

Kapitel 3

Termalgebren

3.1 Formeln

3.2 Boolesche Algebra

3.3 Algebraische Strukturen
und Algebren

3.4 Abbildungen zwischen Algebren

3.5 Termalgebren

3.6 Termalgebren mit Variablen

3.7 Termersetzungssysteme

3.8 Vertiefung Algorithmusbegriff



- Alle Datentypen und Datenstrukturen sind Termalgebren
- Unterscheide Spezifikation (Schnittstellenbeschreibung) und Implementierung:
 - Spezifikation nennt die an der Schnittstelle vorhandenen und gültigen
 - Operationen (Signatur)
 - Gesetze (Axiome)
 - beides zusammen kennzeichnet eine abstrakte Algebra
 - Implementierung beschreibt eine mögliche Realisierung
 - konkrete Algebra, die auch die Gesetze erfüllt
 - konkrete Algebra ist homomorphes Bild der abstrakten Algebra
 - sonst wäre die Implementierung fehlerhaft
- Termersetzung ist Berechnungsmethode auf der Spezifikationsebene
 - beschreibt auch die Berechnung von Funktionsaufrufen
 - grundlegendes Rechenmodell für funktionale und dann auch für imperative Programmiersprachen



- Was ist „3+5“ ?
Antwort: „drei plus fünf“
⇒ Wir unterscheiden zwischen der **algebraischen Formel** „3+5“ und dem **Ergebnis** dieser Formel.
 - die Formel heißt eine **Rechenvorschrift** (Funktion, Prozedur,...)
 - das Ergebnis erhält man durch **Auswertung (Berechnung)** der Rechenvorschrift
- Formeln heißen auch **Terme**.
- In einer **Algebra A** bildet auch die Menge der Terme eine Algebra (**Termalgebra**).



Satz: \mathcal{B} ist boolesche Algebra, wenn gilt:

1. Von V1-V5 gilt das erste (oder jeweils das zweite) Gesetz.
2. Entweder die Gesetze V8 oder die Gesetze V10 gelten.
3. \mathcal{B} ist vollständig, d.h. $\inf(X), \sup(X) \in U$ für alle $X \subseteq U$.

Also: zum Nachweis, daß eine boolesche Algebra vorliegt, braucht man nicht alle Gesetze beweisen. Insbesondere folgen V6, V7 und V9 aus den anderen Gesetzen (siehe auch die früheren Beweise).



Ausgangspunkt: Signatur Σ , Elementaroperanden X , Menge \mathcal{T} der korrekten Terme zu Σ und X .

Abstrakte Algebra (algebraische Struktur):

- Tripel $\mathcal{A} = (\mathcal{T}, \Sigma, Q)$ wobei
 - Q Menge von Gesetzen (Axiome), denen die $f \in \Sigma$ genügen
 - Für zwei Terme $t, t' \in \mathcal{T}$ gilt:
 $t = t'$ gdw. t nach den Gesetzen Q in t' umgeformt werden kann.

Beispiele:

Halbgruppen, Monoide, Verbände, Gruppen, Ringe,
Körper, Vektorräume, boolesche Algebren,
alle Datenstrukturen der Informatik (**abstrakte Datentypen**)

Beispiele für Gesetze:

die Halbgruppen- und Monoidgesetze HG1, HG2 (nur bei kommutativen HG),
die Gesetze V1-V10 der booleschen Algebra



Gegeben: Signatur Σ , Menge $X \supseteq \Sigma^{(0)}$ von Elementaroperanden, abstrakte Algebra $\mathcal{A} = (\mathcal{T}, \Sigma, Q)$

konkrete Algebra (Σ -Algebra): Paar $A = (T, \Phi)$ wobei:

- T ist eine Menge mit $X \subseteq T$ (**Trägermenge**)
- $\Phi = \{f_T: T^n \rightarrow T \mid f/n \in \Sigma\}$,
d.h. jedem n -stelligen Operationssymbol f wird eine Funktion $f_T: \underbrace{T \times \dots \times T}_{n \text{ mal}} \rightarrow T$ zugeordnet
- die Gesetze Q sind durch die Funktionen f_T erfüllt

Man sagt auch: A ist ein **Exemplar der algebraischen Struktur \mathcal{A}** .
oder: A ist eine **Implementierung von \mathcal{A}**

Beispiel: die natürlichen Zahlen mit Addition implementieren ein kommutatives Monoid:

- die Signatur entspricht einem Monoid
- die Gesetze eines kommutativen Monoids sind erfüllt



Weitere Beispiele:

- \mathbb{R} , \mathbb{Q} , \mathbb{C} sind Exemplare der algebraischen Struktur Körper.

also:

- Es kann mehrere verschiedene Exemplare (Implementierungen) derselben algebraischen Struktur geben.
- Manche konkreten Algebren sind identisch mit ihren abstrakten Algebren, z.B. die Mengenalgebra und boolesche Algebra.

⇒ unterscheide abstrakte und konkrete Algebra nur bei Bedarf

- in der Mathematik sehr häufig keine Unterscheidung
- in der Informatik wegen unterschiedlicher Implementierungen Unterscheidung meist unvermeidlich



- auf mehrsortige Algebren könnte man auch verzichten
 - alle Datentypen zusammennehmen: \mathbb{N} , \mathbb{R} , \mathbb{Q} , \mathbb{C} , {wahr,falsch}, Listen, Zeichenreihen, ...
 - alle Operationen nur partiell definieren (keine Division für Wahrheitswerte oder Zeichenreihen, ...)
 - in Maschinensprachen tut man das: keine Unterscheidung, ob eine Folge von 32 Bit eine ganze Zahl oder eine Gleitpunktzahl darstellt
- aber:
 - dann sind die Möglichkeiten, die Richtigkeit eines Programms anhand des Programmtexts einzusehen, eingeschränkt:
 - Laufzeitfehler wegen falscher Typen
 - riesiger Testaufwand bei großen Programmen
 - Keine Möglichkeit für Polymorphie („+“ bei Bedarf Festpunkt-/Gleitpunktaddition)
- also:
 - mehrsortige Algebren eine praktische Notwendigkeit, in der Theorie reichen homogene Algebren



Ziel: auf einer Mengenfamilie $\{N_s\}_{s \in S}$ Funktionen $f \in F$ so definieren, daß eine vorgegebene Abbildung h einen Homomorphismus darstellt

- Sei $\Sigma = (S, F)$ eine Signatur,
- $\mathcal{A} = (M, \Phi)$ eine Σ -Algebra, die Gesetze Q erfüllt
- $N_s, s \in S$ eine Familie von Mengen
- $h_s: M_s \rightarrow N_s, s \in S$ surjektive Abbildungen
- für alle $f: s_1 \times \dots \times s_n \rightarrow s \in F$ sei f_N so definiert, daß die Homomorphiebedingung gilt, d.h. $f_N(h_{s_1}(a_1), \dots, h_{s_n}(a_n)) = h_s(f_M(a_1, \dots, a_n))$

Behauptung: $\mathcal{B} = (N, \Psi)$ mit $\Psi = \{f_N \mid f \in F\}$ ist eine Σ -Algebra, die die Gesetze Q erfüllt und $h: \mathcal{A} \rightarrow \mathcal{B}$ ist ein Σ -Algebra-Homomorphismus.

Man sagt, h **induziert eine algebraische Struktur** auf N .

praktische Anwendung: N soll zur Implementierung von \mathcal{A} dienen, enthält aber zuerst noch nicht einmal die nötigen Grundoperationen



Beispiel: Betrachte die Algebra $Z = (\mathbb{Z}, \{0/0, +/2\})$ der ganzen Zahlen.

Sei $h: \mathbb{Z} \rightarrow \{0, 1\}$ mit $h(n) = \begin{cases} 0 & \text{falls } n \text{ gerade} \\ 1 & \text{falls } n \text{ ungerade} \end{cases}$

$0 + 0 = 0$	zwei gerade Zahlen addieren ergibt gerade Zahl
$0 + 1 = 1$	gerade und ungerade Zahl addieren ergibt ungerade Zahl
$1 + 0 = 1$	ungerade und gerade Zahl addieren ergibt ungerade Zahl
$1 + 1 = 0$	zwei ungerade Zahlen addieren ergibt gerade Zahl

$\Rightarrow 0$ ist neutrales Element.

Man rechnet leicht nach, daß auch das Assoziativgesetz gilt.



Die **Äquivalenzrelation** $\equiv \subseteq U \times U$ ist

- reflexiv: $\forall x \in U: x \equiv x$
- symmetrisch: $\forall x, y \in U: (x \equiv y) \leftrightarrow (y \equiv x)$
- transitiv: $\forall x, y, z \in U: ((x \equiv y) \wedge (y \equiv z)) \rightarrow x \equiv z$

Beispiel (fortgesetzt):

$U = \mathbb{N}$ und $n \equiv m$ genau dann, wenn

- entweder n und m beide gerade
- oder n und m beide ungerade

alternative Formulierung:

- betrachte $h: \mathbb{N} \rightarrow \{0, 1\}$ mit

$$h(n) = \begin{cases} 0 & \text{falls } n \text{ gerade} \\ 1 & \text{falls } n \text{ ungerade} \end{cases}$$

- $n \equiv m$ genau dann, wenn $h(n) = h(m)$

Einsicht:

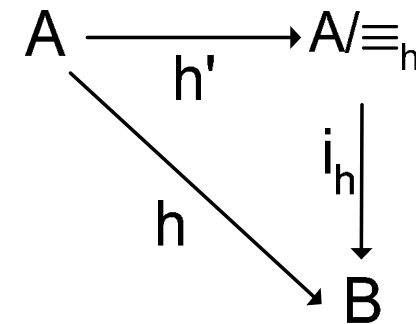
- Jeder Homomorphismus $f: A \rightarrow B$, A Algebra, definiert eine Äquivalenzrelation auf A : $a \equiv b$ gdw. $f(a) = f(b)$



Satz: Gegeben seien Σ -Algebren $\mathcal{A} = (M, \Phi)$, $\mathcal{B} = (N, \Psi)$ und ein Homomorphismus $h: \mathcal{A} \rightarrow \mathcal{B}$. Dann gibt es einen Isomorphismus $i_h: M/\equiv_h \rightarrow \text{Bild}(h)$ mit $h = i_h \circ h'$, wobei $h': M \rightarrow M/\equiv_h$ die Abbildung $h'(x) = [x]$ ist.

Beweis:

- Definiere $i_h: M/\equiv_h \rightarrow N$ durch $i_h([x]) = h(x)$
- i_h ist bijektive Funktion (Übung)
- i_h ist Homomorphismus
- $h = i_h \circ h'$ nach Konstruktion



- Folgerung: Falls $h: \mathcal{A} \rightarrow \mathcal{B}$ ein Epimorphismus ist, so ist \mathcal{B} isomorph zu M/\equiv_h
- praktische Anwendung: Die Gesetze Φ einer abstrakten Datenstruktur \mathcal{A} definieren eine Äquivalenzrelation \equiv und alle Implementierungen \mathcal{B} von \mathcal{A} sind auch Implementierungen von M/\equiv . Wenn man M/\equiv studiert, hat man die Eigenschaften, die allen Implementierungen gemeinsam sind.



Beispiel: Betrachte Grundtermalgebra G zu folgender Signatur:

- $0: \quad \quad \quad \rightarrow \mathbb{N}$
- $\text{succ}: \quad \mathbb{N} \quad \rightarrow \mathbb{N}$
- $\text{plus}: \quad \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$

Sei $A = (\mathbb{N}, \Phi)$ Algebra mit

- $0_{\mathbb{N}}: \quad \quad \quad \rightarrow \mathbb{N}$
- $\text{succ}_{\mathbb{N}}: \quad \mathbb{N} \quad \rightarrow \mathbb{N}$
- $\text{plus}_{\mathbb{N}}: \quad \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$

$$\left. \begin{aligned} 0_{\mathbb{N}} &= 0 \\ \text{succ}_{\mathbb{N}}(n, m) &= n + 1 \\ \text{plus}_{\mathbb{N}}(n, m) &= n + m \end{aligned} \right\} \text{Eingebaut!}$$

Homomorphismus $h: G \rightarrow A$:

- $h(0) = 0_{\mathbb{N}}$
- $h(\text{succ}) = \text{succ}_{\mathbb{N}}$
- $h(\text{plus}) = \text{plus}_{\mathbb{N}}$

$$\begin{array}{l} 1 + 1 \quad \rightarrow \text{succ}(1) = 2 \\ 2 \end{array}$$



Ziel: Beweis von Aussagen über fundierte Halbordnungen

Sei (U, \leq) fundierte Halbordnung und $<$ zugehörige strenge Halbordnung.

Behauptung: Aussage $P(x)$ ist wahr, falls gilt:

{ Wenn $P(z)$ wahr für alle $z < y$, dann ist $P(y)$ wahr.

Beweis: Sei $U_P = \{x \in U \mid P(x) \text{ wahr}\}$

Annahme: $M = U \setminus U_P \neq \emptyset$

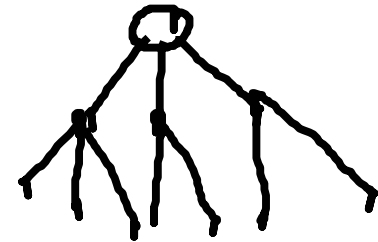
\Rightarrow Es gibt minimales Element $m \in M$, weil (U, \leq) fundiert ist

$\Rightarrow P(m)$ ist falsch, weil $m \in M$

$\Rightarrow P(z)$ ist wahr für alle $z \in U$ mit $z < m$, weil m minimal in M ist

$\Rightarrow P(m)$ ist wahr - **Widerspruch!**

Beobachtung: $P(y)$ muß auch gelten, wenn $\{z \in U \mid z < y\} = \emptyset$



1. f_i Konstanten
2. P gültig für
 t_1, \dots, t_n
Beh.: $f(t_1, \dots, t_n)$
 $f \in \Sigma$

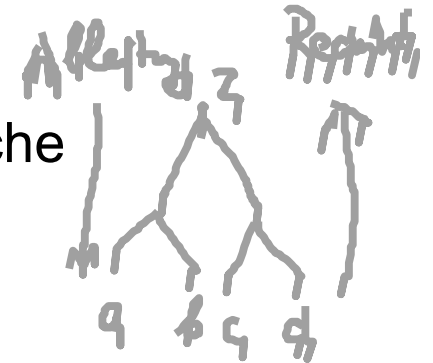
- gegeben sei eine Termalgebra $\mathcal{A} = (\mathcal{T}, \Phi)$ zur Signatur Σ
- jeder Term $t \in \mathcal{T}$ ist
 - entweder eine Konstante $t = k$
 - oder entsteht aus Konstanten durch sukzessive Anwendung von Operationen $f \in \Sigma$ auf einfachere Terme t_i : $t = f(t_1, \dots, t_n)$
 - „einfacher“: die t_i enthalten weniger Funktionsanwendungen als t
- **Halbordnung auf \mathcal{T} : $t' \leq t$, wenn der Term t' als Operand in t vorkommt:**
 - $t' = t$
 - oder $t = f(\dots, t', \dots)$
 - oder (transitive Hülle) $t = f(\dots, t'', \dots)$ und t' ist Operand von t''
- Ein Term t enthält nur endlich viele Funktionsanwendungen (sonst könnte man ihn nicht hinschreiben)
- wenn $t' \leq t$, dann ist t' einfacher als t
- also: **die Halbordnung der Terme ist artinsch (und daher fundiert):** jede absteigende Kette $\dots \leq t'' \leq t' \leq t$ bricht nach endlich vielen Schritten ab

$$t \geq t' \geq t'' \dots$$



Beobachtung:

\mathcal{T} terminiert genau dann für alle $x \in \Sigma^*$, wenn \Rightarrow^* noethersche Halbordnung ist



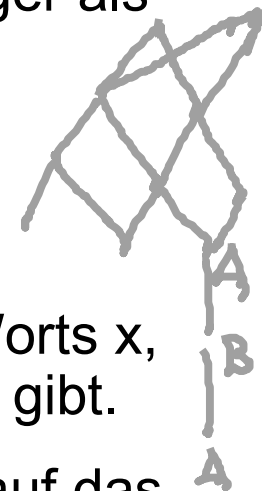
Spezialfall: Reduktion von $w \in \Sigma^*$ auf Z einer anständigen kontextfreien Grammatik $G = (\Sigma, N, P, Z)$

- Produktionen sind terminalisierend oder rechte Seite ist länger als linke Seite

Aber: Endet Semi-Thue-System immer mit dem selben Wort?

In einer Halbordnung \leq heißt ein Wort w Normalform eines Worts x , wenn es maximal ist, wenn es also kein w' mit $w \leq w'$, $w \neq w'$ gibt.

Also ist jedes Wort w mit dem eine Ableitung $x \Rightarrow^* w$ endet, auf das also keine weiteren Regeln mehr anwendbar sind, eine Normalform von x



$+ | \rightarrow | +$

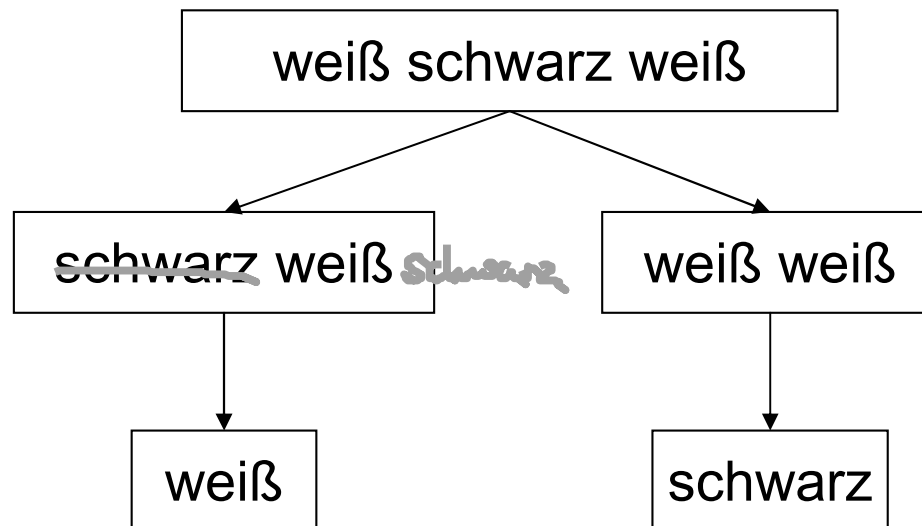
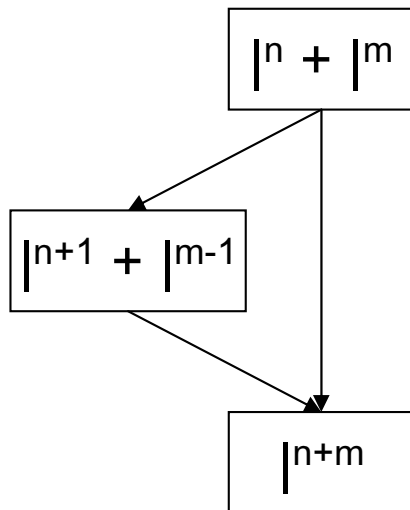
$+ \rightarrow \varepsilon$

schwarz schwarz \rightarrow schwarz

weiß weiß \rightarrow schwarz

weiß schwarz \rightarrow weiß

schwarz weiß \rightarrow schwarz }



▪ **Beispiel Keller:**

- Definiere $x \prec y$ genau dann, wenn x aus y durch Anwenden eines der Gesetze K1-K4 entsteht.

⇒ x enthält weniger Operationen als y

⇒ \prec^* ist eine **noethersche Halbordnung**

$$\underbrace{\text{pop}(\dots)}_{y} = x$$

$$\text{push}(\text{pop}(s), t)$$

$$\rightarrow \text{push}(\text{pop}(\text{push}(s', t'), t), t')$$

▪ \prec^* ist **lokal konfluent** (Übung)

⇒ Jeder Term der Sorte $\text{stack}(T)$ hat eindeutige **Normalform**, da aufgrund des Diamantenlemmas \prec^* konfluent ist.

⇒ Normalformen der Sorte „Keller“

- enthalten keine der Operationen: pop, top, isEmpty
- haben die Form: $\text{push}(\dots(\text{push}(\text{createStack}, n_1), \dots), n_k)$
- repräsentieren Keller



- Das Beispiel ist leider nicht beliebig verallgemeinerbar:
 - die durch die Gesetze induzierte Ordnungsrelation $x \prec y$ ist
 - oft nicht noethersch
 - oft nicht konfluent (mehrere oder keine Normalform)
- Beispiele:
 - arithmetische Ausdrücke im Körper der rationalen oder reellen Zahlen
 - Datentypen mit Gesetzen wie Assoziativität, Kommutativität, die die Länge nicht reduzieren



▪ Strikte Auswertung

(*strict, eager evaluation*)

- Alle Argumente werden ausgewertet, bevor die Funktion ausgewertet wird
- Bei $\text{fi rst } (g \ x) \ (h \ y)$ also zunächst $(g \ x)$ und $(h \ y)$, dann fi rst .

▪ Faule Auswertung

(*lazy evaluation*)

- Versuch, die Funktion auszuwerten. Argumente nur bei Bedarf berechnen.
- Bei $\text{fi rst } (g \ x) \ (h \ y)$ also zunächst fi rst . Dafür wird allein $(g \ x)$ benötigt. $(h \ y)$ wird nicht berechnet.
- Fast alle modernen Programmiersprachen werten logisches Und und logisches Oder faul aus.
- Haskell verwendet faule Auswertung, soweit nicht anders ausgewiesen.



- Problem: Wie formuliert man Terme M , k , t so, daß gilt
 - „für alle Mengen M gilt $M \cup M = M^c$ oder
 - „für alle Keller k und alle Elemente t gilt $\text{pop}(\text{push}(k,t)) = t$ “
- Lösung: Zulassen von Variablen in Termen
 - sie heißen auch logische Variable oder Unbestimmte
 - sie werden durch **Variablenbezeichner** notiert
- **Initiale Termalgebra** $\mathcal{I}(\Sigma, V)$ zur Signatur Σ mit Variablenmenge V
 - Definition der Konstantenmenge X als $X = \Sigma^{(0)} \cup V$
 - V bezeichnet eine abzählbar unendliche Menge von Variablen
 - wenn es noch weitere Konstante gibt, entfällt der Zusatz „initial“
- Einsicht: in jeder Menge von Formeln (Termen), die man hinschreiben kann, kommen nur endlich viele Variable vor, niemals abzählbar unendlich viele!
 - V abzählbar, damit man immer noch eine neue Variable finden kann



Beispiel: Keller

$$\text{pop}(\text{push}(\text{push}(\text{createStack}, 1), 2)) \stackrel{K4}{=} \text{push}(\text{createStack}, 1)$$

(Handwritten annotations: A bracket labeled 'k' spans the inner 'push' arguments in both terms. Another bracket labeled 'K4' spans the entire equality.)

(Erinnerung: K4: $\text{pop}(\text{push}(k, t)) = k$)

Beobachtung: Es gibt eine Substitution der linken Seite des Axioms, so daß ein Unterterm von $\text{pop}(\text{push}(\text{push}(\text{createStack}, 1), 2))$ entsteht.

Term t paßt auf Term t'

genau dann, wenn es eine Substitution σ gibt, so daß $t'\sigma = t$ gilt.

(man sagt, t' ist **allgemeiner** als t bzw. t ist eine **Spezialisierung** von t')

Beispiel:

$$\sigma = [\text{push}(\text{createStack}, 1) / k, 2 / t]$$

- $\text{pop}(\text{push}(\text{push}(\text{createStack}, 1), 2))$ paßt auf $\text{pop}(\text{push}(s, t))$
- $\text{pop}(\text{pop}(\text{push}(\text{push}(\text{createStack}, 1), 2)))$ paßt nicht auf $\text{pop}(\text{push}(s, t))$



Eigenschaften:

- Assoziativgesetz gilt

Neutrales Element: leere Substitution $\chi = []$

⇒ Substitutionen bilden bzgl. Hintereinanderausführung ein Monoid

⇒ Relation „paßt auf“ ist reflexiv, transitiv und Quasiordnung

⇒ Relation „paßt auf“ ist Quasiordnung



- Aber keine Halbordnung: für Variable x, y paßt $f(x)$ auf $f(y)$ und umgekehrt

⇒ starke Zusammenhangskomponenten: Terme, die sich durch Umbenennen von Variablen ineinander überführen lassen

⇒ „paßt auf“ ist Halbordnung bis auf Umbenennungen

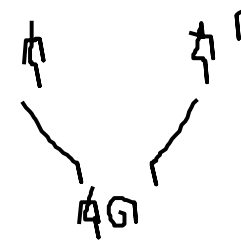
⇒ größtes Element ist eine einzelne Variable x

⇒ kleinste obere Schranken existieren



Untere Schranke zweier Terme t, t' :

- Es gibt Substitution σ , so daß $t\sigma = t'\sigma$.
- σ heißt **Unifikator**.



Beispiel: $f(g(x), y)$ und $f(y, g(x))$

Unifikator $\sigma = [g(x)/y]$, untere Schranke $f(g(x), g(x))$

$$\begin{aligned} t\sigma &= f(g(x))\sigma \\ &= f(g(x)) \end{aligned}$$

Gemeinsame untere Schranken müssen nicht existieren:

x und $f(x)$ haben keine gemeinsame untere Schranke.

Größte untere Schranke zweier Terme t, t' :

- Es gibt Substitution σ so, daß $t\sigma = t'\sigma$ und alle Terme s , auf die $t\sigma$ paßt, nur durch Variablen-Umbenennung aus $t\sigma$ entstanden sind.
- σ heißt **allgemeinster Unifikator**.

*most general unifier
mgu*

Beispiel: $f(g(x), y)$ und $f(y, g(x))$

allgemeinster Unifikator $\sigma = [g(x)/y]$

Unifikation ist zentrale Grundoperation des logischen Programmierens



▪ **Termersetzungsregeln:**

- $\text{sort}(\text{nil}) \rightarrow \text{nil}$
- $\text{sort}(\text{cons}(x,l)) \rightarrow \text{insert}(x, \text{sort}(l))$
- $\text{insert}(x,\text{nil}) \rightarrow \text{cons}(x, \text{nil})$
- $\text{insert}(x, \text{cons}(y, l)) \rightarrow \text{if}(x \leq y, \text{cons}(x, \text{cons}(y, l)), \text{cons}(y, \text{insert}(x, l)))$
- $\text{if}(O, l_1, l_2) \rightarrow l_2$
- $\text{if}(L, l_1, l_2) \rightarrow l_1$

- Der Term $x \leq y$ kann direkt durch O und L ersetzt werden.
- auch das kann man durch Termersetzungsregeln ausdrücken

Beispielauswertung:

$\text{sort}(\text{cons}(2, \text{cons}(1, \text{cons}(3, \text{nil}))))$

$\text{sort}([2,1,3])$

$\text{cons}(2, \text{cons}(1, \text{cons}(3, \text{nil})))$

$\Rightarrow \text{insert}(2, \text{sort}(\text{cons}(1, \text{cons}(3, \text{nil}))))$

gleichgültig welche Fortsetzung: Regel 2, Regel 4 das System konfluent



- Ein **Textersetzungssystem** über einem Zeichenvorrat V besteht aus einer endlichen Menge R von (Text-) Ersetzungsregeln (über V)
 - eine **Ersetzungsregel** ist gegeben durch ein Paar $(v, w) \in V^* \times V^*$ und wird in der Form $v \rightarrow w$ geschrieben
 - die ansonsten üblichen Klammern ` \langle ` und ` \rangle ` für Wörter aus V^* werden aufgrund der besseren Lesbarkeit im Zusammenhang mit den Ersetzungsregeln weggelassen
 - Ersetzungsregeln können auf beliebige Teilwörter angewendet werden:
 - Eine Ersetzung $s \rightarrow t$ heißt **Anwendung** der Regel $v \rightarrow w$, falls es Wörter $a, v, w, z \in V^*$ gibt, so daß gilt:
 - $s = a v z,$ $t = a w z$
 - Ein Wort $s \in V^*$ heißt **terminal** in R , falls keine der im Textersetzungssystem vorhandenen Ersetzungsregeln angewendet werden kann

