

# Übungen zu Informatik I Wintersemester 03/04

Übungsleiter: Dipl.-Inform. Tom Gelhausen



### ▪ 3.3 Algebraische Strukturen und Algebren

- Gegeben: max. abzählbar unendlich!
  - $\Sigma$ : Signatur (also Menge von Operationen  $\Sigma = \Sigma^{(0)} \cup \Sigma^{(1)} \cup \dots$ )
  - $X$ : Elementaroperanden ( $\Sigma^{(0)} \subseteq X$ )
  - $\mathcal{T}$ : Menge der korrekten Terme zu  $\Sigma$  und  $X$
  - $Q$ : Menge von Gesetzen (Axiome), die bedeutungstreue Umformungen  $f \rightarrow f'$  definieren ( $f, f' \in \Sigma$ )

**Beispiele (für Axiome):** die Halbgruppen- und Monoidgesetze HG1, HG2 (nur bei kommutativen HG), die Gesetze V1-V10 der booleschen Algebra
- Das Tripel  $\mathcal{A} = (\mathcal{T}, \Sigma, Q)$  bildet eine **algebraische Struktur**, bzw. eine **abstrakte Algebra**. Dabei gilt für zwei Terme  $t, t' \in \mathcal{T}$ :  
 $t \equiv t' \Leftrightarrow t$  kann nach den Gesetzen  $Q$  in  $t'$  umgeformt werden.
- **Beispiele:** Halbgruppen, Monoide, Verbände, Gruppen, Ringe, Körper, Vektorräume, boolesche Algebren



## Theorie

Abstrakte Algebra

Konkrete Algebra

Grundtermalgebra

Axiomenmenge  $Q \mapsto$

## „in Haskell“

Abstrakter Datentyp

(konkreter) Datentyp

Konstruktoren

Funktionen (inkl. Sign.)

## im Beispiel

kommutatives Monoid

$(\text{Nat}, +)$

Null, Succ(Null), Succ(Su ...

+

- $\Sigma = \Sigma^{(0)} \cup \Sigma^{(1)} \cup \Sigma^{(2)}$

- $\Sigma^{(0)} = \{ \text{“Null”} \}$
- $\Sigma^{(1)} = \{ \text{“Succ”} \}$
- $\Sigma^{(2)} = \{ \text{“op”} \}$

- $X = \{ \text{Null}, \text{Succ} \}$

- $\mathcal{T} = \{ \langle \text{Term} \rangle ::= \text{Null} \mid \text{Nachfolger } \langle \text{Term} \rangle \mid \text{'(op' } \langle \text{Term} \rangle \langle \text{Term} \rangle \text{' )' } \}$

- $Q = \{ \text{algebraische Abgeschlossenheit} \}$   
 Assoziativgesetz  
 Kommutativgesetz  
 Einselement

```
data Nat = Null | Succ Nat deriving (Eq, Show)
plus :: Nat -> Nat -> Nat           -- Abgeschlossenheit
plus Null a = a                   -- Einselement
plus a Null = a                   -- Einselement
plus (Succ a) b = plus a (Succ b) -- der Rest
plus (plus a b) c = plus a (plus b c) -- unzulässig
plus a (plus b c) = plus (plus a b) c -- unzulässig
plus a b = plus b a               -- nicht erreichbar
```

- plus  $a \ b \in \mathbb{N}_0$ ,
- plus plus a b c = plus a plus b c,
- plus a b = plus b a
- plus Null a = a
- plus a Null = a }



- Erster Ansatz:

$$\text{fib } 0 = 0$$

$$\text{fib } 1 = 1$$

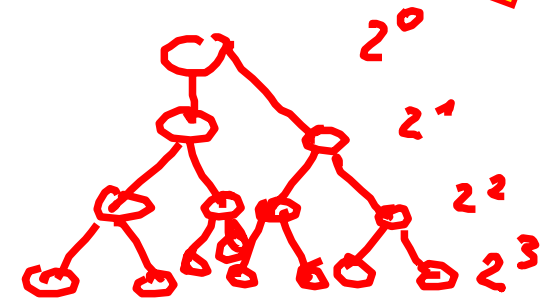
$$\text{fib } (n + 2) = (\text{fib } (n+1)) + (\text{fib } n)$$

$$t(n) = n.$$

$$\begin{cases} t(n) = t(t(n-1)) + t(t(n-2)) \\ t(n-1) = t(t(n-2)) + t(t(n-3)) \\ t(n-2) = t(t(n-3)) + t(t(n-4)) \end{cases}$$

$$\rightarrow t(n) = t(t(n-2)) + t(t(n-3)) + t(t(n-3)) + t(t(n-4))$$

**$O(n)$  offenbar falsch!**



- Keine Abgabe – keine Punkte!
- Aufgabe 3: Rekurrenzrelationen
  - Stoff der Vorlesung nächste Woche... Üben!!!!

- Pseudocode

```
function foo(int i, int j)
begin
    if (j-i<4) then return;
    bar (1, i);
    foo (i, i+(j-i)/4);
    for k:=0 to j-i
        bar (k, j);
    foo (i+(j-i)/4+1, j);
end
```



- Semi-Thue-System  $T=(\Sigma, T)$  mit

- endlichem Zeichenvorrat  $\Sigma$
- endlicher Regelmenge  $T$ 
  - Regeln  $l \rightarrow r$  mit  $l, r \in \Sigma^*$

$$\begin{aligned} \varepsilon &\rightarrow abc \\ abc &\rightarrow \varepsilon \end{aligned}$$

- Ausführung:

- Wähle eine beliebige anwendbare Regel
- Wähle eine beliebige passender Anwendungsstelle
- Wende die Regel an (Direkte Ableitung  $x \Rightarrow x'$ )
- Ausführung stoppt wenn keine Regel mehr anwendbar

- $x \Rightarrow^* x'$ : Endliche Folge direkter Ableitungen



- Markov-Algorithmus  $(\Sigma, R)$  mit
  - endlicher Alphabet  $\Sigma$
  - endlicher Folge von Regeln  $R$
  - Regeln  $l \rightarrow r$  mit  $l, r \in \Sigma^*$
  - haltende Regeln  $l \rightarrow \underline{\cdot} r$  mit  $l, r \in \Sigma^*$
  
- Ausführung:
  - Wähle die erste anwendbare Regel (Ordnung der Folge)
  - Wähle die erste passende Anwendungsstelle (v.l.n.r)
  - Wende die Regel an
  - Ausführung stoppt nach haltender Regel



- Wir definieren eine Grammatik als Quadrupel  $G = (\Sigma, N, P, S)$ , wobei
  - $\Sigma$  die Menge der Terminale,
  - $N$  die Menge der Nichtterminale (wobei  $\Sigma \cap N = \emptyset$ ),
  - $S \in N$  das Startsymbol,
  - $(\mathcal{V}, P)$  ein Semi-Thue-System und
  - $\mathcal{V} = \Sigma \cup N$  das Vokabular von  $G$  ist.
  
- Sei  $G$  eine Grammatik. Dann bezeichnet

$$\mathcal{L}(G) = \{ w \in \Sigma^* \mid S \Rightarrow^* w \}$$

die Sprache, die von  $G$  erzeugt wird.



## ▪ CH-0 Grammatik

- Produktionen beliebiger Form ( $l \mapsto r$  ;  $l, r \in V^*$ )

## ▪ CH-1 Grammatik

- kontextsensitive Produktionen  
( $\underline{u}A\underline{v} \mapsto \underline{u}r\underline{v}$  ;  $A \in N$  ;  $r \in V^+$  ;  $u, v \in V^*$ )
- $S\varepsilon$ -Produktionen ( $S \mapsto \varepsilon$ ) nur, wenn  $S$  auf keiner rechten Seite vorkommt

## ▪ CH-2 Grammatik

- kontextfreie Produktionen ( $\underline{A} \mapsto r$  ;  $A \in N$  ;  $r \in V^*$ )

## ▪ CH-3 Grammatik

- entweder linkslineare ( $A \mapsto Bx$  ;  $A, B \in N$  ;  $x \in \Sigma$ ) oder rechtslineare Produktionen ( $A \mapsto xB$  ;  $A, B \in N$  ;  $x \in \Sigma$ )
- terminierende Produktionen ( $A \mapsto x$  ;  $A \in N$  ;  $x \in \Sigma$ )
- $\varepsilon$ -Produktionen ( $A \mapsto \varepsilon$  ;  $A \in N$ )



- Eine Sprache  $L$  ist vom Typ Chomsky  $X$  genau dann, wenn es eine Grammatik  $G$  mit  $\mathcal{L}(G) = L$  und  $G \in \text{CH-}X$  gibt. Die dementsprechend erzeugten Sprachen bezeichnet man als
  - Chomsky 0, (allgemeinen oder berechenbaren Sprachen),
  - Chomsky 1, (kontextsensitiven oder entscheidbaren Sprachen),
  - Chomsky 2, (kontextfreien Sprachen) und
  - Chomsky 3, (regulären Sprachen).
  
- Chomsky 3  $\subset$  Chomsky 2  $\subset$  Chomsky 1  $\subset$  Chomsky 0
  
- Hinweis: <http://www.infoeins.de/dokumente/Chomsky.pdf>



- Kettenproduktionen

- $A \mapsto B ; A, B \in N$
- lassen sich vollständig entfernen
- keine Beeinflussung des Sprachtyps

*CH-X  
Chomsky X*

- Nichtprimitive lineare Produktion

- $A \mapsto Bx ; A, B \in N ; x \in \Sigma^+$
- kann auf primitive lineare Produktionen ( $A \mapsto B'y ; y \in \Sigma$ ) zurückgeführt werden

~~*A  $\mapsto$  BBc*~~

- Nichtprimitive terminierende Produktion

- $A \mapsto x ; A \in N ; x \in \Sigma^+$
- kann auf primitive terminierende Produktionen ( $A \mapsto y ; y \in \Sigma$ ) zurückgeführt werden

- $\epsilon$ -Produktion ( $I \mapsto \epsilon, I \in V^*$ ): Zu Grammatiken mit  $\epsilon$ -Produktion gibt es für jede Chomsky-1/2/3-Sprache immer eine äquivalente Grammatik ohne  $\epsilon$ -Produktionen (abgesehen von  $S\epsilon$ -Produktionen)



- Eigentliche Theorie in Info III
- Faustregeln: die Beschreibung von L und die daraus resultierende maximale Sprachklasse
  - enthält Primzahlen etc.  $\Rightarrow L \in \text{Chomsky-0}$
  - enthält Klammerstrukturen  $\Rightarrow L \in \text{Chomsky-2}$
  - enthält unbeschränkte Exponenten,  
 $2\times$  die mehrfach auftreten  $\Rightarrow L \in \text{Chomsky-2}$   $a^n b^m c^n$
  - endliche Aufzählung  $\Rightarrow L \in \text{Chomsky-3}$

$((())()) (\dots)$



- Operatoren in fallender Präzedenz

Operator (Buch)	Synonyme
$\bar{C}x$	$\bar{x}, \neg x$
$x \wedge y$	
$x \vee y$	

- Spezielle Werte der Algebra
  - Top  $\top$
  - Bottom  $\perp$
- Beweise können nur auf die Axiome zurückgreifen
  - Axiome sind teilweise redundant
  - Keine Wahrheitstabellen möglich!



Die boolesche Algebra ist ein vollständiger, distributiver, komplementärer Verband  $\mathcal{B} = (U, \vee, \wedge, \complement)$  mit den Gesetzen:

V1 Assoziativität:  $(x \wedge y) \wedge z = x \wedge (y \wedge z)$   $(x \vee y) \vee z = x \vee (y \vee z)$

V2 Kommutativität:  $x \wedge y = y \wedge x$   $x \vee y = y \vee x$

V3 Idempotenz:  $x \wedge x = x$   $x \vee x = x$

~~V4~~ Verschmelzung:  $(x \vee y) \wedge x = x$   $(x \wedge y) \vee x = x$

~~V5~~ Distributivität:  $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$   
 $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$

~~V6~~ Modularität:  $x \vee (y \wedge z) = (x \vee y) \wedge z$  falls  $x \leq z$

V7 neutrale Elemente:  $x \wedge \perp = \perp$   $x \vee \perp = x$   
 $x \wedge \top = x$   $x \vee \top = \top$

V8 Komplement:  $x \wedge \complement x = \perp$   $x \vee \complement x = \top$

~~V9~~ Involution:  $\complement(\complement x) = x$

~~V10~~ De Morgan:  $\complement(x \wedge y) = \complement x \vee \complement y$   $\complement(x \vee y) = \complement x \wedge \complement y$




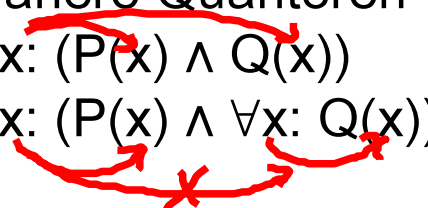
- Operatoren der Aussagenlogik in fallender Präzedenz

Operator (Buch)	Definition	Synonyme
$\neg$		$\overline{x}$
$\wedge$		
$\vee$		
$\rightarrow$	$\neg x \vee y$	$\Rightarrow$
$\equiv$	$(x \rightarrow y) \wedge (y \rightarrow x)$ oder $(x \wedge y) \vee (\overline{x} \wedge \overline{y})$	$\Leftrightarrow, \Leftrightarrow$



- Literale
  - Variable, negierte Variable
  
- Konjunktive Normalform (KNF)
  - Konjunktion von Disjunktionen von Literalen
  - Beispiel:  $(x \vee \bar{y} \vee z) \wedge (\bar{x} \vee y) \wedge (y \vee z)$   
*Dis. Kon.*
  
- Disjunktive Normalform (DNF)
  - Disjunktion von Konjunktionen von Literalen
  - Beispiel:  $(x \wedge \bar{y} \wedge z) \vee (\bar{x} \wedge y) \vee (y \wedge z)$
  - Zeigt unmittelbar die Erfüllbarkeit
  
- Überführung in DNF / KNF
  - Operatoren in  $\neg$ ,  $\wedge$ ,  $\vee$  auflösen
  - Distributivität und deMorgan einsetzen



- Erweitert die Aussagenlogik um Quantoren
  - $\forall$  Aussage gilt für alle Belegungen der Variable
  - $\exists$  Aussage gilt für mindestens eine Belegung der Variable
- Freie und gebundene Variable
  - Quantoren wirken nach rechts:  
 $P(x) \wedge \forall x: Q(x)$   

  - Nähere Quantoren verdecken fernere Quantoren:  
 $\exists x: (P(x) \wedge Q(x))$   
 $\exists x: (P(x) \wedge \forall x: Q(x))$   

  - Klammern schützen!  
 $(\forall x: P(x)) \wedge Q(x)$   
 $\forall x: P(x) \wedge Q(x)$   
 $\forall x: (P(x) \wedge Q(x))$

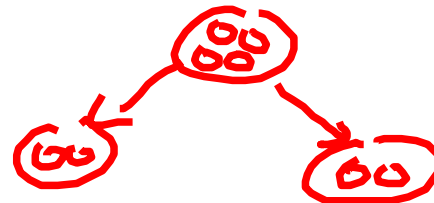
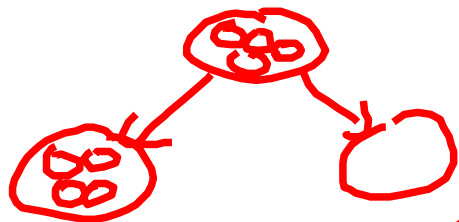


$$\neg \forall x: P(x) \Leftrightarrow \exists x: \neg P(x)$$

$$\exists x \forall y: P(x,y) \Rightarrow \forall y \exists x: P(x,y) \text{ (nicht umgekehrt!)}$$

$$\forall x: P(x) \wedge \forall x: Q(x) \Leftrightarrow \forall x: (P(x) \wedge Q(x))$$

$$\forall x: P(x) \vee \forall x: Q(x) \Rightarrow \forall x: (P(x) \vee Q(x)) \text{ (nicht umgekehrt!)}$$



$$\exists x: (P(x) \vee Q(x)) \Leftrightarrow \exists x: P(x) \vee \exists x: Q(x)$$

$$\exists x: (P(x) \wedge Q(x)) \Rightarrow \exists x: P(x) \wedge \exists x: Q(x) \text{ (nicht umgekehrt!)}$$



## Prädikaten

- bereinigte Form
  - Nur noch  $\forall, \exists, \wedge, \vee, \neg$
- Pränexform:
  - Alle Quantoren stehen vorne
  - Beachte: niemals  $\exists$  vor  $\forall$  ziehen!
- Skolemform
  - Grundidee: existenzquantisierte Variable ist Funktion der vorangehenden allquantisierten Variablen
  - Entferne Quantor, ersetze an ihn gebundene Variable  $v$  durch Skolemfunktionen  $sk_v(x_1, \dots, x_k)$

$$\forall x \forall y \exists z : P(x, y, z)$$

$sk_2(z)$



- [...] ich habe eine Frage bzgl. des Lambda-Kalküls. Verstehe ich die Definition der  $\eta$ -Konversion auf Folie 9 Kapitel 5:
  - $t$  keine lambda-Abstraktion  $\Rightarrow (t \ t')$  darf durch  $t$  ersetzt werden insoweit richtig, dass folgendes gilt:
    - $(\lambda a.(a)) \ 6 \ 7$
    - $= (6) \ 7$
    - $(6 \text{ ist keine Lambda-Abstraktion } \Rightarrow)$
    - $= 6$

Vielen Dank im Voraus Mit freundlichen Grüßen [...]

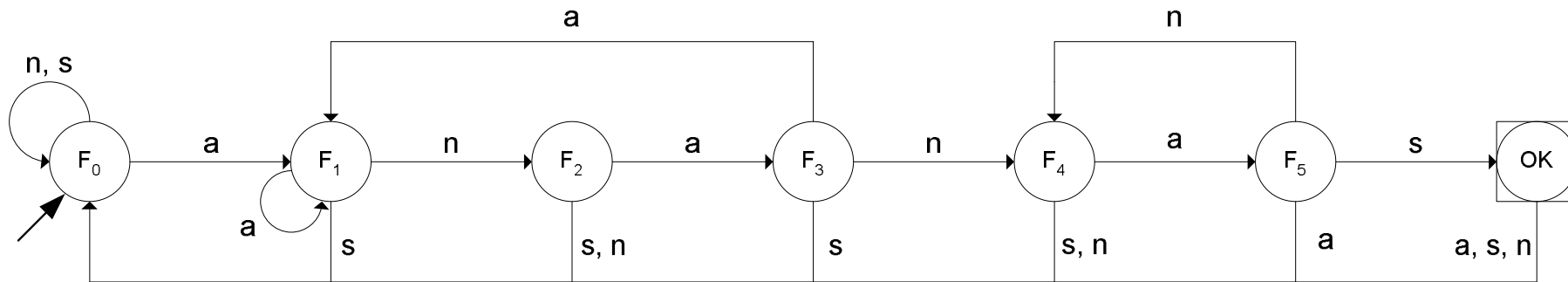
## ▪ Initiale Termalgebra $\Lambda$ :

- Variable  $v$  wenn  $v \in V$
- Lambda-Abstraktion  $\lambda v.F$  wenn  $F \in \Lambda$
- Funktionsanwendung  $(F \ G)$  wenn  $F, G \in \Lambda$
- $\alpha$ -Konversion: ersetze  $\lambda v.t$  durch  $\lambda v'.t[v'/v]$  wenn Subst. zul.
- $\beta$ -Konversion: ersetze  $((\lambda v.t) \ t')$  durch  $t[t'/v]$  wenn Subst. zul.
- $\eta$ -Konversion: ersetze  $(\lambda v.(t \ v))$  durch  $t$  wenn  $v$  in  $t$  nicht frei

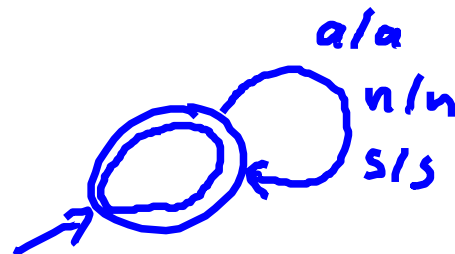


Gegeben: Eine Sprache  $(a+n+s)^*ananas$

- Ein Automat, der ein Wort der Sprache *akzeptiert*.



- Ein Automat, der ein Wort *kopiert*.



- Haskell
  - Regelung Prelude.hs
  - Bezeichner (gilt auch für SQL und Prolog)
  
- 2. Korrekturblatt zum Buch
  
- Folien werden zusammengeführt und bereinigt



- Sowie die Musterlösungen finden Sie unter

**<http://www.infoeins.de/uebungsblaetter.php>**

