegt

$\mathbf{S} = \mathbf{Ide} \mapsto (\mathbf{V} \cup \{\underline{\text{err}}\})$ als ***Zustands***- bzw. ***Belegungsmenge***,

- auch für „unbestimmt“, kein Wert zugewiesen

$F = M \mapsto N$ Menge aller Funktionen von M in N

Semantische Bereiche:

$\mathbf{Ed} = \mathbf{S} \mapsto (\mathbf{V} \cup \{\underline{\text{err}}\})$ Ausdrucksbedeutungen

$\mathbf{Cd} = \mathbf{S} \mapsto (\mathbf{S} \cup \{\underline{\text{err}}\})$ Anweisungsbedeutungen

---

Für die folgende Definition der semantischen Operationen seien immer $s \in \mathbf{S}$, $\mathrm{I} \in \mathbf{Ide}$, $e, e', e_1, e_2 \in \mathbf{Ed}$, $c, c_1, c_2 \in \mathbf{Cd}$.

Beispiel zur „algebraischen Semantik“ (Forts.)

Semantische Operationen $g_p \quad (p \in P)$

g_0 :	$\mapsto \mathbf{Ed}$	(d.h. $g_0 \in \mathbf{Ed}$)	mit	$g_0(s) = 0$	konstante Funktionen!
g_1 :	$\mapsto \mathbf{Ed}$	(d.h. $g_1 \in \mathbf{Ed}$)	mit	$g_1(s) = 1$	
g_{true} :	$\mapsto \mathbf{Ed}$	(d.h. $g_{true} \in \mathbf{Ed}$)	mit	$g_{true}(s) = \mathrm{T}$	
g_{false}:	$\mapsto \mathbf{Ed}$	(d.h. $g_{false} \in \mathbf{Ed}$)	mit	$g_{false}(s) = \mathrm{F}$	
g_{I} :	$\mapsto \mathbf{Ed}$	(d.h. $g_{\mathrm{I}} \in \mathbf{Ed}$)	mit	$g_{\mathrm{I}}(s) = s(\mathrm{I})$	

$g_\neg : \mathbf{Ed} \mapsto \mathbf{Ed}$ mit $g_\neg(e) = e'$, wobei $e'(s) = non(e(s))$

$g_= : \mathbf{Ed} \times \mathbf{Ed} \mapsto \mathbf{Ed}$ mit $g_=(e_1, e_2) = e$,

$$\text{wobei } e(s) = \begin{cases} \mathrm{T} \text{ , falls } e_1(s) = e_2(s) \in \mathbf{V}, \\ \mathrm{F} \text{ , falls } e_1(s), e_2(s) \in \mathbf{V}, e_1(s) \neq e_2(s), \\ \underline{\mathrm{err}} \text{ sonst.} \end{cases}$$

Beispiel zur „algebraischen Semantik“ (Forts.)

$g_+ : \mathbf{Ed} \times \mathbf{Ed} \mapsto \mathbf{Ed}$ mit $g_+(e_1, e_2) = e$,

wobei $e(s) = plus(e_1(s), e_2(s))$

$g_{:=} : \mathbf{Ide} \times \mathbf{Ed} \mapsto \mathbf{Cd}$ mit $g_{:=}(\mathrm{I}, e) = c$,

wobei $c(s) = s[e(s) / \mathrm{I}]$

Funktionsabänderung an genau einer Stelle

$g_{if} : \mathbf{Ed} \times \mathbf{Cd} \times \mathbf{Cd} \mapsto \mathbf{Cd}$ mit $g_{if}(e, c_1, c_2) = c$,

$$\text{wobei } c(s) = \begin{cases} c_1(s) & \text{, falls } e(s) = \mathrm{T}, \\ c_2(s) & \text{, falls } e(s) = \mathrm{F}, \\ \underline{\mathrm{err}} & \text{sonst.} \end{cases}$$

$g_{;} : \mathbf{Cd} \times \mathbf{Cd} \mapsto \mathbf{Cd}$ mit $g_{;}(c_1, c_2) = c$,

wobei $c(s) = c_2(c_1(s))$.

Beispiel zur „algebraischen Semantik“ (Forts.)

Damit haben wir

$$\underline{\text{SYN}} \;=\; [\mathbf{Ide}, \mathbf{Exp}, \mathbf{Com}; (f_p)_{p \in P}]$$

als syntaktische Algebra. Sie ergibt sich mit der allgemeinen Methode der Zuordnung der syntaktischen Algebra zur definierenden kontext-freien Grammatik. Wir erkennen, daß $\underline{\text{SYN}}$ wegen der syntaktischen Eindeutigkeit der Sprache selbst PEANO-Algebra ist. Und wir haben nun

$$\underline{\text{SEM}} \;=\; [\mathbf{Ide}, \mathbf{Ed}, \mathbf{Cd}; (g_p)_{p \in P}]$$

als semantische Algebra. Der eindeutig bestimmte Homomorphismus

$$h : \underline{\text{SYN}} \mapsto \underline{\text{SEM}}$$

mit $h = (h_I, h_e, h_c)$, wobei

$$h_I = \iota_{\mathbf{Ide}} \qquad \text{(Identität über Ide)}$$
$$h_e : \mathbf{Exp} \mapsto \mathbf{Ed}$$
$$h_c : \mathbf{Com} \mapsto \mathbf{Cd}\ ,$$

legt die Semantik der Sprache fest.

Beispiel zur „algebraischen Semantik“ (Forts.)

Beispiel zur Berechnung der Semantik: Syntax grün, Semantik blau

Programm

$$x := \underline{1} \; ; \; a := (a + x)$$

(a, x ∈ **Ide** vorausgesetzt)

Wir wenden den Homomorphismus $h : \underline{SYN} \mapsto \underline{SEM}$ an:

$$\begin{aligned}
& h(\, x := \underline{1} \; ; \; a := (a + x) \,) \\
= \; & h_c(\; f_{;}\,(x := \underline{1},\; a := (a + x)) \,) \\
= \; & h_c(\; f_{;}\,(f_{:=}\,(x\,,\, \underline{1}),\; f_{:=}\,(a,\; (a + x))) \,) \\
= \; & h_c(\; f_{;}\,(f_{:=}\,(x\,,\, f_1),\; f_{:=}\,(a,\; f_+(a, x)\,)) \,) \\
= \; & g_{;}(\; h_c(f_{:=}\,(x\,,\, f_1)\,),\; h_c(f_{:=}\,(a,\; f_+(a, x)\,)\,)\,) \\
= \; & g_{;}(\; g_{:=}\,(h_I(x), h_e(f_1)\,),\; g_{:=}\,(h_I(a)\,,\; h_e(f_+(a, x)\,))\,) \\
= \; & g_{;}(\; g_{:=}\,(x, g_1),\; g_{:=}\,(a\,,\; g_+\,(h_e(a), h_e(x)\,)\,)\;) \\
= \; & g_{;}(\; g_{:=}\,(x, g_1),\; g_{:=}\,(a\,,\; g_+\,(h_e(f_a), h_e(f_x)\,)\,)\;) \\
= \; & g_{;}(\; g_{:=}\,(x, g_1),\; g_{:=}\,(a\,,\; g_+\,(g_a, g_x)\,)\;) \;=\; c \in \mathbf{Cd}
\end{aligned}$$

Beispiel zur „algebraischen Semantik“ (Forts.)

Die Bedeutung des gegebenen Programms ist also die erhaltene „Belegungsänderungsfunktion“ c aus **Cd**.

Wegen $\mathbf{Cd} = \mathbf{S} \mapsto (\mathbf{S} \cup \{\underline{\text{err}}\})$ kann man die Wirkung auf eine gegebene Belegung s ausrechnen. Z.B. sei $s(\text{x}) = 0$, $s(\text{a}) = n$.

$c(s) = [g_{;}(\underbrace{g_{:=}(\text{x}, g_1)}_{c_1}, \underbrace{g_{:=}(\text{a}, g_{+}(g_{\text{a}}, g_{\text{x}}))}_{c_2})](s)$

$= g_{;}(c_1, c_2)(s) = c_2(c_1(s))$

mit

$c_1(s) = g_{:=}(\text{x}, g_1)(s) = s[g_1(s) / \text{x}] = s[1 / \text{x}] =_{df} s_1,$

d.h. $s_1(\text{x}) = 1$, $s_1(\text{a}) = n$,

$c(s) = c_2(s_1) = g_{:=}(\text{a}, g_{+}(g_{\text{a}}, g_{\text{x}}))(s_1) = s_1[w / \text{a}]$

mit $w = g_{+}(g_{\text{a}}, g_{\text{x}})(s_1) = plus(g_{\text{a}}(s_1), g_{\text{x}}(s_1)) = plus(s_1(\text{a}), s_1(\text{x}))$

$= plus(n, 1) = n + 1,$

d.h. $c(s) = s_1[w / \text{a}] = s_1[n + 1 / \text{a}] =_{df} s_2$,

d.h. $s_2(\text{x}) = 1$, $s_2(\text{a}) = n + 1$.

Wertberechnung und Substitution

Wir betrachten eine beliebige Ausdrucksalgebra

$$\mathcal{E} = [(E_s)_{s \in S}, (f_\omega)_{\omega \in \Omega}]$$

über dem Alphabet X als gegebene syntaktische Algebra, z.B. eine Algebra logischer Ausdrücke.

Zu einer vorgegebenen semantischen Algebra

$$\mathcal{A} = [(A_s)_{s \in S}, (g_\omega)_{\omega \in \Omega}]$$

und jeder $\mathcal{A}$-Belegung b von X, d.h. $b_s: X_s \mapsto A_s$, existiert nach dem Fortsetzungssatz eindeutig ein Homomorphismus b^*, der b fortsetzt.

b stellt eine ***Belegung der Variablen*** von X mit Werten aus A dar, b^* damit die ***Auswertung der Ausdrücke*** in dem semantischen Bereich $\mathcal{A}$ bei der Belegung b :

semantischer Homomorphismus

$b^* = wert(\text{--}, b) = wert_b$, d.h. $b^*(e) = wert(e, b) = wert_b(e)$ für $e \in E$

Nun wählen wir als „semantische Algebra“ wieder die Ausdrucksalgebra

$$\mathcal{E} = [(E_s)_{s \in S}, (f_\omega)_{\omega \in \Omega}]$$

selber. Damit haben wir einen Spezialfall:

Jede $\mathcal{E}$-Belegung s von X, $s : X \mapsto E$, ordnet den Variablen sortenrein Ausdrücke zu, stellt damit also eine sog. ***simultane Einsetzung*** von Ausdrücken für Variablen dar, die „Auswertung“ der Ausdrücke bei einer solchen ***Einsetzung*** ist durch deren homomorphe Fortsetzung $s^* : E \mapsto E$ gegeben!

s^* beschreibt die Wirkung der Einsetzung s auf Ausdrücke und heißt deshalb Substitutionsendomorphismus oder nur kurz ***Substitution***.

Substitutionsendomorphismus

$s^* = sub(\text{--}, s) = sub_s$, d.h. $s^*(e) = sub(e, s) = sub_s(e)$ für $e \in E$

Substitution ist ein Spezialfall der Semantik!

Wertänderung bei Substitution:

$\mathcal{E} = [E, F]$ sei eine Σ-Ausdrucksalgebra über X und $\mathcal{A} = [A, G]$ eine beliebige Σ-Algebra. $b : X \mapsto A$ sei eine $\mathcal{A}$-Belegung von X und $s : X \mapsto E$ bezeichne eine Einsetzung. Ist dann $b' : X \mapsto A$ die wie folgt durch die Einsetzung s festgelegte $\mathcal{A}$-Belegung mit

$$b'(x) = wert(s(x), b) \qquad \text{für alle } x \in X ,$$

so gilt für alle $e \in E$:

$$wert(sub(e, s), b) = wert(e, b') .$$

Beweis:

Nach Voraussetzung ist $b' = b^* \circ s$. Außerdem ist $b^* \circ s^*$ Homomorphismus von $\mathcal{E}$ in $\mathcal{A}$ und es gilt

$$(b^* \circ s^*)|_X = b^* \circ s^*|_X = b^* \circ s = b' .$$

Wegen der Eindeutigkeit der homomorphen Fortsetzung von b' ist damit $b'^* = b^* \circ s^*$, was behauptet wurde.

Kongruenzen

Definition "Kongruenzrelation"

Eine **Σ-*Kongruenz*** R über einer Σ-Algebra $\mathcal{A} = [A, F]$ ist eine Familie $R = (R_s)_{s \in S}$ von Äquivalenzrelationen R_s in A_s mit der Eigenschaft der ***Kompatibilität***,
d.h. für alle $\omega \in \Omega$ mit $\alpha(\omega) = (s_1, s_2, ..., s_n, s)$ gilt:
wenn

$$(a_i, b_i) \in R_{s_i} \qquad \text{für } 1 \leq i \leq n,$$

so ist auch

$$(f_\omega(a_1, a_2, ..., a_n),\ f_\omega(b_1, b_2, ..., b_n)) \in R_s .$$

Äquivalenzklasse von $a \in A_s$ bzgl. R_s :

$$[a]_{R_s} \text{ oder } [a]_R \text{ oder } [a]_s \text{ oder } [a]$$

Familie der Mengen der Äquivalenzklassen:

$$A/R = (A_s/R_s)_{s \in S}$$

mit $A_s/R_s = \{\ [a]_{R_s} \mid a \in A_s\ \}$.

Durch f_ω ***induzierte Operationen*** f_ω^* über den Äquivalenzklassen:

$$f_\omega^*([a_1]_{R_{s1}}, [a_2]_{R_{s2}}, ..., [a_n]_{R_{sn}}) =_{df} [f_\omega(a_1, a_2, ..., a_n)]_{R_s}$$

Wenn die R_s bzgl. den f_ω kompatibel sind, so ist diese Definition repräsentantenunabhängig, d.h. die f_ω^* sind tatsächlich Operationen über A/R.

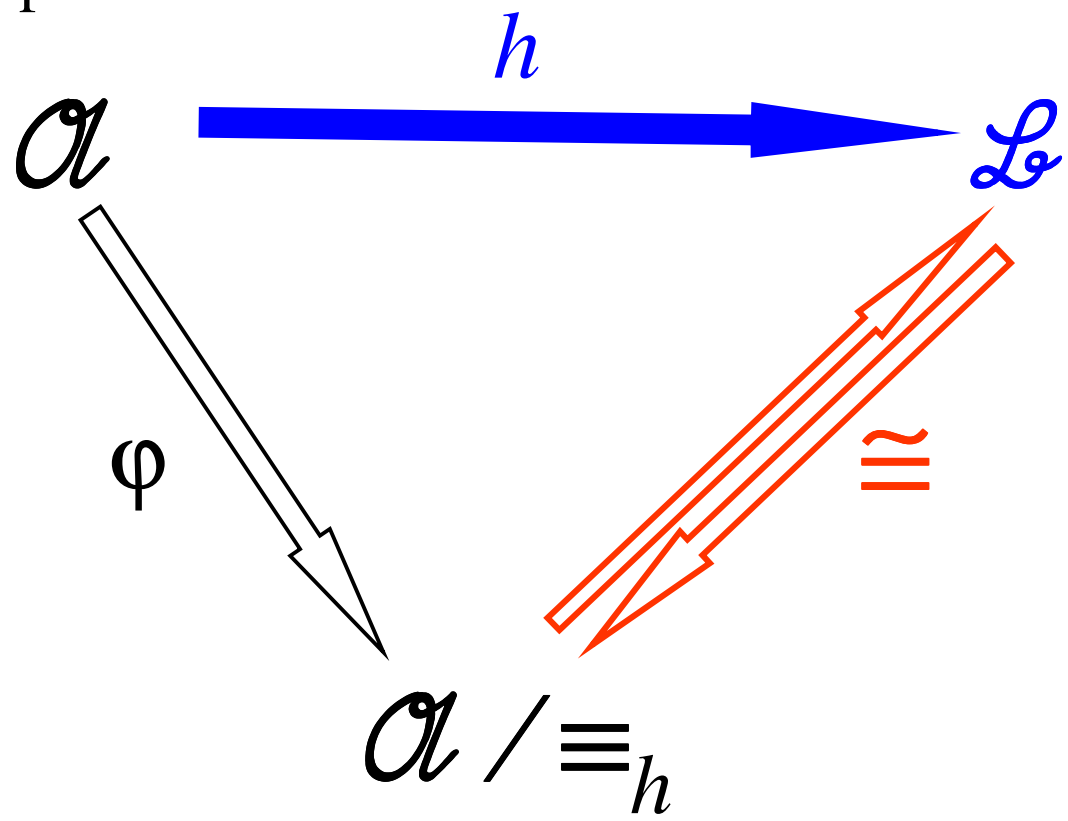
R sei eine Σ-Kongruenz über der Σ-Algebra $\mathcal{A} = [A, F]$. Die Σ-Algebra $[A/R, (f_\omega^*)_{\omega \in \Omega}]$ mit den durch f_ω induzierten Operationen f_ω^* heißt die ***Faktoralgebra*** $\mathcal{A}/R$ der Σ-Algebra $\mathcal{A}$ nach der Σ-Kongruenz R.

Homomorphiesatz:

Es sei $\mathcal{A} = [(A_s)_{s \in S}, (f_\omega)_{\omega \in \Omega}]$ eine Σ-Algebra und $\equiv$ eine Σ-Kongruenz über $\mathcal{A}$. Dann ist die Abbildung $\varphi: A \mapsto A/\equiv$ mit $\varphi(a) = [a]_\equiv$ ein Σ-Homomorphismus von $\mathcal{A}$ nach $\mathcal{A}/\equiv$.

Ist h: $\mathcal{A} \mapsto \mathcal{L}$ ein weiterer Σ-Homomorphismus, und ist $\equiv_h$ die Bildgleichheit bzgl. h, so ist $\equiv_h$ eine Σ-Kongruenz über $\mathcal{A}$; und wenn h zusätzlich surjektiv ist, dann sind $\mathcal{A}/\equiv_h$ und $\mathcal{L}$ isomorph.

Homomorphiesatz:



Die Abbildung φ heißt auch natürliche Abbildung von $\equiv$.

$\varphi = nat(\equiv)$.

1. Jede Faktoralgebra ist homomorphes Bild der Ausgangsalgebra, und umgekehrt ist jedes homomorphe Bild einer Algebra isomorph zu einer Faktoralgebra dieser Algebra.
2. Damit kann *jedes* homomorphe Bild einer Algebra völlig „innerhalb“ der Algebra gefunden werden.
3. Eine Faktoralgebra ist festgelegt durch die Algebra und eine Kongruenz. Nach dem Homomorphiesatz kann das Studium der Homomorphismen gleichwertig ersetzt werden durch das Studium der Kongruenzen!
4. Semantik ist ein homomorphes Bild SEM der syntaktischen Algebra SYN. Damit gilt

$$\underline{\text{SEM}} \cong \underline{\text{SYN}} \,/\, R$$

für eine eindeutig bestimmte Kongruenz R.

- **Semantik entspricht also einer Kongruenzrelation über der Syntax!**

Faktorisierung des Erzeugendensystems:

Es sei $\mathcal{A}$ eine Σ-Algebra und R eine Σ-Kongruenz über $\mathcal{A}$. Wenn X ein Erzeugendensystem von $\mathcal{A}$ ist, dann ist X/R ein Erzeugendensystem der Faktoralgebra $\mathcal{A}/R$.

Beweis:

Wir haben $A = [X]_{\mathcal{A}}$. $\varphi = nat(R)$ ist Homomorphismus von $\mathcal{A}$ **auf** $\mathcal{A}/R$. Also gilt $A/R = \varphi(A) = \varphi([X]_{\mathcal{A}}) = [\varphi(X)]_{\mathcal{A}/R}$ entsprechend dem Satz vom homomorphen Bild der Hülle.

Weiter ist $\varphi(X) = (\varphi_s(X_s))_{s \in S}$ mit

$\varphi_s(X_s) = \{ \varphi_s(x) \mid x \in X_s \} = \{ [x]_{R_s} \mid x \in X_s \} = X_s/R_s$,

womit $A/R = [X/R]_{\mathcal{A}/R}$ bewiesen ist.

Gleichungskalkül

Gruppe ist eine Algebra $\mathcal{G} = [G ; \circ , inv, e]$, für die gilt:

$$(x \circ y) \circ z = x \circ (y \circ z)$$
$$x \circ e = e \circ x = x$$
$$x \circ inv(x) = inv(x) \circ x = e$$

für alle Gruppenelemente $x, y, z \in G$.

Dann haben wir aber z.B. auch

$$inv(e) = e$$
$$y \circ (inv(y) \circ x) = x$$
$$inv(x \circ y) = inv(y) \circ inv(x)$$

~~~~~~~~~~~~~~

? definierende Gleichungen ⇔ gültige Gleichungen ?

In $\underline{T_\Sigma(X)}$ gibt es keinerlei Gesetze - ***„anarchische Algebra“***
~~~~~~~~~~~~~~

Eine **Σ-*Gleichung*** der Sorte s über dem Variablensystem $X = (X_s)_{s \in S}$ ist ein Paar (e_1 , e_2) von Ausdrücken aus $T_\Sigma(X)_s$.
Ein **Σ-*Gleichungssystem*** E über X ist eine S-sortige Familie von Mengen von Σ-Gleichungen.
Für $(e_1 , e_2) \in E$ schreibt man oft $e_1 = e_2$.

Wesentliche Voraussetzung für alle folgenden Überlegungen: alle betrachteten Algebren haben ***nur nichtleere*** Trägermengen

vgl. ***J.A. Goguen, J. Meseguer***,
Completeness of many-sorted equational logic [**1981**]

Es sei $\mathcal{A}$ eine Σ-Algebra. Die $\mathcal{A}$-Belegung b von X ***erfüllt*** die Σ-Gleichung (e_1 , e_2) über X, kurz

$$b \; \mathbf{erf}_{\mathcal{A}} \; e_1 = e_2 \; ,$$

gdw. für die homomorphe Fortsetzung b^* von b auf $T_\Sigma(X)$ gilt:

$$b^*(e_1) = b^*(e_2).$$

Die Gleichung $e_1 = e_2$ heißt ***in*** $\mathcal{A}$ ***gültig*** oder eine ***Identität in*** $\mathcal{A}$, kurz

$$\mathcal{A} \models e_1 = e_2 \; ,$$

gdw. für alle $\mathcal{A}$-Belegungen b von X gilt

$$b \; \mathbf{erf}_{\mathcal{A}} \; e_1 = e_2 \; .$$

Ein Σ-Gleichungssystem E heißt ***gültig in*** $\mathcal{A}$ oder $\mathcal{A}$ heißt ein ***Modell für*** E, kurz

$$\mathcal{A} \models E ,$$

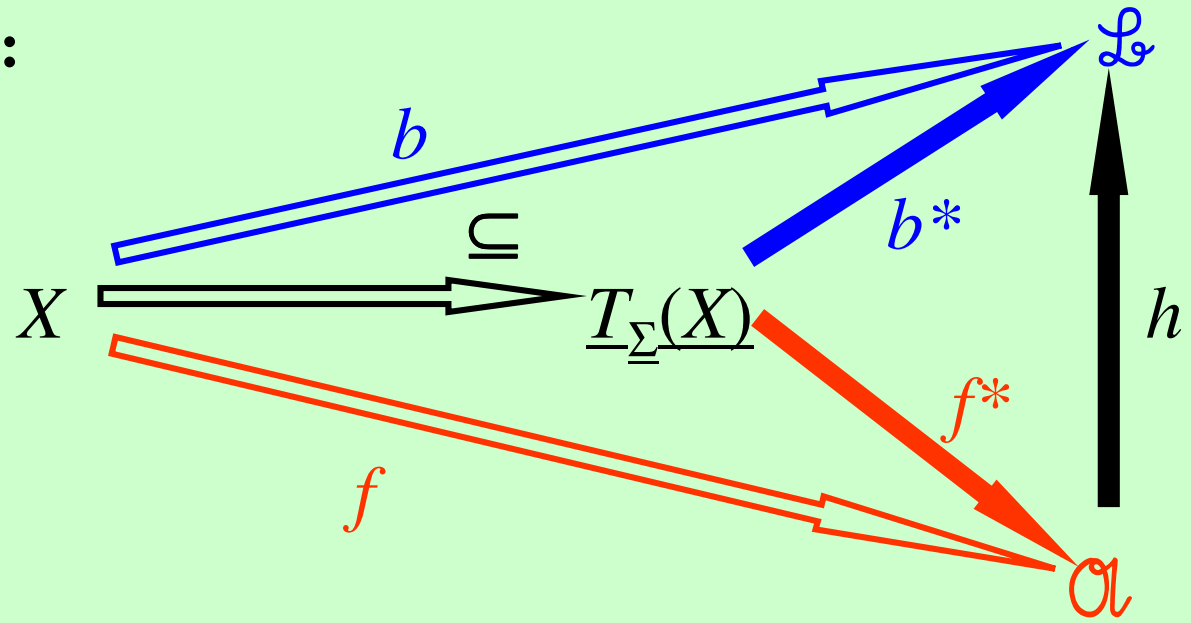
gdw. alle Gleichungen in $\mathcal{A}$ gültig sind.
$\mathcal{A}$ heißt dann auch eine (Σ,E)***-Algebra***.
Die Klasse aller (Σ,E)-Algebren heißt die durch E definierte ***Varietät*** bzw. eine ***gleichungsdefinierbare Klasse***.
Wir bezeichnen sie im folgenden durch $\underline{\mathbf{Alg}_{\Sigma,E}}$.

Die Klasse **aller** Σ-Algebren bezeichnen wir im folgenden analog durch $\underline{\mathbf{Alg}_{\Sigma}}$.

Homomorphe Bilder von Gleichungsmodellen:

Es sei E ein Σ-Gleichungssystem über X, und $\mathcal{A}$ eine (Σ,E)-Algebra. Dann ist jedes homomorphe Bild $\mathcal{B}$ von $\mathcal{A}$ ebenfalls Modell von E.

Beweisidee:



Jedes b ist $b = h \circ f$ für geeignetes f $\Rightarrow$ $b^* = h \circ f^*$.

Für alle $(e_1, e_2) \in E$ ist $f^*(e_1) = f^*(e_2)$ $\Rightarrow$ $b^*(e_1) = b^*(e_2)$.

1. BIRKHOFF-Theorem:

Georg David Birkhoff (1884-1944)

Varietäten lassen sich als **diejenigen** Algebrenklassen charakterisieren, die gegen ***Unteralgebrenbildung***, ***homomorphe Bilder*** und ***Bildung des direkten Produkts*** abgeschlossen sind.

Der Abschluß der Algebrenklasse $\mathfrak{K}$ gegenüber den drei genannten Operatoren über Algebrenklassen ist notwendige und hinreichende Bedingung dafür, daß $\mathfrak{K}$ Modellklasse eines geeigneten Gleichungssystems, also durch Gleichungen axiomatisierbar, ist.

E sei ein Σ-Gleichungssystem über X.
Eine Σ-Gleichung $(e_1 , e_2) \in T_\Sigma(X) \times T_\Sigma(X)$ ***entsteht durch Einsetzung aus*** E gdw. es $e_1', e_2' \in T_\Sigma(X)$ und eine Einsetzung $s: X \mapsto T_\Sigma(X)$ gibt, so daß
$e_1 = sub(e_1', s)$, $e_2 = sub(e_2', s)$ und $(e_1', e_2') \in E$.
$\mathbf{S}(E)$ bezeichne das System aller Gleichungen, das aus E durch Einsetzung entsteht, und heißt der ***stabile Abschluß*** von E.

Einsetzungsregel:
$\mathcal{A}$ sei eine (Σ,E)-Algebra. Dann gilt auch $\mathcal{A} \models \mathbf{S}(E)$.

m.a.W.: In Modellen von E sind auch die durch Einsetzung aus E entstehenden Gleichungen gültig.

Definition "erzeugte Kongruenz"

Die Relation

$\equiv_E = \bigcap\{\ R \mid \mathbf{S}(E) \subseteq R$ und R ist Σ-Kongruenz über $\underline{T_\Sigma(X)}\ \}$

heißt ***die von der Relation*** $\mathbf{S}(E)$ ***erzeugte Kongruenz*** über $\underline{T_\Sigma(X)}$.

$\equiv_E$ ist offensichtlich Kongruenz über $\underline{T_\Sigma(X)}$, und zwar die kleinste, die $\mathbf{S}(E)$ umfaßt.

Definition "syntaktische Äquivalenz"

Die von der Relation $\mathbf{S}(E)$ erzeugte Kongruenz $\equiv_E$ heißt auch ***syntaktische Äquivalenz*** der Klasse $\underline{\mathbf{Alg}_{\Sigma,E}}$ aller (Σ,E)-Algebren.

Ableitungsbegriff für die syntaktische Äquivalenz:

Anfang

0. Für $(e_1 , e_2) \in \mathbf{S}(E)$ gilt $\vdash_E e_1 = e_2$.

reflexiver Abschluß

1. Für alle $e \in T_\Sigma(X)$ gilt $\vdash_E e = e$.

symmetrischer Abschluß

2. Wenn $\vdash_E e_1 = e_2$, so gilt $\vdash_E e_2 = e_1$.

transitiver Abschluß

3. Wenn $\vdash_E e_1 = e_2$ und $\vdash_E e_2 = e_3$, so $\vdash_E e_1 = e_3$.

kompatibler Abschluß

4. Für alle $\omega \in \Omega$ mit $\alpha(\omega) = (s_1, s_2, ..., s_n, s)$ gilt:
   Wenn $\vdash_E e_i = e_i'$ und $e_i, e_i' \in T_\Sigma(X)_{s_i}$ für $1 \le i \le n$, so
   $\vdash_E \omega e_1 e_2 \ldots e_n = \omega e_1' e_2' \ldots e_n'$.

Weitere Gründe für $\vdash_E e_1 = e_2$ gibt es nicht.

$e_1 \equiv_E e_2$ gilt genau dann, wenn $\vdash_E e_1 = e_2$.

Man erhält die syntaktische Äquivalenz $\equiv_E$ auch als direkte Anwendung der Hüllenoperatoren ***reflexiver Abschluß***, ***stabiler Abschluß***, ***symmetrischer Abschluß***, ***transitiver Abschluß*** und ***kompatibler Abschluß*** (*in geeigneter Reihenfolge!*) auf das Gleichungssystem E als Ausgangsrelation.

Widerspruchsfreiheit der syntaktischen Äquivalenz:
E sei ein Σ-Gleichungssystem und es gelte $e_1 \equiv_E e_2$.
Dann gilt für jede (Σ,E)-Algebra $\mathcal{A}$ auch $\mathcal{A} \models e_1 = e_2$.

Der **Beweis** kann leicht induktiv über eine der Verfahren zur Erzeugung der syntaktischen Äquvalenz geführt werden.
Wesentliche Grundlage ist die obige Einsetzungsregel.

Das S-sortige System von Äquivalenzrelationen $\approx_E$ in $T_\Sigma(X)$, das definiert ist durch

$$e_1 \approx_E e_2$$

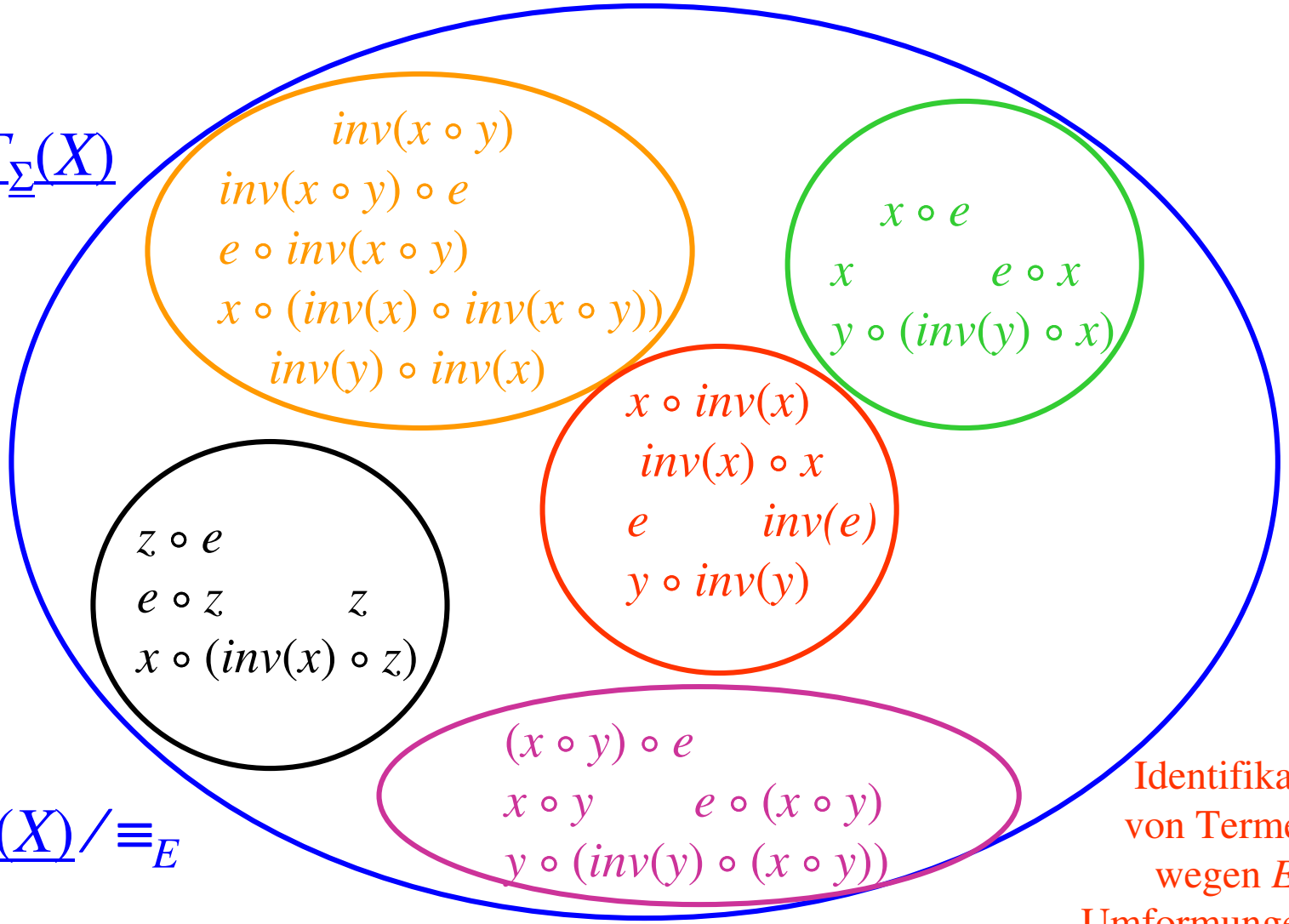
gdw. für alle (Σ,E)-Algebren $\mathcal{A}$ gilt $\mathcal{A} \models e_1 = e_2$,
heißt ***semantische Äquivalenz*** der Klasse $\underline{\mathbf{Alg}_{\Sigma,E}}$ aller (Σ,E)-Algebren.

Aus der syntaktischen Äquivalenz folgt stets die semantische Äquivalenz:

$$e_1 \equiv_E e_2 \quad \Rightarrow \quad e_1 \approx_E e_2 \,.$$

? **Existieren freie Algebren in $\underline{\mathbf{Alg}_{\Sigma,E}}$** ?

Zu Gruppe $\mathfrak{G} = [G ; \circ , inv, e]$ gehören Gleichungen E



Identifikation von Termen wegen E „Umformungen“

E sei ein Σ-Gleichungssystem über X.
Dann wird gesetzt:

$$\underline{T_{\Sigma,E}(X)} =_{df} \underline{T_{\Sigma}(X)}/\equiv_E \ .$$

$\underline{T_{\Sigma,E}(X)}$ ist eine (Σ,E)-Algebra.

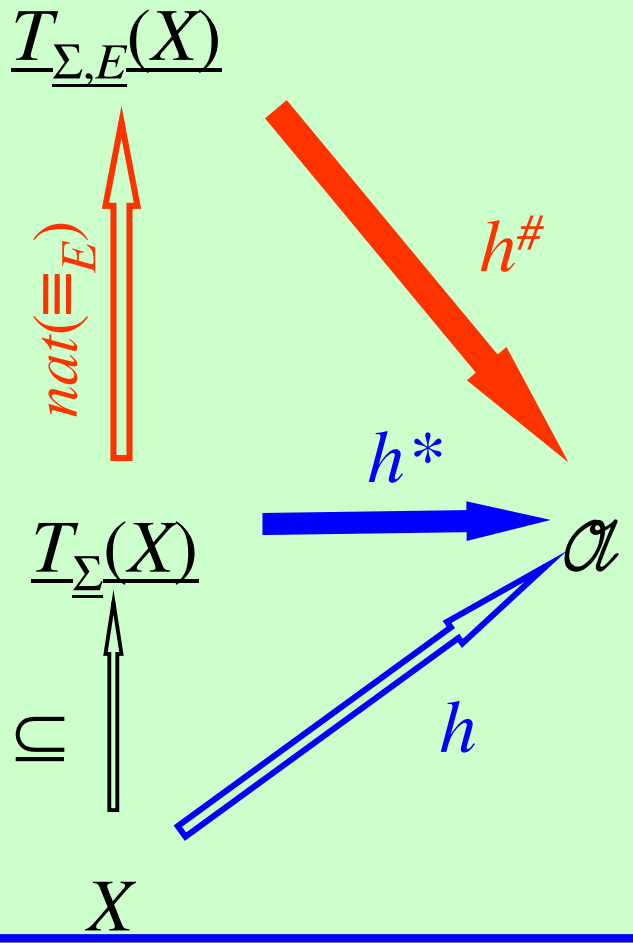
Vollständigkeit des Gleichungskalküls / 2. BIRKHOFF-Theorem:
Die syntaktische Äquivalenz $\equiv_E$ fällt mit der semantischen Äquivalenz $\approx_E$ zusammen.

Beweis: $e_1 \approx_E e_2$, also speziell $\underline{T_{\Sigma,E}(X)} \models e_1 = e_2$. $\Rightarrow$ $b^*(e_1) = b^*(e_2)$ für alle $\underline{T_{\Sigma,E}(X)}$-Belegungen b, auch für φ' mit $\varphi'(x)=[x]_{\equiv_E}$.
Hom. Forts. von φ' ist die natürliche Abb. $\varphi = nat(\equiv_E)$ **Homomorphiesatz!**
Also $\varphi(e_1) = \varphi(e_2)$ bzw. $[e_1]_{\equiv_E} = [e_2]_{\equiv_E}$ $\Rightarrow$ $e_1 \equiv_E e_2$.

Fortsetzungssatz:

Es sei E ein Σ-Gleichungssystem über X, und $\mathcal{A}$ eine (Σ,E)-Algebra. Dann existiert zu jeder $\mathcal{A}$-Belegung h von X genau ein Homomorphismus $h^{\#}$ von $\underline{T_{\Sigma,E}(X)}$ in $\mathcal{A}$, für den $h^{\#}([x]_{\equiv_E}) = h(x)$ für alle $x \in X$ gilt.

Beweis:



Man überlege sich den variablenfreien Spezialfall

$$\underline{T_{\Sigma,E}} = \underline{T_\Sigma}/\equiv_E = \underline{T_{\Sigma,E}(\emptyset)}$$

Abstrakte Datentypen

Datentyp ≙ Zusammenfassung von Mengen von "Werten" mit auf ihnen erklärten Operationen

= heterogene Algebra

Bei Verwendung im Softwareentwurf ***Spezifikation*** erforderlich:

- Festlegung von Eigenschaften
  - nur wesentliche Eigenschaften, nur Prinzip
  - keine Details, unabhängig von konkreter Implementation
  - vollständige Beschreibung oft unmöglich - auch nicht erwünscht!

Abstrakter Datentyp ≙ unvollständige Charakterisierung eines Datentyps

⇒ wird von *Klasse* von konkreten Datentypen erfüllt

Damit gehören zu einem abstrakten Datentyp alle diejenigen konkreten Datentypen, die der Beschreibung bzw. Charakterisierung genügen. $\Rightarrow$ ***vorläufige*** Definition:

Ein ***abstrakter Datentyp*** (**ADT**) ist eine Klasse von Algebren (Datentypen).

Beispiel ADT "geordnete endliche Menge" **ORD**:

endliche Menge M mit Ordnung $\preccurlyeq$ auf M

$\preccurlyeq$ - als Operation : $\preccurlyeq : M \times M \mapsto$ **bool** mit **bool**=$\{true, false\}$

ORD - Klasse von Algebren [M , **bool** ; $\preccurlyeq$] :

- M = { 0, 1, 2, 3, 4, 5} mit $\leq$
- M = { -5, -3, -1, 1, 3, 5, 7} mit $\geq$
- irgendeine endliche Menge reeller Zahlen mit $\leq$
- eine bel. endliche Menge von Wörtern mit einer lexikographischen Ordnung
- Menge von Institutsangehörigen mit Ordnung entspr. Gehalt

Beispiel ADT "Zähler modulo 8" **COUNT8**:

8-elementige Menge N mit 2 Operationen *inc* und *reset* auf N

$inc : N \mapsto N$ (Weiterzählen)

$reset : \mapsto N$ (Rücksetzen/Löschen)

mit den Eigenschaften:

- 8-malige Anwendung von *inc* nach *reset* führt zum selben Resultat wie *reset*
- alle möglichen Werte sind von *reset* aus mittels *inc* zu erreichen

COUNT8 - Klasse von Algebren [N ; *inc*, *reset*] :

dazu gehören zum Beispiel:

- $N = \{0, 1, 2, 3, 4, 5, 6, 7\}$ mit

$$inc(n) = \begin{cases} n+1 & \text{für } 0 \le n \le 6 \\ 0 & \text{für } n = 7 \end{cases} ,$$

$reset = 0$.

Beispiel ADT "Zähler modulo 8" **COUNT8**:

dazu gehören aber z.B. auch:

- $N = \{0, -1, -2, -3, -4, -5, -6, -7\}$ mit

$$inc(n) = \begin{cases} n - 1 & \text{für } -6 \le n \le 0 \\ 0 & \text{für } n = -7 \end{cases},$$

$$reset = 0 .$$

- $N = \{-10, -8, -6, -4, -2, 0, 2, 4\}$ mit

$$inc(n) = \begin{cases} -10 & \text{für } n = 4 \\ n + 2 & \text{sonst} \end{cases},$$

$$reset = -10 .$$

ÜA: Man überlege sich einen weiteren Datentyp zu **COUNT8**, dessen Elemente keine Zahlen sind.

Es ist leicht zu sehen, daß alle Datentypen, die zu **COUNT8** gehören, zueinander ***isomorph*** sind.
ÜA: Man beweise dies.

Zu **ORD** hingegen gehören sehr unterschiedliche Datentypen. Da die Unterschiede insbesondere auch die Anzahl der Elemente betreffen, sind die Datentypen i.a. ***nichtisomorph*** zueinander.

COUNT8 heißt deshalb ***monomorpher*** abstrakter Datentyp, **ORD** ist ein ***polymorpher*** abstrakter Datentyp.

Definition "monomorpher bzw. polymorpher ADT"

Ist ein abstrakter Datentyp eine Isomorphieklasse von Algebren, so heißt der abstrakte Datentyp monomorph, sonst polymorph.

Spezifikationen

Die Festlegung der Eigenschaften von abstrakten Datentypen nennt man Spezifikation. Im Softwarelebenszyklus gibt es einige Phasen, in denen ***Spezifikationen*** notwendig oder erwünscht sind:

- Problemspezifikation
- Systemspezifikation
- Programmspezifikation
- Datentypspezifikation

Spezifikationen im Software-Lebenslauf

1. Problemspezifikation

- Leistungsbeschreibung, Pflichtenheft, Anforderungsdefinition, Vertragsgrundlage
- verbale Formulierung, Basis für formale Spezifikation

2. Systemspezifikation

- Ergebnis des Systementwurfs, Grundlage der Implementation, formale und halbformale Methoden

3. Programmspezifikation

- nach Modularisierung des Systems Beschreibung der Anforderungen an die Modulprogramme

4. Datentypspezifikation

- nützlich für Programmentwurf als Algorithmen über Datenstrukturen

Aufgaben einer Spezifikationen

1.) Korrektheit der Spezifikation

- Übereinstimmung mit Absicht, gemeinten Eigenschaften
- Konsistenz
- hinreichende Vollständigkeit, Ausschluß unerwünschter Realisierungen

2.) Korrektheit der Implementierung

- Verifikation der Implementierung bzgl. Spezifikation

3.) Automatische Entwurfshilfen

- Entwicklung und Einsatz ermöglichen

Anforderungen an eine Spezifikationen

a) Modularität

- Verständlichkeit
- Überschaubarkeit

b) Eindeutigkeit

- präzise Semantik
- eindeutige Interpretierbarkeit des Spezifikationstextes

c) Abstraktion

- keine zeitige Festlegung bestimmter Implementationen
- keine irrelevanten Details

Spezifikationsmethoden

A) Verbale Spezifikation

B) Formale Spezifikation

Imperative Methoden	Exemplarische Methoden
Applikative Methoden	Axiomatische Methoden

4 Methoden durch Kombination der Merkmale

1. Imperative Methoden

Benutzung von Mitteln höherer Programmiersprachen

(a) exemplarische imperative Spezifikation

- Datenelemente werden analog wie in Programmiersprachen dargestellt: z.B. Keller als **array**
- Operationen als Prozeduren:

  z.B. *push* als Funktionsprozedur, die Kellerarray um ein Element erweitert

Nachteil:

- Spezifikation entspricht eher einer konkreten Implementierung
- nicht abstrakt genug!

(b) axiomatische imperative Spezifikation

- Benutzung einer Programmlogik

  z.B. sehen die Spezifikationsformeln wie folgt aus:

  $\{p\}S\{q\}$

  S - Programmteil,

  p, q - Formeln eines geeigneten Logikkalküls

- Operationen (der Datentypen) werden nicht "programmiert", sondern durch ihre Vor- und Nachbedingungen axiomatisch beschrieben

  $\triangleq$ *abstrakte Prozedur* (nur durch ihre Eigenschaften gegeben)

  z.B.:

  $\{\mathbf{true}\}\ push(s, x)\ \{s \neq \Lambda \wedge top(s) = x\}$

  $\{s = s'\}\ push(s, x); pop(s)\ \{s = s'\}$

(b) axiomatische imperative Spezifikation (Forts.)

Vorteil:

- guter Abstraktionsgrad
- erfüllt die Anforderungen an Spezifikationen

Nachteil:

- imperative Methoden generell nicht adäquat für die Beschreibung von Datentypen (Algebren!), besser geeignet für Speichertransformationen wie z.B. bei Datenbanken

Solche axiomatischen imperativen Spezifikationen sind deshalb (Speichertransformation = Zustandsänderung) auch geeignet für die Semantikbeschreibung imperativer Programmiersprachen:
Axiomatische Semantik - *Grundlage für Programmverifikation!*

2. Applikative Methoden

(a) exemplarische applikative Spezifikation

- Konzepte wie in der denotationalen Semantik
- Angabe eines mathematischen Modells für Datenelemente und Operationen

z.B. **stack = char*** - "Zeichenkeller"

$$= \{(z_1, z_2, \ldots, z_n) \mid n \in \mathbf{Nz}, z_i \in \mathbf{char}\} \cup \{\Lambda\}, \quad \Lambda = ()$$

$$push((z_1, z_2, \ldots, z_n), x) = (x, z_1, z_2, \ldots, z_n)$$

$$pop((z_1, z_2, \ldots, z_n)) = \begin{cases} (z_2, z_3, \ldots, z_n) \text{ , falls } n > 0 \\ \text{undefiniert, falls } n = 0 \end{cases}$$

$$top((z_1, z_2, \ldots, z_n)) = \begin{cases} z_1 \text{ , falls } n > 0 \\ \text{undefiniert, falls } n = 0 \end{cases}$$

(a) exemplarische applikative Spezifikation (Forts.)

Vorteil:

- guter Abstraktionsgrad
- gute Verständlichkeit

Nachteil:

- Generierung automatischer Entwurfshilfen problematisch
- Unklar: andere (mathematische) Beschreibung äquivalent ?
  d.h. gehört anderes Modell zum selben ADT i.a. nicht zu beantworten

Ansatz evtl. noch brauchbar für Definition monomorpher ADT, für Spezifikation von polymorphen ADT i.a. ungeeignet.

(b) axiomatische applikative Spezifikation

Eigenschaften des ADT als Formeln einer geeigneten logischen Sprache formulieren

z.B. ADT „Kellerspeicher über **char**“ ist Menge **stack** mit folgenden Eigenschaften:

i. $\Lambda \in \textbf{stack}$

$\forall s \forall x (s \in \textbf{stack} \wedge x \in \textbf{char} \rightarrow push(s, x) \in \textbf{stack})$

$\forall s (s \in \textbf{stack} \wedge s \neq \Lambda \rightarrow pop(s) \in \textbf{stack}$

$\forall s (s \in \textbf{stack} \wedge s \neq \Lambda \rightarrow top(s) \in \textbf{char})$

$\forall s (s \in \textbf{stack} \rightarrow empty(s) \in \{\underline{\text{true}}, \underline{\text{false}}\})$

ii. $\forall s \forall x (s \in \textbf{stack} \wedge x \in \textbf{char} \rightarrow push(s, x) \neq \Lambda)$

$\forall s \forall x \forall y (s \in \textbf{stack} \wedge x, y \in \textbf{char} \rightarrow$

$(\ push(s, x) = push(s, y) \rightarrow x = y\ )\ )$

$\forall s_1 \forall s_2 \forall x (s_1, s_2 \in \textbf{stack} \wedge x \in \textbf{char} \rightarrow$

$(\ push(s_1, x) = push(s_2, x) \rightarrow s_1 = s_2\ )\ )$

(b) axiomatische applikative Spezifikation (Forts.)

iii. $\forall s \forall x (s \in \mathbf{stack} \wedge x \in \mathbf{char} \rightarrow$
$top(push(s, x)) = x \wedge pop(push(s, x)) = s)$
$\forall s \forall x (s \in \mathbf{stack} \wedge x \in \mathbf{char} \rightarrow empty(push(s, x)) = \underline{\text{false}})$
$empty(\Lambda) = \underline{\text{true}}$

iv. $\forall \mathbf{M} (\mathbf{M} \subseteq \mathbf{stack} \wedge \Lambda \in \mathbf{M} \wedge$
$\forall s \forall x (s \in \mathbf{M} \wedge x \in \mathbf{char} \rightarrow push(s, x) \in \mathbf{M})$
$\rightarrow \mathbf{M} = \mathbf{stack})$

Man vergleiche diese Axiome mit den PEANO-Axiomen für die natürlichen Zahlen bzw. mit den verallgemeinerten PEANO-Axiomen in der Definition der PEANO-Algebra!

Das Axiom iv. ist nicht im PK 1. Stufe formulierbar!
Für automatische Entwurfshilfen Beschränkung auf PK1 ratsam!

(b) axiomatische applikative Spezifikation (Forts.)

Vorteil:

- hoher Abstraktionsgrad
- tatsächlich werden - wie gewünscht - nur Eigenschaften beschrieben

Nachteil:

- Abstraktion auch problematisch: Keinerlei Ansatz für Art der Implementation erkennbar
- Sprache der Prädikatenlogik in ihrer Allgemeinheit zu kompliziert (z.B. für automatische Entwurfshilfen)

(b) axiomatische applikative Spezifikation (Forts.)

Fragen:

- Ist die Spezifikation ***widerspruchsfrei***?
  Existiert überhaupt ein solcher Datentyp, d.h. ein Modell

- Ist die Spezifikation ***vollständig***?
  Beschreibt sie den gewünschten ADT genau genug?
  Oder lassen sich unerwünschte Modelle finden?
  Lassen sich alle gewünschten Eigenschaften ableiten?

Im allgemeinen sind diese Fragen schwer oder gar nicht zu beantworten!

$\Rightarrow$ Ausdrucksmittel für die Axiome beschränken!

(b) axiomatische applikative Spezifikation (Forts.)

Termgleichungen sind die ***einfachsten Formeln*** des PK.

Beschränkung auf Axiome der Gestalt:

$$\forall x_1 \forall x_2 \ldots \forall x_n \, (t_1 = t_2)$$

mit $X = \{x_1, x_2, \ldots, x_n\}$ und $t_1, t_2 \in T_\Sigma(X)$

Vorteil: Es existiert eine gut bekannte, ausgearbeitete Theorie.
Frage der Widerspruchsfreiheit sofort beantwortbar!

$\Rightarrow$ Jede solche Spezifikation besitzt Modelle,
nämlich die entsprechende Klasse gleichungsdefinierter Algebren $\underline{\mathbf{Alg}}_{\Sigma,E}$

$\Rightarrow$ *Spezialform* der axiomatischen applikativen Spezifikation:
Algebraische Gleichungsspezifikation

Gleichungsspezifikation abstrakter Datentypen

Definition "Spezifikation"

Eine (***Gleichungs***-)***Spezifikation*** ist ein Paar

$$\underline{\text{SPEC}} = (\Sigma, E),$$

wobei Σ eine Signatur und E ein Σ-Gleichungssystem ist.

Die Spezifikation wird erfüllt von allen **Σ-**Algebren, in denen das Gleichungssystem E gültig ist, also von allen (Σ, E)-Algebren.
Damit bestimmt eine Spezifikation SPEC eine ganze Klasse von Algebren: $\underline{\mathbf{Alg}_{\Sigma,E}}$ bzw. $\underline{\mathbf{Alg}_{\text{SPEC}}}$.
Zur Angabe von Spezifikationen benutzen wir eine einfache sich selbst erklärende Sprache.

Beispiele für Spezifikationen

1.) **group** = **sorts** G

oprs $\bullet : G, G \mapsto G$

$e : \mapsto G$

$i : G \mapsto G$

var $x, y, z : G$

eqns $(x \bullet y) \bullet z = x \bullet (y \bullet z)$

$e \bullet x = x$

$i(x) \bullet x = e$

end group

Modelle dieser Spezifikation sind alle Gruppen.
Diese Spezifikation beschreibt die Klasse **Alg** $_{\textbf{group}}$.

Beispiele für Spezifikationen (Forts.)

2.) **stack** =

sorts stack, char, bool

oprs *push* : stack, char $\mapsto$ stack
top : stack $\mapsto$ char
pop : stack $\mapsto$ stack
clear : $\mapsto$ stack
empty: stack $\mapsto$ bool
true : $\mapsto$ bool
false : $\mapsto$ bool

var s : stack ; x : char

eqns $empty(push(s, x)) = false$
$empty(clear) = true$
$top(push(s, x)) = x$
$pop(push(s, x)) = s$

end stack

Das 1. Beispiel zeigt, daß mit Gleichungsspezifikationen auf direkte Weise i.a. nur polymorphe Datentypen spezifiziert werden können.
Denn die Klasse $\mathbf{Alg}_{\mathbf{group}}$ enthält sehr verschiedene nichtisomorphe Gruppen.
Auch zur Klasse $\mathbf{Alg}_{\mathbf{stack}}$ gehören Modelle, die man wohl nicht meint: sog. ***Nichtstandardmodelle***.
Man betrachte etwa ein Modell, das als Keller unendliche Folgen von Elementen enthält. Auch dies wäre eine **stack**-Algebra. Sie enthält viel zu viele Elemente (!), nicht alle sind von der Konstanten *clear* aus durch endlichmalige Anwendung der Operationen erreichbar.
Ausschluß solcher zu großer Modelle *nicht* durch Formeln des PK 1. Stufe möglich! Vgl. vorn Axiom iv. und auch das sog. Induktionsaxiom in den PEANO-Axiomen sind Formeln 2. Stufe.

Durch Gleichungsspezifikationen werden also i.a.
zu große Klassen von Algebren beschrieben.

Frage:

Wie kann man diese bequemen, einfachen
Ausdrucksmittel der Gleichungsspezifikation trotzdem
nutzen?

Wie kann man die oft gebrauchten
monomorphen Datentypen
evtl. doch mittels Gleichungen spezifizieren?

Initialität

Eine beliebige gleichungsdefinierbare Klasse $\underline{\mathbf{Alg}}_{\Sigma,E}$ kann niemals - egal, wie E konkret aussieht - ein (interessierender) monomorpher Datentyp sein! Denn eine *entartete* Σ-Algebra mit ***genau einem Element in jeder Trägermenge*** muß natürlich alle Gleichungen erfüllen.

Bei jeder Gleichung der Sorte s kommt bei der Auswertung in einer solchen entarteten Algebra links und rechts trivialerweise stets das Element der Sorte s heraus.

Damit gehören diese entarteten Algebren stets zu $\underline{\mathbf{Alg}}_{\Sigma,E}$, d.h. ein monomorpher ADT könnte nur aus diesen trivialen Algebren bestehen.

Um einen nichttrivialen monomorphen ADT innerhalb von $\underline{\mathbf{Alg}}_{\Sigma,E}$ auszuzeichnen, sind weitere - nicht durch Gleichungen ausdrückbare - Bedingungen nötig!

- **Suche solcher Bedingungen**:

Der bekannteste und zuerst axiomatisch charakterisierte monomorphe ADT ist die Menge **Nz** der natürlichen Zahlen mit 0 und der Nachfolgerbildung.
Die PEANO-Axiome legen **Nz** bis auf Isomorphie fest:

1. $0 \in \mathbf{Nz}$
2. $\forall n(n \in \mathbf{Nz} \rightarrow suc(n) \in \mathbf{Nz})$
3. $\forall n(n \in \mathbf{Nz} \rightarrow suc(n) \neq 0)$
4. $\forall n \forall m(n, m \in \mathbf{Nz} \wedge suc(n) = suc(m) \rightarrow n = m)$
5. *Induktionsaxiom:*
   Jede Teilmenge von **Nz**, die 0 und mit n auch $suc(n)$ enthält, ist gleich **Nz**.

Algebraische Umschreibung der Axiome:

• Axiome 1. und 2.:

Nz ist Trägermenge einer Algebra der Signatur

nat = **sorts** *nat*

oprs 0 : $\mapsto$ *nat*

suc : *nat* $\mapsto$ *nat*

• Axiome 3. und 4. bedeuten, daß

0, *suc*(0), *suc*(*suc*(0)), *suc*(*suc*(*suc*(0))), **...**

alles verschiedene Elemente in **Nz** bezeichnen, d.h. es gelten keine nichttrivialen Gleichungen! ***no confusion***

• Axiom 5. ist eine Minimalitätsforderung:

es gibt keine echte Unteralgebra, die auch die Axiome erfüllt, m.a.W., es gibt keine "überflüssigen" Elemente ***no junk***

Wir sahen früher, daß die Algebra $T_{\mathbf{nat}}$ (Grundtermalgebra der gegebenen Signatur) die PEANO-Axiome erfüllt.

Verallgemeinerung:
Die Forderungen

no confusion: ***es fällt nichts zusammen***

no junk: ***keine Elemente zuviel***

entsprechen bei beliebiger Signatur den verallgemeinerten PEANO-Axiomen, so daß sich die Betrachtung der Eigenschaften der bekannten Grundtermalgebren $\underline{T_\Sigma}$ lohnt. Wir wissen, daß zu jeder Σ-Algebra $\mathcal{A}$ genau ein Σ-Homomorphismus $h_{\mathcal{A}}: \underline{T_\Sigma} \mapsto \mathcal{A}$ von $\underline{T_\Sigma}$ in $\mathcal{A}$ existiert.
Allgemein definiert man:

Definition "initiale Algebra"

Eine Algebra $\mathcal{A}$ heißt ***initial*** in einer Klasse **Alg** von Algebren genau dann, wenn zu jeder Algebra $\mathcal{L} \in$ **Alg** genau ein Homomorphismus h von $\mathcal{A}$ in $\mathcal{L}$ existiert.

Initialität:

T_Σ ist initial in der Klasse $\mathbf{Alg}_\Sigma$ aller Σ-Algebren.

$T_{\Sigma,E}$ ist initial in der Klasse $\mathbf{Alg}_{\Sigma,E}$ aller (Σ,E)-Algebren.

Beweis: Vgl. Fortsetzungsätze für den Spezialfall der leeren Erzeugendensysteme

Isomorphie initialer Algebren:

Sei $\mathcal{A}$ initial in einer Klasse **Alg** von Algebren. Dann ist eine Σ-Algebra $\mathcal{L}$ genau dann initial in **Alg**, wenn $\mathcal{A}$ $\mathcal{A}$ und $\mathcal{L}$ isomorph zueinander sind.

Die Klasse der initialen Algebren in einer Algebrenklasse **Alg** ist eine Isomorphieklasse.

Die Forderung der Initialität eignet sich zur Festlegung monomorpher abstrakter Datentypen.

Definition "abstrakter Datentyp"

Es sei SPEC = (Σ, E) eine Spezifikation. Dann heißt die Isomorphieklasse der initialen Algebren in $\mathbf{Alg}_{SPEC}$ der durch SPEC gegebene ***abstrakte Datentyp*** $\mathbf{ADT}_{SPEC}$.

1. Bei dieser Definition des Begriffs ADT spricht man von der ***initialen Semantik*** abstrakter Datentypen.
2. Zu $\mathbf{ADT}_{SPEC}$ gehört immer die Algebra $T_{\Sigma,E}$, die man durch Faktorisierung der Grundtermalgebra T_Σ der gegebenen Signatur nach der von dem definierenden Gleichungssystem E induzierten Kongruenz $\equiv_E$ erhält.

Beispiel

nat = **sorts** *nat;*
oprs $0 : \mapsto nat$
$suc : nat \mapsto nat$
$+ : nat, nat \mapsto nat$
var $n, m : nat$
eqns $n + 0 = n$
$n + suc(m) = suc(n + m)$

end nat

Der durch **nat** gegebene abstrakte Datentyp $\mathbf{ADT_{nat}}$ ist die Isomorphieklasse von $T_{\mathbf{nat}}$.

Beispiel (Forts.)

Kongruenzklassen von $\underline{T_{\mathbf{nat}}}$:

$$[0] = \{0,\ 0 + 0,\ 0 + (0 + 0),\ \ldots\ \}$$

$$[suc(0)] = \{suc(0),\ suc(0) + 0,\ suc(0 + 0),\ 0 + suc(0),\ suc((0 + 0) + 0),\ \ldots\ \}$$

$$[suc(suc(0))] = \{suc(suc(0)),\ suc(suc(0) + 0),\ suc(0) + suc(0),\ suc(0 + suc(0)),\ suc(suc(0 + 0)),\ \ldots\ \}$$

Offensichtlich ist $\underline{\mathbf{ADT}_{\mathbf{nat}}}$ die Isomorphieklasse der Algebra der natürlichen Zahlen mit 0, Nachfolgerbildung und Addition.

Gleichungskalkül und Induktion

Es sei SPEC = (Σ, E) eine Spezifikation. In allen Algebren aus $\mathbf{Alg}_{\Sigma,E}$ gelten alle aus E ableitbaren Gleichungen.
In Algebren aus $\mathbf{ADT}_{\mathrm{SPEC}}$, die durch die Spezifikation beschrieben werden sollen, können natürlich wesentlich mehr Gleichungen gelten!
Betrachte den eben spezifizierten ADT **nat**.
In $\mathbf{ADT}_{\mathbf{nat}}$ gelten zweifellos die Gleichungen

(K) $$m + n = n + m$$

(A) $$(k + m) + n = k + (m + n).$$

Sie lassen sich aber aus den Gleichungen der Spezifikation

(1) $$n + 0 = n$$

(2) $$n + suc(m) = suc(n + m)$$

nicht ableiten! **Begründen Sie das!**

Die Termalgebra mit den Elementen 0, $suc(0)$, $suc(suc(0))$, ... gehört offensichtlich zu $\mathbf{ADT}_{nat}$.
Ein konkretes Beispiel für die Gleichung (K) in dieser Algebra ist

$$suc(0) + suc(suc(0)) = suc(suc(0)) + suc(0).$$

Diese Gleichung allerdings ist **ableitbar** aus den Spezifikationsgleichungen (1) und (2) :

$$\begin{aligned} suc(0) + suc(suc(0)) &=_{(2)} suc(suc(0) + suc(0)) \\ &=_{(2)} suc(suc(suc(0)) + 0\ ) \\ &=_{(1)} suc(suc(suc(0))) \\ &=_{(1)} suc(suc(suc(0) + 0)) \\ &=_{(2)} suc(suc(0)) + suc(0). \end{aligned}$$

Generell sind alle „konkreten“ Gleichungen, die man durch Einsetzung von Grundtermen aus (K) und (A) erhält, ableitbar!
Solche Gleichungen heißen ***konstante Gleichungen***.

Induktionsbeweisschema:
Gilt für alle Grundterm-Einsetzungen s: $X \mapsto T_\Sigma$

$$s^*(t_1) \equiv_E s^*(t_2),$$

so ist die Gleichung

$$t_1 = t_2$$

in allen initialen (Σ,E)-Algebren gültig.

Dazu überlege man sich, daß sog. minimale Algebren, das sind solche, die nur aus Termelementen bestehen (jedes Element ist „Wert“ eines Grundterms), initial sind. Wegen der Isomorphie initialer Algebren muß deshalb eine Gleichung, die in minimalen Algebren (homomorphe Bilder der Grundtermalgebra!) gilt, in allen initialen gelten.

Beispiel

Kommutativgesetz in initialen **nat**-Algebren, also in $\mathbf{ADT}_{\mathbf{nat}}$:

$$m + n = n + m$$

Die Grundterme der Signatur von **nat** werden aus 0, *suc* und + gebildet.
Die Terme $suc^n(0)$ nennen wir *suc*-Terme:

$$0,\ suc(0),\ suc(suc(0)),\ suc(suc(suc(0))),\ \ldots$$

1. Jeder Grundterm t ist einem *suc*-Term t^* äquivalent:

Entweder t ist selbst ein *suc*-Term oder t enthält einen Teilterm der Gestalt $t' + suc^n(0)$.

Für $n = 0$ ist $t' + 0 \equiv_E t'$.

Für $n \neq 0$ gilt

$$t' + suc^n(0) = t' + suc(suc^{n-1}(0)) \equiv_E suc(t' + suc^{n-1}(0))$$
$$\vdots \quad n\text{-1 Schritte}$$
$$\equiv_E suc^n(t' + 0)$$
$$\equiv_E suc^n(t')$$

Der Teilterm und also auch t selbst ist damit äquivalent einem Term mit einem + weniger. Vollständige Induktion liefert die Behauptung.

Beispiel (Forts.)

2. Das Kommutativgesetz $m + n = n + m$ wird nun durch Anwendung des Induktionsbeweisschemas bewiesen.

Betrachte eine beliebige Grundterm-Einsetzung s mit

$$s(m) = t_1,\ s(n) = t_2\,, \qquad t_1, t_2 \in \underline{T_\Sigma}\,.$$

Zu zeigen: $t_1 + t_2 \equiv_E t_2 + t_1$

Dazu seien $t_1{}^*$, $t_2{}^*$ suc-Terme mit $t_1 \equiv_E t_1{}^*$, $t_2 \equiv_E t_2{}^*$.

$$t_1 + t_2 \equiv_E t_1{}^* + t_2{}^* = suc^i(0) + suc^k(0) = suc^i(0) + suc(suc^{k-1}(0))$$
$$\equiv_E suc(suc^i(0) + suc^{k-1}(0))$$
$$\vdots \quad k\text{-1 Schritte}$$
$$\equiv_E suc^k(suc^i(0) + 0)$$
$$\equiv_E suc^k(suc^i(0))$$
$$= suc^i(suc^k(0))$$
$$\equiv_E suc^i(suc^k(0) + 0)$$
$$\vdots \quad \text{analog wie oben}$$
$$\equiv_E suc^k(0) + suc^i(0)$$
$$= t_2{}^* + t_1{}^* \equiv_E t_2 + t_1$$

Erweiterung von Gleichungsspezifikationen

- Ziel: modularer, schrittweise Spezifikation von ADT

Definition „Kombination“

Es sei SPEC = (Σ, E) eine Spezifikation mit einer S-sortigen Signatur Σ. Hier ist $\Sigma = (\Sigma_{w,s})_{w \in S^*, s \in S}$ (vgl. Kap. 2)
SPEC* = (Σ^*, E^*) heißt eine ***Kombination*** aus SPEC und der ***Ergänzung*** (S_0, Σ_0, E_0), wenn S_0 eine Menge (neuer) Sorten, Σ_0 eine $(S \cup S_0)$-sortige Signatur und E_0 eine Menge von $(\Sigma \cup \Sigma_0)$-Gleichungen ist, und es gilt $\Sigma^* = \Sigma \cup \Sigma_0$, $E^* = E \cup E_0$.

Schreibweise: SPEC* = SPEC + (S_0, Σ_0, E_0)

S_0 , Σ_0 , E_0 können auch leer sein, $\cup$ bezeichnet die disjunkte Vereinigung. (S_0, Σ_0, E_0) selbst ist i.a. keine Spezifikation

Signatur $\Sigma = (S, \Omega, \alpha)$ gegeben.

Bilde für alle $w \in S^*$, $s \in S$ die Mengen

$$\Sigma_{w,s} = \{\omega \mid \omega \in \Omega \wedge \alpha(\omega) = (w, s)\}$$

Man beachte, daß bei Anwendungen fast alle $\Sigma_{w,s}$ leer sind.

Manchmal heißt dann auch die Familie

$$\Sigma = (\Sigma_{w,s})_{w \in S^*, s \in S}$$

eine ***S-sortige Signatur.***

Die Elemente von S sind wieder die ***Sorten***, die von $\Sigma_{w,s}$ ***Operatoren*** mit ***Eingang*** w, ***Ausgang*** s und ***Arität*** (w,s).

$\Rightarrow \alpha(\omega) = (w, s)$ ist gleichbedeutend mit $\omega \in \Sigma_{w,s}$

$\omega \in \Sigma$ steht als Abkürzung für

$$\exists s \exists w (\, s \in S \wedge w \in S^* \wedge \omega \in \Sigma_{w,s} \,).$$

Beispiel 1

bool = **sorts** *bool*

oprs $true, false : \mapsto bool$

$not : bool \mapsto bool$

$and, or, implies : bool, bool \mapsto bool$

var $p, q : bool$

eqns $not(true) = false$

$not(false) = true$

$not(not(p)) = p$

$and(true, p) = p$

$and(false, p) = false$

$and(p, q) = and(q, p)$

$or(p, q) = not(and(not(p), not(q)))$

$implies(p, q) = or(not(p), q)$

end bool

Beispiel 1 (Forts.)

Bilde Kombination durch Ergänzung von **bool** :

ternary = **bool** +

oprs $maybe : \mapsto bool$

eqns $not(maybe) = maybe$

$and(true, maybe) = maybe$

$and(false, maybe) = false$

$or(true, maybe) = true$

end ternary

- Absicht: 3-wertige Logik

Was passiert?

Zu $\mathbf{ADT_{bool}}$ gehört $T_{\mathbf{bool}}$ mit genau 2 Kongruenzklassen als Elementen: [*true*], [*false*].

$T_{\mathbf{ternary}}$ besitzt die 3 Elemente [*true*], [*false*], [*maybe*].

Beispiel 1 (Forts.)

Zu $T_{\underline{\mathbf{ternary}}}$ gehören aber auch noch die Elemente

$[or(maybe, maybe)]$,

$[or(maybe, or(maybe, maybe))]$,

$[or(maybe, or(maybe, or(maybe, maybe)))]$

... usw.

Statt 3 Werten hat man unendlich viele Elemente spezifiziert!

Die Gleichung (oder eine geeignete andere)

$$or(maybe, maybe) = maybe$$

hätte dies verhindert.

- Achtung: Initiale Semantik kann durch ein (kleines) Versehen viel **zu viele Elemente** erzeugen!

In der Regel will man durch Hinzunahme neuer Sorten, Operatoren und Gleichungen die Eigenschaften und Anzahl der ursprünglichen Elemente nicht verändern!

Beispiel 2

Betrachte die früheren Spezifikationen **nat** und **bool** und bilde:

order = **nat** + **bool** +

oprs $\leq$: $nat, nat \mapsto bool$

var $m, n : nat$

eqns $(0 \leq n) = true$

$(suc(n) \leq 0) = false$

$(suc(n) \leq suc(m)) = (n \leq m)$

end order

Bei dieser „Erweiterung“ von **nat** + **bool** entstehen keine neuen Elemente.

Denn die neuen Gleichungen verändern die „Zahlen“ $[suc^n(0)]$ nicht. Es entstehen auch keine neuen Elemente der Sorte *bool*. Denn jeder Term der Sorte *bool* der Gestalt $(t_1 \leq t_2)$ läßt sich durch die Gleichungen aus **nat** in $(t_1^* \leq t_2^*)$ umformen mit *suc*-Termen t_1^*, t_2^*, und dann mit den neuen Gleichnugen in *true* oder *false*.

Beispiel 3

Es ist auch möglich, daß durch eine solche "Kombination" ursprüngliche Elemente weg- bzw. zusammenfallen.

Betrachte z.B. die Kombination

testbool = **bool** +

eqns $implies(true, false) = true$

end testbool

Hier erhält man

$$[true] = [false],$$

denn es gilt dann

$$true \equiv_{E^*} implies(true, false) \equiv_{E^*} or(not(true), false)$$
$$\equiv_{E^*} or(false, false) \equiv_{E^*} not(and(not(false), not(false)))$$
$$\equiv_{E^*} not(and(true, true)) \equiv_{E^*} not(true) \equiv_{E^*} false\ .$$

Damit ist diese Kombination **keine** „Erweiterung“ von **bool** .

- Ziel: Untersuchung, wann eine Kombination tatsächlich eine **echte Erweiterung** der ursprünglichen Spezifikation darstellt.

Definition „Redukt“

Es sei SPEC = (Σ, E) eine Spezifikation und
SPEC* = SPEC + (S_0, Σ_0, E_0).
Das SPEC-***Redukt*** von T_{SPEC*} ist die Σ-Algebra

$$(T_{SPEC*})_{SPEC} = [(T_{SPEC*})_{SPEC}, (f_\omega{}^*)_{\omega \in \Sigma}]$$

mit

$$(T_{SPEC*})_{SPEC} = (T_{SPEC*, s})_{s \in S}$$

(Nur die Sorten und Operatoren, die auch in SPEC vorkommen!)

Mit diesem Begriff läßt sich nun eine **echte Erweiterung** vernünftig definieren.

Definition „Erweiterung“

Eine Spezifikation SPEC* heißt eine ***Erweiterung*** der Spezifikation SPEC genau dann, wenn

$$(T_{SPEC*})_{SPEC} \cong T_{SPEC} .$$

Bei den vorangegangenen Beispielen trifft diese Bedingung offensichtlich nur auf das Beispiel 2 zu.

Die **Bedingung**

$$(\underline{T_{SPEC*}})_{\underline{SPEC}} \cong \underline{T_{SPEC}}$$

für eine Kombination, eine **Erweiterung** zu sein, ist eine ***semantische Bedingung***.
Denn es ist im wesentlichen Isomorphie für gewisse Faktor-algebren zu überprüfen. Aber sowohl Homomorphismen als auch Kongruenzen entsprechen Semantik!

Deshalb ist folgender Satz nicht verwunderlich:

Die Frage, ob eine Spezifikation SPEC* eine Erweiterung einer Spezifikation SPEC ist, ist im allgemeinen unentscheidbar.

$\Rightarrow$ Suche nach **hinreichenden Kriterien**

Es sei $\underline{SPEC} = (\Sigma, E)$ eine Spezifikation und
$\underline{SPEC^*} = \underline{SPEC} + (S_0, \Sigma_0, E_0)$.
h sei der eindeutig bestimmte Homomorphismus [1)]

$$h : \underline{T}_{\underline{SPEC}} \mapsto (\underline{T}_{\underline{SPEC^*}})_{\underline{SPEC}} .$$

Dann heißt $\underline{SPEC^*}$

vollständig bezüglich $\underline{SPEC}$ genau dann,
wenn h surjektiv ist, und

konsistent bezüglich $\underline{SPEC}$ genau dann,
wenn h injektiv ist.

[1)] Beachte, daß beides (Σ, E)-Algebren sind, und $\underline{T}_{\underline{\Sigma,E}}$ in der Klasse $\underline{\mathbf{Alg}}_{\underline{\Sigma,E}}$ initial ist.

Beispiel

Man überzeugt sich leicht davon, daß bei den vorangegangenen drei Beispielen gilt:

- **ternary** ist bezüglich **bool** konsistent, aber nicht vollständig.
- **order** ist bezüglich **nat + bool** konsistent und vollständig.
- **testbool** ist bezüglich **bool** vollständig, aber nicht konsistent.

Trivialerweise gilt der Satz:

Konsistenz und Vollständigkeit:

SPEC* ist eine Erweiterung von SPEC genau dann, wenn SPEC* bezüglich SPEC konsistent und vollständig ist.

Beweis: Ein Homomorphismus *h* ist Isomorphismus genau dann, wenn *h* surjektiv und injektiv ist.

Beispiel

Beim Entwurf von Spezifikationen können sehr leicht *inkonsistente Kombinationen* entstehen:

set(nat) = **nat** +

sorts *set*

oprs $\varnothing : \mapsto set$

$insert : set, nat \mapsto set$

var $s : set$

$m, n : nat$

(1) **eqns** $insert(insert(s, n), n) = insert(s, n)$

(2) $insert(insert(s, n), m) = insert(insert(s, m), n)$

end **set(nat)**

Nun soll noch eine Lösch-Operation hinzukommen.
Wie könnte man da diese Spezifikation noch „erweitern“?

Beispiel (Forts.)

Man muß einen zusätzlichen Operator einführen und dessen Eigenschaften durch geeignete Gleichungen beschreiben:

newset(nat) = **set(nat)** +

oprs $del : set, nat \mapsto set$

var $s : set$

$n : nat$

(3) **eqns** $del(\emptyset, n) = \emptyset$

(4) $del(insert(s, n), n) = s$

end newset(nat)

Sieht vernünftig aus - oder?

Beispiel (Forts.)

Leider funktioniert dies nicht in der offenbar gewünschten Weise.

Denn es ergibt sich „dummerweise“:

$$\begin{aligned} \varnothing &\equiv_{E^*} del(insert(\varnothing, n), n) && \text{wegen (4)} \\ &\equiv_{E^*} del(insert(insert(\varnothing, n), n), n) && \text{wegen (1)} \\ &\equiv_{E^*} insert(\varnothing, n) && \text{wegen (4)} \end{aligned}$$

Die Gleichung (1) läßt sich wie alle anderen ja auch von rechts nach links verwenden.

Damit hat die Operation *insert* keine Wirkung mehr!

Die Sorte *set* besteht nur noch aus einem einzigen Element: [∅] !

Die Spezifikation **newset(nat)** ist bezüglich **set(nat)** inkonsistent!

Lemma:
Falls SPEC* = SPEC + (S_0, ∅, ∅), so ist SPEC* eine Erweiterung von SPEC .

Beweis: trivial

Lemma:
Wenn SPEC* eine Erweiterung von SPEC ist, und SPEC** eine Erweiterung von SPEC*, so ist SPEC** auch eine Erweiterung von SPEC .

Beweis: gilt wegen der Eigenschaften bei Nacheinanderausführung von Homomorphismen

Gilt $\underline{SPEC^*} = \underline{SPEC} + (\varnothing, \Sigma_0, E_0)$, so heißt $\underline{SPEC^*}$ eine ***Anreicherung*** von $\underline{SPEC}$.

Wegen der obigen Lemmata genügt es, die

Suche nach **hinreichenden Kriterien**

dafür, daß eine Kombination eine Erweiterung ist, auf Anreicherungen zu beschränken.
Das heißt, wir suchen nun Kriterien für die Vollständigkeit und Konsistenz von **Anreicherungen**.

Vollständigkeit und Konsistenz von Anreicherungen:

Es sei SPEC* eine Anreicherung einer Spezifikation SPEC, SPEC* = SPEC + $(\emptyset, \Sigma_0, E_0)$,

wobei SPEC* = (Σ^*, E^*), SPEC = (Σ, E),

$\Sigma^* = \Sigma \cup \Sigma_0$, $E^* = E \cup E_0$ (alles S-sortig).

Dann ist SPEC*

1.) vollständig bezüglich SPEC genau dann, wenn

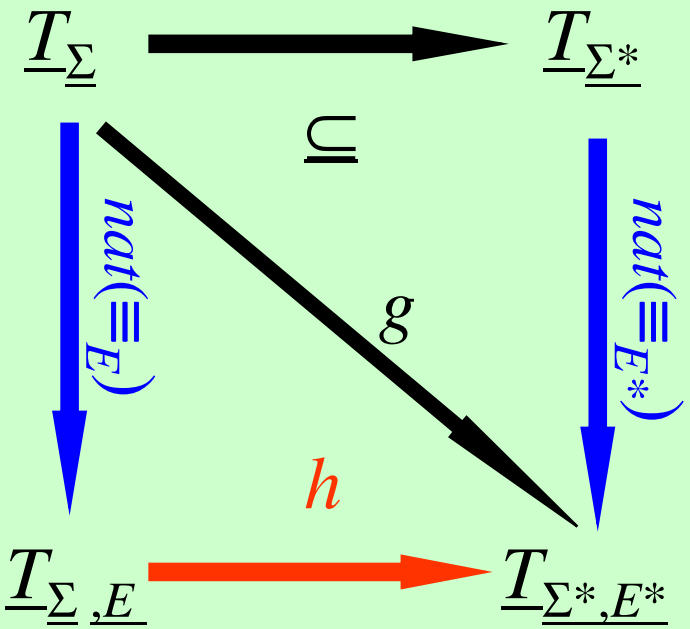
$$\forall t' \, (t' \in T_{\Sigma^*} \rightarrow \exists t \, (t \in T_\Sigma \wedge t' \equiv_{E^*} t \,),$$

und

2.) konsistent bezüglich SPEC genau dann, wenn

$$\forall t_1 \, \forall t_2 \, (\, t_1 \,, t_2 \in T_\Sigma \wedge t_1 \equiv_{E^*} t_2 \rightarrow t_1 \equiv_E t_2).$$

Beweis:



Die Homomorphismen g, h und die Inklusion werden jeweils auf den SPEC-Redukten betrachtet. Wegen Initialität existieren alle diese Homomorphismen und sind eindeutig bestimmt.

Weil speziell auch g eindeutig ist, muß auch gelten:

$(*)$ $\quad h([t]_{\equiv_E}) = [t]_{\equiv_{E^*}}$ für alle $t \in T_\Sigma$

zu 1.) SPEC* vollständig bezüglich SPEC gdw.

$$\forall t' \, (t' \in T_{\Sigma^*} \rightarrow \exists t \, (t \in T_\Sigma \wedge t' \equiv_{E^*} t))$$

Beweis (Forts.):

(∗) $h([t]_{\equiv_E}) = [t]_{\equiv_{E^*}}$ für alle $t \in T_\Sigma$

zu 1.) SPEC* vollständig gdw. h surjektiv ist. Dies gilt gdw. zu jeder Kongruenzklasse $[t']_{\equiv_{E^*}}$ eine Kongruenzklasse $[t]_{\equiv_E}$ existiert mit $h([t]_{\equiv_E}) = [t']_{\equiv_{E^*}}$.

Wegen (∗) haben wir

$$[t']_{\equiv_{E^*}} = h([t']_{\equiv_E}) = h([t]_{\equiv_E}) = [t]_{\equiv_{E^*}} \quad \text{gdw.} \quad t' \equiv_{E^*} t \ .$$

Also ist SPEC* vollständig gdw. zu jedem $t' \in T_{\Sigma^*}$ ein $t \in T_\Sigma$ existiert mit $t' \equiv_{E^*} t$.

zu 2.) SPEC* konsistent bezüglich SPEC gdw.

$\forall t_1 \forall t_2 (t_1, t_2 \in T_\Sigma \wedge t_1 \equiv_{E^*} t_2 \rightarrow t_1 \equiv_E t_2)$

Beweis (Forts.):

(∗) $h([t]_{\equiv_E}) = [t]_{\equiv_{E^*}}$ für alle $t \in T_\Sigma$

zu 2.) SPEC* konsistent gdw. h injektiv ist. Dies gilt gdw.

$\forall t_1, t_2 \in T_\Sigma$:

$$h([t_1]_{\equiv_E}) = h([t_2]_{\equiv_E}) \rightarrow [t_1]_{\equiv_E} = [t_2]_{\equiv_E}$$

Wegen (∗) gilt dies gdw.

$$[t_1]_{\equiv_{E^*}} = [t_2]_{\equiv_{E^*}} \rightarrow [t_1]_{\equiv_E} = [t_2]_{\equiv_E}$$

also gdw.

$$t_1 \equiv_{E^*} t_2 \rightarrow t_1 \equiv_E t_2 ,$$

w.z.b.w.

Das Kriterium für die Vollständigkeit läßt sich noch verschärfen: Man muß nur solche Terme beachten, bei denen die **neuen Operatoren nur als Hauptverknüpfungsfunktoren** auftreten.

Definition „Σ_0-normaler Term“

Sei $\underline{SPEC^*} = \underline{SPEC} + (\varnothing, \Sigma_0, E_0)$, $\underline{SPEC^*} = (\Sigma^*, E^*)$, $\underline{SPEC} = (\Sigma, E)$.

Ein Term $t \in T_{\Sigma^*}$ heißt ein Σ_0-***Term*** genau dann, wenn

$$t = \omega t_1 t_2 \ldots t_n \text{ für ein } \omega \in \Sigma_0.$$

Ein Σ_0-Term $\omega t_1 t_2 \ldots t_n$ heißt ein Σ_0-***normaler Term*** genau dann, wenn $\forall i\ (1 \leq i \leq n)$: $t_i \in T_\Sigma$ ist.

Vollständigkeitskriterium:

Mit denselben Bezeichnungen wie eben gilt:

SPEC* ist vollständig bezüglich SPEC genau dann, wenn für alle Σ_0-normalen Terme $t_0 \in T_{\Sigma^*}$ ein $t \in T_\Sigma$ existiert mit $t_0 \equiv_{E^*} t$.

Der **Beweis** läßt sich leicht induktiv über die Anzahl der Vorkommen von neuen Operatoren aus Σ_0 in einem Term $t' \in T_{\Sigma^*}$ zeigen. Man ersetze im Induktionsschritt den jeweils innersten Σ_0-normalen Term durch den gemäß obigem Kriterium existierenden äquivalenten Term aus T_Σ . Der resultierende Term ist nach dem Gleichungskalkül zum ursprünglichen t' äquivalent und enthält mindestens einen neuen Operator weniger.

Die vorgestellten Kriterien erfordern die Prüfung der Äquivalenz von Termen:

$$t_1 \equiv_E t_2 \quad ?$$

für verschiedene Gleichungssysteme, d.h. die Lösung des entsprechenden sogenannten **Wortproblems**.
Folglich ist diese Frage i.a. ***unentscheidbar***!
Man benötigt i.a. weitere **hinreichende Kriterien**. Für eine Reihe konkreter Gleichungssysteme ist die Frage auch entscheidbar! Damit beschäftigt sich die Theorie der **Termersetzungssysteme**.

- Für die **Vollständigkeit** gibt es eine Reihe von praktikablen, d.h. relativ einfachen, hinreichenden Kriterien.
- Für die Entscheidung der **Konsistenz** gibt es vergleichbar einfache Kriterien leider bisher nicht.

Finale Semantik und beobachtbares Verhalten

Das Ziel der Benutzung von Gleichungsspezifikationen SPEC = (Σ, *E*) zur Festlegung eines **monomorphen** abstrakten Datentyps $\mathbf{ADT}_{SPEC}$ hat uns zur **initialen Semantik** geführt.
Aber die Isomorphieklasse der initialen Algebren in $\mathbf{Alg}_{SPEC}$ ist natürlich *nicht die einzige Isomorphieklasse* in $\mathbf{Alg}_{SPEC}$.
Sie ist auch nicht a priori vor anderen Isomorphieklassen ausgezeichnet. Die initiale Semantik besitzt auch **Nachteile**.
Es ist z.B. sehr leicht möglich, inkonsistente „Erweiterungen“ zu entwerfen.
Deshalb ist es zweckmäßig, auch **andere Semantiken** zu suchen und zu untersuchen. Zur Motivation der Richtung der Suche betrachten wir wieder ein **Beispiel**.

Beispiel

Es seien wieder **nat** und **bool** wie bei den früheren Beispielen gegeben.

nateq = **nat** + **bool** +

oprs $eq : nat, nat \mapsto bool$

var $n, m : nat$

eqns $(0\ eq\ 0) = true$

$(0\ eq\ suc(n)) = false$

$(suc(n)\ eq\ 0) = false$

$(suc(n)\ eq\ suc(m)) = (n\ eq\ m)$

end nateq

Es ist relativ leicht, nachzuweisen, daß die Spezifikation **nateq** eine Erweiterung von **nat** ist. **Überzeugen Sie sich selbst davon!**

Beispiel (Forts.)

andere Spezifikation von
„Mengen natürlicher Zahlen“
als in früherem Beispiel

setofnat = **nateq** +

sorts *set*

oprs $\emptyset : \mapsto set$

$insert : set, nat \mapsto set$

$del : set, nat \mapsto set$

$if\text{-}then\text{-}else\text{-}set : bool, set, set \mapsto set$

$if\text{-}then\text{-}else\text{-}nat : bool, nat, nat \mapsto nat$

$if\text{-}then\text{-}else\text{-}bool : bool, bool, bool \mapsto bool$

$\in : nat, set \mapsto bool$

$equal : set, set \mapsto bool$

$\subseteq : set, set \mapsto bool$

$card : set \mapsto nat$

var …

eqns …

siehe nächste Folie

end **setofnat**

Beispiel (Forts.)

setofnat = **nateq** +

sorts …

oprs …

siehe vorige Folie

var $s, t : set \; ; \; m, n : nat \; ; \; p, q : bool$

eqns $if\ true\ then\ s\ else\ t\ set = s$

$if\ false\ then\ s\ else\ t\ set = t$

$if\ true\ then\ m\ else\ n\ nat = m$

$if\ false\ then\ m\ else\ n\ nat = n$

$if\ true\ then\ p\ else\ q\ bool = p$

$if\ false\ then\ p\ else\ q\ bool = q$

$n \in \emptyset = false$

$n \in insert(s, m) = if\ (n\ eq\ m)\ then\ true\ else\ n \in s\ bool$

…

weitere Gleichungen nächste Folie

end setofnat

Beispiel (Forts.)

setofnat = **nateq** +

sorts …

oprs …

siehe Folie vorn

var $s, t : nat \;;\; m, n : nat \;;\; p, q : bool$

eqns …

Gleichungen aus voriger Folie und ...

$card(\emptyset) = 0$

$card(insert(s, n)) = if\ n \in s\ then\ card(s)\ else\ suc(card(s))\ nat$

$del(\emptyset, n) = \emptyset$

$del(insert(s, n), m) = if\ (n\ eq\ m)\ then\ del(s, m)$
$else\ insert(del(s, m), n)\ set$

$(\emptyset \subseteq \emptyset) = true$

$(\emptyset \subseteq insert(s, n)) = true$

$(insert(s, n) \subseteq t\ ) = and(n \in t\ ,\ s \subseteq t)$

$(s\ equal\ t\ ) = and(s \subseteq t, t \subseteq s)$

end **setofnat**

Beispiel (Forts.)

Wie man sich leicht überzeugt, ist nun in **setofnat** nicht mehr

$$\emptyset \equiv_{E^*} insert(\emptyset, 0)$$

abzuleiten, allerdings gilt nun auch

$$insert(insert(s, 0), 0) \not\equiv_{E^*} insert(s, 0),$$

d.h. die Kongruenzklassen

$[insert(s, 0)]$ und $[insert(insert(s, 0), 0)]$

in $T_{\textbf{setofnat}}$ sind verschieden!

Es werden also verschiedene Darstellungen „derselben Menge“ unterschieden.

Das liegt hier auch an der initialen Semantik.

Das initiale Modell erweist sich hier wieder einmal als **zu groß**!

Wenn man sich aber die in **setofnat** erfaßbaren Eigenschaften der Mengen genauer ansieht, stellt man fest, daß hierbei verschiedene Darstellungen derselben Menge dieselben Eigenschaften haben!

Beispiel (Forts.)

In **setofnat** gilt z.B.

$card(insert(insert(\emptyset, n), n)) \equiv_{E^*} if\ \ n \in insert(\emptyset, n)$

$then\ card(insert(\emptyset, n))\ else\ suc(card(insert(\emptyset, n)))\ nat$

$\equiv_{E^*} card(insert(\emptyset, n))$

und analoge Gleichungen für $\in$, *equal*, $\subseteq$.

Ein oft genutzter Ansatz für Datentypen ist es, zwischen ***„sichtbaren“*** und ***„versteckten“*** Sorten und Operatoren zu unterscheiden. Wenn man also über die „neuen“ Elemente ***nur durch Anwendung der „erlaubten“ Operationen*** etwas erfährt, dann lassen sich die obigen verschiedenen Darstellungen nicht mehr unterscheiden! Das ***beobachtbare Verhalten*** ist gleich!

setofnat …

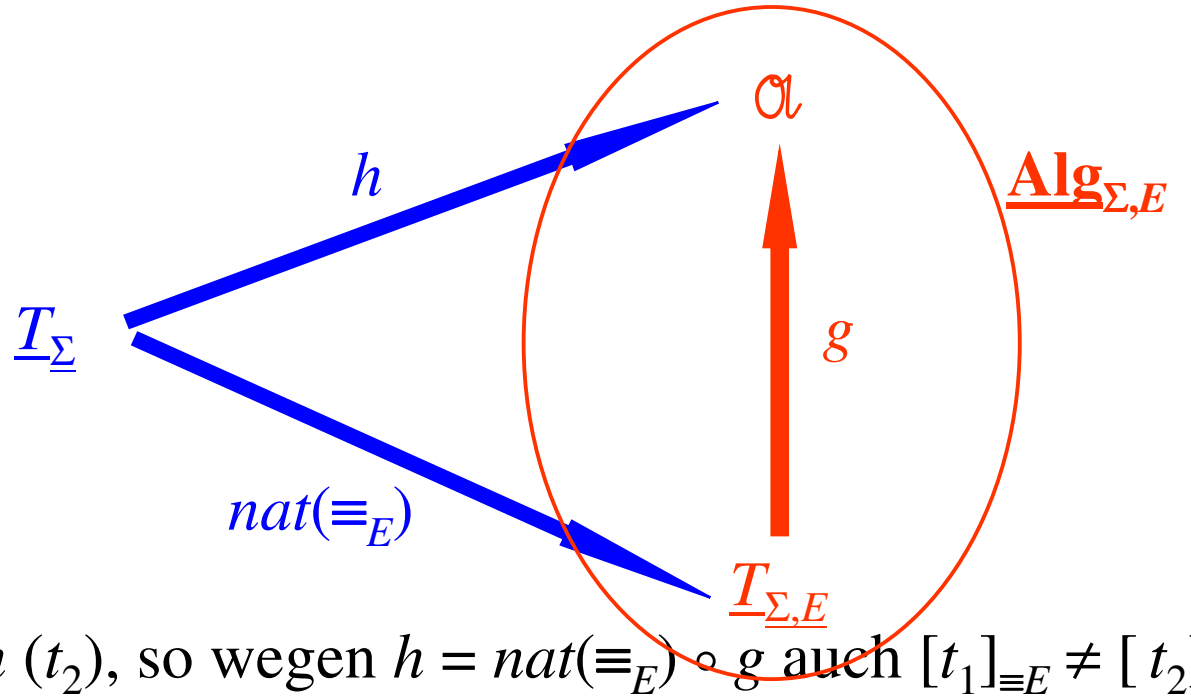
hidden sorts *set*

oprs $\emptyset$, *insert*, *del*

…

Initiale Modelle sind zur Erfassung des beobachtbaren Verhaltens i.a. zu groß! Generell kann man die initiale Algebra $\underline{T_{\Sigma,E}}$ in $\underline{\mathbf{Alg}_{\Sigma,E}}$ als „möglichst große“ Algebra mit leeren Erzeugendensystem beschreiben.

Für beliebige Terme $t_1, t_2 \in T_\Sigma$ gilt $[t_1]_{\equiv E} \neq [t_2]_{\equiv E}$, sobald sie nur in einer einzigen Algebra $\mathcal{A} \in \underline{\mathbf{Alg}_{\Sigma,E}}$ verschiedene Werte haben!



Wenn $h(t_1) \neq h(t_2)$, so wegen $h = nat(\equiv_E) \circ g$ auch $[t_1]_{\equiv E} \neq [t_2]_{\equiv E}$.

Initiale Modelle zu groß!

- Gibt es evtl. auch ***„möglichst kleine“*** Algebren mit leerem Erzeugendensystem in $\underline{\mathbf{Alg}_{\Sigma,E}}$?

  Dual zum initialen Fall sollten zwei Terme $t_1, t_2 \in T_\Sigma$ in dieser neuen „semantischen Algebra“ nur dann verschiedene Werte haben, wenn sie in ***allen*** Modellen, d.h. in allen (Σ, E)-Algebren, verschiedene Werte haben.

Beachte aber: In $\underline{\mathbf{Alg}_{\Sigma,E}}$ liegt immer die entartete Algebra mit genau einem Element in jeder Trägermenge. Diese erfüllt obige Bedingung, ist aber **viel zu klein**!

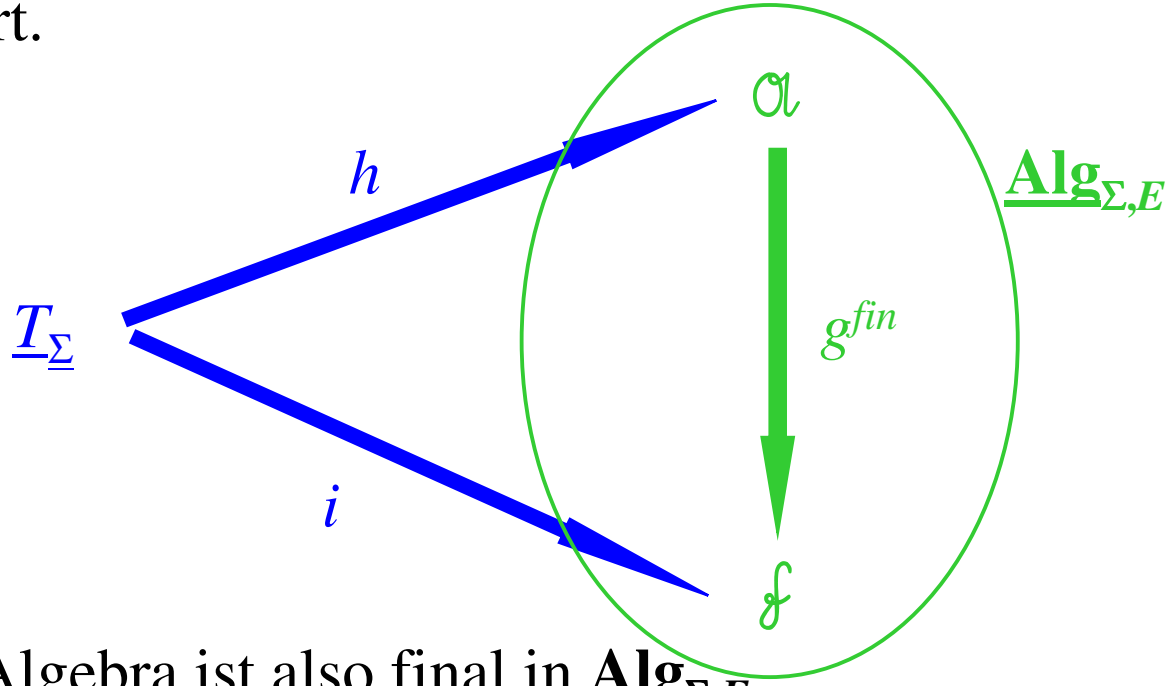
- Initialität wurde mittels Homomorphismen definiert. Wir definieren einen dualen Begriff ***„finale Algebra“***.

Sei $\underline{\mathbf{K}} \subseteq \underline{\mathbf{Alg}_\Sigma}$ eine Klasse von Σ-Algebren.
Dann heißt eine Algebra $\mathcal{A} \in \underline{\mathbf{K}}$ ***final*** in $\underline{\mathbf{K}}$ genau dann, wenn zu jeder Algebra $\mathcal{B} \in \underline{\mathbf{K}}$ genau ein Homomorphismus h von $\mathcal{B}$ in $\mathcal{A}$ existiert.

Analog zum Falle der Initialität kann man leicht zeigen, daß die in einer Klasse $\underline{\mathbf{K}}$ finalen Algebren eine Isomorphieklasse bilden.

Aber natürlich gibt es nicht in jeder Klasse $\underline{\mathbf{K}}$ finale Algebren.

Es sei $\mathfrak{A}$ eine (Σ, E)-Algebra. Angenommen, in $\underline{\mathbf{Alg}_{\Sigma,E}}$ gibt es eine finale Algebra $\mathfrak{f}$, so ergibt sich:
Für beliebige Terme $t_1, t_2 \in T_\Sigma$ folgt aus $i(t_1) = i(t_2)$ sofort $g^{fin}(h(t_1)) = g^{fin}(h(t_2))$.
Umgekehrt, wenn t_1, t_2 in der Algebra $\mathfrak{A}$ denselben Wert haben, $h(t_1) = h(t_2)$, ist auch $i(t_1) = i(t_2)$, d.h. sie haben auch in $\mathfrak{f}$ denselben Wert.



Die entartete Algebra ist also final in $\underline{\mathbf{Alg}_{\Sigma,E}}$.

Eine sinnvolle ***finale Semantik*** erfordert die Verkleinerung der Algebrenklasse, um die entartete Algebra auszuschließen. Üblich sind zwei **Methoden**:

- Definition einer Unterklasse $\mathbf{MOD} \subset \mathbf{Alg}_{\Sigma,E}$ zulässiger Modelle von SPEC = (Σ, E). Wenn es in **MOD** eine finale Algebra $\mathcal{A}$ gibt, so ist deren Isomorphieklasse der durch die finale Semantik bestimmte abstrakte Datentyp $\mathbf{FIN}_{SPEC}$.

- Definition einer Kongruenzrelation $\sim_E$ mittels des Gleichungssystems E und bilde $T_\Sigma / \sim_E$. Zeige, daß $T_\Sigma / \sim_E$ final in einer bestimmten Modellklasse $\mathbf{MOD} \subset \mathbf{Alg}_{\Sigma,E}$ ist.
  Die Isomorphieklasse von $T_\Sigma / \sim_E$ ist dann wieder der abstrakte Datentyp $\mathbf{FIN}_{SPEC}$.

- ***Initiale Semantik*** ist ein fester Begriff.

- Bei der ***finalen Semantik*** gibt es Wahlmöglichkeiten in der direkten Bestimmung der zulässigen Modellklasse **MOD** bzw. durch die Festlegung einer geeigneten Kongruenz $\sim_E$.

- Es gibt ***sehr unterschiedliche*** Ansätze für die ***finale Semantik***.

- Durch geeignete Definition der Kongruenzrelation $\sim_E$ gelingt es z.B., das sog. ***„beobachtbare Verhalten“*** zu modellieren.
  Auch für die Formalisierung des beobachtbaren Verhaltens gibt es ***verschiedene*** Möglichkeiten.

Kapitel 14

Implementation von Spezifikationen

Es sei SPEC = (Σ, *E*) eine Spezifikation. Für die *Implementation* einer solchen Spezifikation sind zweifellos wieder gewisse *Datenmengen* und darüber bestimmte *Operationen* festzulegen. Das heißt, eine Implementation ist auch eine ***Algebra***.

> *Wann ist eine Algebra $\mathcal{A}$ eine Implementation der Spezifikation* SPEC ?

Der einfache Gedanke, daß $\mathcal{A}$ zu dem spezifizierten abstrakten Datentyp $\mathbf{ADT}_{\mathrm{SPEC}}$ gehört, daß also $\mathcal{A} \cong T_{\Sigma,E}$ ist, reicht auch bei der initialen Semantik offensichtlich nicht aus.
Denn $\mathcal{A}$ ist Σ-Algebra, in der Praxis wird man für eine Implementation andere, evtl. auch mehr, Sorten und Operationen verwenden. Man muß also eine ***andere Signatur*** Σ' zulassen!

Fragestellungen:

- Was bedeutet, daß eine Σ'-Algebra $\mathfrak{A}$ eine ***Implementation*** einer Spezifikation SPEC = (Σ, E) ist?
- Wie soll eine solche Implementation $\mathfrak{A}$ angegeben werden:
  - ***konkrete*** Definition der Trägermengen (Datenbereiche) und Operationen?
    oder
  - ***abstrakt*** durch eine Spezifikation?

Beides ist möglich. Mit dem Ziel, eine strukturierte Entwicklung auch von Implementationen im Sinne der schrittweisen Verfeinerung möglich zu machen, wird hier die zweite Variante bevorzugt.

Wann implementiert eine Spezifikation SPEC′ = (Σ', E') *eine Spezifikation* SPEC = (Σ, E) *korrekt?*

Beispiel 1

bool sei die uns schon bekannte Spezifikation. Bekanntlich können alle Booleschen Funktionen allein durch die sogenannte *nand*-Funktion (= *not*(*and*(-,-)) dargestellt werden. Also muß man **bool** durch den folgenden ADT implementieren können:

bit = **sorts** *bit*
oprs $0, 1 : \mapsto bit$
$nand : bit, bit \mapsto bit$
var $b : bit$
eqns $nand(0, b) = 1$
$nand(b, 0) = 1$
$nand(1, 1) = 0$

end bit

Beispiel 1 (Forts.)

Zur ***Implementation*** von **bool** durch **bit** muß man die

Sorten umbenennen: $bool \rightsquigarrow bit$,

Datenelemente/**Konstanten umbenennen**: $false \rightsquigarrow 0$

$true \rightsquigarrow 1$,

die **Operationen** *not*, *and*, *or*, *implies* von **bool** durch die Operation von **bit** **erklären**:

$$\begin{aligned} not(p) &:= nand(p, p) \\ and(p, q) &:= nand(nand(p, q), nand(p, q)) \\ or(p, q) &:= nand(nand(p, p), nand(q, q)) \\ implies(p, q) &:= nand(p, nand(q, q)). \end{aligned}$$

Das heißt, den Operationen von **bool** entsprechen ***abgeleitete Operationen*** von **bit**.

Da die **Operation** *nand* in **bool** zuviel ist, muß sie anschließend **„vergessen“** werden.

Beispiel 2

nat sei die bekannte Spezifikation. Oft gibt es in den Programmiersprachen keinen eigenen Datentyp **nat**. Die natürlichen Zahlen werden durch **integer**-Zahlen dargestellt. Also muß man **nat** durch **int** implementieren können:

int = **sorts** int

oprs $0 : \mapsto int$

$suc, pred : int \mapsto int$

$+ : int, int \mapsto int$

var $k, l : int$

eqns $pred(suc(k)) = k$

$suc(pred(k)) = k$

$k + 0 = k$

$k + suc(l) = suc(k + l)$

$k + pred(l) = pred(k + l)$

end int

Beispiel 2 (Forts.)

Innerhalb von $\underline{T_{\mathbf{int}}}$ läßt sich $\underline{T_{\mathbf{nat}}}$ ***implementieren*** durch **„Vergessen“ der Operation** *pred* und **Einschränkung der Trägermenge**, d.h. durch **„Vergessen“ von Datenelementen**, hier der negativen Zahlen $pred^n(0)$.
Das bedeutet, daß man nur noch den Teil von $\underline{T_{\mathbf{int}}}$ betrachtet, der bei den noch erlaubten Operationen von den Konstanten (hier nur die „0“) aus ***erreichbar*** ist.
Das heißt, man bildet die entsprechende **Unteralgebra**.

Beispiel 3

Man stelle sich vor, den ADT **int** zu implementieren durch einen ADT **vorznat**, in dem mit Paaren von Vorzeichen und natürlichen Zahlen gerechnet wird. Neben der Sorte *int* hat man dann z.B. die Sorten *nat* und *bool* (für das Vorzeichen) und eine Paarbildungsfunktion

$$i : bool, nat \mapsto int.$$

Zur ***Implementation*** hat man dann wieder die **Operationen** aus **int** durch ***abgeleitete Operationen*** aus **vorznat** zu **erklären**, anschließend die **Operationen** als auch die **Sorten** *bool* und *nat* aus **vorznat** zu **vergessen**.

Zusätzlich wird man hier noch Datenelemente **identifizieren** müssen, denn es gibt ja zwei Darstellungen der „Null“:

$$i(true, 0) \quad \text{und} \quad i(false, 0).$$

Man beachte, daß es hier keine Konstanten der Sorte *int* gibt, sondern alle *int*-Datenelemente Resultate von Operationen sind.

Schritte zur Implementation

Synthese
Zu implementierende Operationen und Sorten müssen mit Hilfe der Sorten, Operationen und abgeleiteten Operationen der Implementation erklärt werden.

Identifikation
Mehrfachdarstellungen derselben zu implementierenden Datenelemente müssen mittels einer geeigneten Kongruenzrelation identifiziert werden.

Restriktion
Überflüssige Sorten und Operationen als auch zu große Trägermengen müssen durch Vergessen und Einschränkung, d.h. durch Teil- und Unteralgebrenbildung entfernt werden.

Syntax, Semantik, Spezifikation

Grundlagen der Informatik

R. Hartwig

ENDE