

# Grundlagen Theoretischer Informatik 3

SoSe 2010 in Trier

Henning Fernau

Universität Trier

fernau@uni-trier.de

## Organisatorische Ergänzungen

Die Tutorensprechstunde montags wird auf denselben Ort und dieselbe Zeit dienstags verlegt.

Vorlesungsumlegungen:

Wie besprochen, findet VL4 am MI, 5.5., 10.05-11.35 statt, und zwar im H 6.

Aus persönlichen Gründen würde ich ebenfalls die VL5 verlegen, und zwar auf MI, 12.5., 10.05-11.35 im H 12.

Für die Abschlussklausur gibt es jetzt einen festen Termin und Ort:  
DI, 20.7., 9:00-11:00 im HZ 204.

## Realistischere While-Programme : Syntax

Das Grundalphabet  $A$  besteht aus:

*Schlüsselwörtern:*  $S = \{\mathbf{skip}, \leftarrow, ;, \mathbf{if}, \mathbf{then}, \mathbf{else}, \mathbf{fi}, \mathbf{while}, \mathbf{do}, \mathbf{od}\}$

*Booleschen Variablen:*  $V_{\mathbb{B}}$

*Booleschen Operatoren:*  $O_{\mathbb{B}} = \{\neg, \wedge, \vee, \mathbf{true}, \mathbf{false}, \dots\}$

*Zahlvariablen:*  $V_{\mathbb{N}}$

*Zahlenoperatoren:*  $O_{\mathbb{N}} = \{0, 1, +, -, *, /, \dots\}$

*Zahlenprädikaten:*  $P_{\mathbb{N}} = \{=, <, >, \leq, \geq, \dots\}$

*Klammern:*  $K = \{(, )\}$ .

Mit Hilfe einer kontextfreien Grammatik könnte man die Sprachen

$WBA \subseteq (V_{\mathbb{B}} \cup O_{\mathbb{B}} \cup K)^*$  der *wohlgeformten Booleschen Ausdrücke* und

$WAA \subseteq (V_{\mathbb{N}} \cup O_{\mathbb{N}} \cup K)^*$  der *wohlgeformten arithmetischen Ausdrücke* definieren.

**Syntax von WHILE-Programmen** definiert formale Sprache  $L_{\text{while}} \subseteq A^*$ :

(1) *Leere Anweisung*: **skip**  $\in L_{\text{while}}$ .

(2) *Wertzuweisung*: Gilt  $v \in V_{\mathbb{B}}$  und ist  $e$  ein WBT, so gilt:  $v \leftarrow e \in L_{\text{while}}$ .

Ist  $x \in V_{\mathbb{N}}$  und ist  $w$  ein WAA, so gilt:  $x \leftarrow w \in L_{\text{while}}$ .

(3) *Anweisungsfolge*: Mit  $S_1$  und  $S_2$  aus  $L_{\text{while}}$  folgt auch  $S_1; S_2 \in L_{\text{while}}$ .

(4) *Verzweigung*: Mit WBT  $e$  und  $S_1$  und  $S_2$  aus  $L_{\text{while}}$  folgt auch

**if**  $e$  **then**  $S_1$  **else**  $S_2$  **fi**  $\in L_{\text{while}}$ .

(5) *Schleife*: Mit WBT  $e$  und  $S \in L_{\text{while}}$  folgt auch **while**  $e$  **do**  $S$  **od**  $\in L_{\text{while}}$ .

Nur Wörter über  $A$ , die wie in (1) bis (5) beschrieben konstruiert werden können, gehören zu  $L_{\text{while}}$ .

Ist  $p \in L_{\text{while}}$  ein WHILE-Programm mit vorkommenden Booleschen Variablen  $V_{\mathbb{B},p}$  und vorkommenden Zahlvariablen  $V_{\mathbb{N},p}$ , so lässt sich der Berechnungsstand in einer *Konfiguration*  $(S, \sigma_{\mathbb{B}}, \sigma_{\mathbb{N}})$  festhalten, mit  $S \in L_{\text{while}} \cup \{\lambda\}$ ,  $\sigma_{\mathbb{B}} \in (\mathbb{B} \cup \{\uparrow\})^{|V_{\mathbb{B},p}|}$ ,  $\sigma_{\mathbb{N}} \in (\mathbb{N} \cup \{\uparrow\})^{|V_{\mathbb{N},p}|}$ ,  $\uparrow$  meint undef.

## Konfigurationsübergänge definieren die Semantik von WHILE-Programmen

Hinweis: Dies ist ein einfacher Spezialfall für die Definition einer *operationellen Semantik*.

Mehr zu diesem Thema bieten Spezialvorlesungen.

Es seien  $(S, \sigma_{\mathbb{B}}, \sigma_{\mathbb{N}})$  und  $(T, \tau_{\mathbb{B}}, \tau_{\mathbb{N}})$  zwei Konfigurationen.

$(S, \sigma_{\mathbb{B}}, \sigma_{\mathbb{N}}) \Rightarrow (T, \tau_{\mathbb{B}}, \tau_{\mathbb{N}})$  gelte, wenn eine der nachstehenden Bedingungen erfüllt ist.

**Ziel:** Daraus kann man sofort die Semantik  $f_S$  eines WHILE-Programms  $S$  definieren als Ausgabewert  $\rho = (\rho_{\mathbb{B}}, \rho_{\mathbb{N}})$  bei Eingabe von  $\eta = (\eta_{\mathbb{B}}, \eta_{\mathbb{N}})$ , falls nämlich  $(S, \eta) \Rightarrow^* (\lambda, \rho)$  gilt. Hinweis: Ähnlichkeit mit Automaten und Grammatiken!

(1)  $(S = \mathbf{skip}; T$  oder  $(S = \mathbf{skip}$  und  $T = \lambda))$  und  $\tau = \sigma$

...

## Beispiel einer Konfigurationsfolge

Betrachte das folgende Programm:

$a \leftarrow 0;$

$b \leftarrow x;$

**while**  $b \geq y$  **do**

$b \leftarrow b - y;$

$a \leftarrow a + 1$

**od**

$a, b, x, y \in V_{\mathbb{N}}$ .

Betrachte  $\sigma(x) = 14, \sigma(y) = 5$ .

}  $=: S_1$

$(S, \sigma) = (a \leftarrow 0; b \leftarrow x; S_1, \sigma = (\uparrow, \uparrow, 14, 5))$

$\Rightarrow (b \leftarrow x; S_1, (0, \uparrow, 14, 5))$

$\Rightarrow (S_1, (0, 14, 14, 5))$

$\Rightarrow (b \leftarrow b - y; a \leftarrow a + 1; S_1, (0, 14, 14, 5))$

$\Rightarrow (a \leftarrow a + 1; S_1, (0, 9, 14, 5))$

$\Rightarrow (S_1, (1, 9, 14, 5))$

$\Rightarrow (b \leftarrow b - y; a \leftarrow a + 1; S_1, (1, 9, 14, 5))$

$\Rightarrow (a \leftarrow a + 1; S_1, (1, 4, 14, 5))$

$\Rightarrow (S_1, (2, 4, 14, 5))$

$\Rightarrow (\lambda, (2, 4, 14, 5))$

Was aber liefert das Programm als Ausgabe ( $(a, b)$ -Werte) für bel.  $(x, y)$ -Eingaben??

## Funktionale Interpretation von WHILE-Programmen

Mit der Schreibweise  $M \langle S \rangle$  anstelle von  $f_S$  für WHILE-Programme  $S$  können wir formulieren:

**Satz:** Für alle Variablenbelegungen  $\sigma$  gilt:

$$(1) M \langle \mathbf{skip} \rangle (\sigma) = \sigma.$$

$$(2) M \langle x \leftarrow t \rangle (\sigma) = \sigma[x/f_t(\sigma)] \text{ für alle WBT oder WAA } t.$$

$$(3) M \langle S; T \rangle (\sigma) = M \langle T \rangle (M \langle S \rangle (\sigma)).$$

$$(4) M \langle \mathbf{if } e \mathbf{ then } S_1 \mathbf{ else } S_2 \mathbf{ fi} \rangle (\sigma) = \begin{cases} M \langle S_1 \rangle (\sigma) & \text{falls } f_e(\sigma) = 1 \\ M \langle S_2 \rangle (\sigma) & \text{falls } f_e(\sigma) = 0 \end{cases}$$

$$(5) M \langle \mathbf{while } e \mathbf{ do } S \mathbf{ od} \rangle (\sigma) = \begin{cases} M \langle \mathbf{while } e \mathbf{ do } S \mathbf{ od} \rangle (M \langle S \rangle (\sigma)) & \text{falls } f_e(\sigma) = 1 \\ \sigma & \text{falls } f_e(\sigma) = 0 \end{cases}$$

## Die Hoaresche Logik

Ein *Hoarescher Ausdruck* ist eine Zeichenkette der Form

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

Dabei ist  $p$  ein WBT (die *Vorbedingung* von  $S$ ),  $S \in L_{\text{while}}$  und  $q$  ein WBT (die *Nachbedingung* von  $S$ ). Das Paar  $(p, q)$  heißt auch *Spezifikation*.

Bedeutung: Für alle Variablenbelegungen  $\sigma$  soll gelten:

Ist  $f_p(\sigma) = 1$  oder terminiert  $S$  auf  $\sigma$  nicht, so gilt auch  $f_q(\mathcal{M}\langle S\rangle(\sigma)) = 1$ .

Wir schreiben dann auch:  $\mathcal{M}\langle\{p\}S\{q\}\rangle(\sigma) = 1$ .

## Die Hoaresche Logik Ein letztes Beispiel

Betrachte das folgende Programm:

$a \leftarrow 0;$

$b \leftarrow x;$

**while**  $b \geq y$  **do**

$b \leftarrow b - y;$

$a \leftarrow a + 1$

**od**

$a, b, x, y \in V_{\mathbb{N}}$ .

Wir wissen bereits:

$\{\mathbf{true}\} a \leftarrow 0; b \leftarrow x \{a = 0 \wedge b = x\}$ .

}  $\equiv S_1$

Gilt  $\sigma(x) < \sigma(y)$ , so auch  $(M \langle a \leftarrow 0; b \leftarrow x \rangle (\sigma))(b) < \sigma(y) = (M \langle a \leftarrow 0; b \leftarrow x \rangle (\sigma))(y)$ , also wird der Schleifenrumpf nie betreten, und die "Ausgabe" liefert für das Variablenpaar  $(a, b)$ :  $(0, \sigma(x))$ .

Wie oben sieht man durch Induktion über die Anzahl  $i$  von Schleifenrumpfdurchläufen für eine Belegung  $\sigma_i$  am Ende des  $i$ -ten Durchlaufs:

1.  $\sigma_i(a) = i$  und
2.  $\sigma_i(b) = \sigma(x) - i * \sigma(y)$ .

Also gilt das auch bei Schleifenabbruch, wobei dann zusätzlich  $\sigma_i(b) < \sigma_i(y) = \sigma(y)$  gilt. Das gewährt, das schließlich das Variablenpaar  $(a, b)$  den Wert

$$(\sigma(x) \text{ DIV } \sigma(y), \sigma(x) \text{ mod } \sigma(y))$$

enthält.

## Der Hoaresche Kalkül

Noch etwas Notation:

Sind  $p, t$  WBTs und  $x \in V_{\mathbb{B},p}$ , so bezeichne  $p_x^t$  denjenigen WBT, der aus  $p$  dadurch entsteht, dass jedes Vorkommen von  $x$  in  $p$  durch  $(t)$  ersetzt wird.

Entsprechend:  $p_x^t$  für WBT oder WAA  $p$ ,  $x \in V_{\mathbb{N},p}$  und WAA  $t$ .

**Beispiel 1:** Für den WBT  $p = (x \leq x * y)$  gilt:  $p_x^{1+1} = ((1 + 1) \leq (1 + 1) * y)$ .

### Axiomenschemata

(S)  $\{p\}$  **skip**  $\{p\}$  für alle WBT  $p$ .

(Z) *Zuweisungsschema*  $\{p_x^t\}$   $x \leftarrow t$   $\{p\}$  für alle WBT  $p$  und (alle  $x \in V_{\mathbb{B},p}$  und alle WBT  $t$ ) sowie (alle  $x \in V_{\mathbb{N},p}$  und alle WAA  $t$ )

**Beispiel 2:**  $\{(1 + 1) \leq (1 + 1) * y\}$   $x \leftarrow (1 + 1)$   $\{(x \leq x * y)\}$  aus Bsp. 1.

## Der Hoaresche Kalkül: Ableitungsregel (F)

Die *Folgerungsregel* (F):

$$\frac{p \rightsquigarrow q, \{q\} S \{r\}, r \rightsquigarrow s}{\{p\} S \{s\}}$$

Um nicht unnötigen weiteren notationellen Ballast anzuhäufen, dürfen wir beim Nachweis der “Implikation”  $\rightsquigarrow$  auch die üblichen Rechengesetze für natürliche Zahlen und Boolesche Ausdrücke verwenden.

Streng formal müsste das alles in dem Kalkül (als mechanischer “Rechenvorschrift”) aufgelistet sein...

**Beispiel 2:**  $\{1 \leq y\} x \leftarrow (1 + 1) \{(x \leq x * y)\}$  aus Bsp. 2. mit (F), denn es gilt  $(1 \leq y) \rightsquigarrow (2 \leq 2 * y) \rightsquigarrow ((1 + 1) \leq (1 + 1) * y)$ .

## Der Hoaresche Kalkül: Von hinten nach vorne

Eine mögliche “Instantiierung” vom Zuweisungsschema (Z) ist:

$$\{(x + 1) > 4\} x \leftarrow x + 1 \{x > 4\}.$$

Die Zahlvariable  $x$ , die in  $x > 4$  vorkommt, wird durch den WAA  $(x + 1)$  ersetzt.

In unserer Schreibweise:  $(x > 4)_x^{x+1} = (x + 1) > 4$ .

Wegen  $(x > 3) \rightsquigarrow ((x + 1) > 4)$  folgt mit (F) aus (Z):

$$\{x > 3\} x \leftarrow x + 1 \{x > 4\}.$$

Beachte: Die Vorbedingung ergibt sich (mechanisch) aus der Nachbedingung, also “von hinten nach vorne”.

Dies entspricht der üblichen “Konstruktionsrichtung” dieses Kalküls.

Das ist aber sehr praktisch: Die Nachbedingung gibt ja (meist) an, “was das Programm tun soll”.

Nun “rechnet man aus”, ob es dies (auch stets) macht.

## Der Hoaresche Kalkül: Ableitungsregel (L)

Die *Linearregel* (L):

$$\frac{\{p\} S_1 \{q\}, \{q\} S_2 \{r\}}{\{p\} S_1; S_2 \{r\}}$$

Für welches  $p$  gilt:  $\{p\} a \leftarrow 0; b \leftarrow x \{a * y + b = x\}$  ?

Aus (Z):  $\{a * y + x = x\} b \leftarrow x \{a * y + b = x\}$ .

Für jede Zahl  $x$  (und alle  $a, y$ ) gilt:  $a * y = 0 \rightsquigarrow a * y + x = x$ .

Aus (F):  $\{a * y = 0\} b \leftarrow x \{a * y + b = x\}$ .

Für welches  $p$  gilt:  $\{p\} a \leftarrow 0 \{a * y = 0\}$  ?

Mit (L) und (Z) ist  $p = (0 * y = 0)$  eine mögliche Antwort.

Da dies eine offensichtliche zahlentheoretische Wahrheit ist, ist mit (F) auch  $p =$  **true** möglich. Also gilt die Nachbedingung  $\{a * y + b = x\}$  immer!

## Der Hoaresche Kalkül: Ableitungsregel (I)

Die *Bedingungsregel* (I):

$$\frac{\{p \wedge e\} S_1 \{q\}, \{p \wedge \neg e\} S_2 \{q\}}{\{p\} \mathbf{if\ } e \mathbf{\ then\ } S_1 \mathbf{\ else\ } S_2 \mathbf{\ fi\ } \{q\}}$$

Ein Trivialbeispiel bietet die Minimumberechnung.

Mit der Kenntnis der Beziehung

$$\text{ggt}(a, b) = \begin{cases} a, & a = b \\ \text{ggt}(a - b, b), & a > b \\ \text{ggt}(a, b - a), & a < b \end{cases}$$

lässt sich zeigen:

$$\begin{aligned} & \{a \neq b \wedge A = \text{ggt}(a, b)\} \\ & \mathbf{if\ } a > b \mathbf{\ then\ } a \leftarrow a - b \mathbf{\ else\ } b \leftarrow b - a \mathbf{\ fi} \\ & \{A = \text{ggt}(a, b)\}. \\ & \text{Insbesondere gilt mit (Z) und (F):} \\ & \{a \neq b \wedge A = \text{ggt}(a, b) \wedge \neg(a > b)\} \\ & \rightsquigarrow \{a < b \wedge A = \text{ggt}(a, b)\} \\ & b \leftarrow b - a \\ & \{A = \text{ggt}(a, b)\}. \end{aligned}$$

## Der Hoaresche Kalkül: Ableitungsregel (W)

Die *Schleifenregel* (W):

$$\frac{\{p \wedge e\} S \{p\}}{\{p\} \mathbf{while} \ e \ \mathbf{do} \ S \ \mathbf{od} \ \{p \wedge \neg e\}}$$

mit *Schleifeninvariante*  $p$ . Betrachte:

$$\{A = \text{ggt}(x, y) \wedge a = x \wedge b = y\} \rightsquigarrow \{A = \text{ggt}(a, b)\}$$

**while**  $a \neq b$  **do**

$$\{a \neq b \wedge A = \text{ggt}(a, b)\}$$

**if**  $a > b$  **then**  $a \leftarrow a - b$  **else**  $b \leftarrow b - a$  **fi**

$$\{A = \text{ggt}(a, b)\}.$$

**od**

$$\{A = \text{ggt}(a, b) \wedge a = b\} \rightsquigarrow \{a = A = \text{ggt}(x, y)\}$$

## Der Hoaresche Kalkül: Beweisskizzen mit Ergänzungshinweisen

Hinweis: Partielle Korrektheit! (WO?)

$$\{\mathbf{true}\} \rightsquigarrow \{0 * y = 0\}$$

$a \leftarrow 0;$  (Z)

$$\{a * y = 0\} \rightsquigarrow \{a * y + x = x\} \textit{Zusicherungsnotation}$$

$b \leftarrow x;$  (Z)

$\{a * y + b = x\}$  *Schleifeninvariante wird durch zwei Zuweisungen eingerichtet*

**while**  $b \geq y$  **do**

$$\{a * y + b = x \wedge b \geq y\} \rightsquigarrow \{(a + 1) * y + (b - y) = x \wedge b - y \geq 0\}$$

$b \leftarrow b - y;$  (Z)

$$\{(a + 1) * y + b = x (\wedge b \geq 0)\}$$

$a \leftarrow a + 1;$  (Z)

$\{a * y + b = x\}$  *Schleifeninvariante wird durch Schleifenrumpf erhalten*

**od** (W)

$$\{a * y + b = x \wedge \neg(b \geq y)\} \rightsquigarrow \{a * y + b = x \wedge b < y\}$$

## Der Hoaresche Kalkül: Ein kleiner Tippfehler! mit Ergänzungshinweisen

Hinweis: Partielle Korrektheit!!!

$$\{\mathbf{true}\} \rightsquigarrow \{a * 0 = 0\}$$

$$y \leftarrow 0; (\mathbf{Z})$$

$$\{a * y = 0\} \rightsquigarrow \{a * y + x = x\}$$

$$b \leftarrow x; (\mathbf{Z})$$

$$\{a * y + b = x\}$$

**while**  $b \geq y$  **do**

$$\{a * y + b = x \wedge b \geq y\} \rightsquigarrow \{(a + 1) * y + (b - y) = x \wedge b - y \geq 0\}$$

$$b \leftarrow b - y; (\mathbf{Z})$$

$$\{(a + 1) * y + b = x (\wedge b \geq 0)\}$$

$$a \leftarrow a + 1; (\mathbf{Z})$$

$$\{a * y + b = x\}$$

**od** ( $\mathbf{W}$ )

$$\{a * y + b = x \wedge \neg(b \geq y)\} \rightsquigarrow \{a * y + b = x \wedge b < y\}$$

## Sortieren durch Einfügen:

Rudimente des Hoare-Kalküls in Programmkomentaren



Schön geordnet ? Aber wie ?



## Sortieren durch Einfügen: Wie geht es beim Skat ?



Verfahren / Vorgehen:

Die neu hinzukommende Karte wird gemäß der “Skatordnung” eingefügt.

Die Herz 10 wird daher zwischen der Pik-Karte und der bisherig höchsten Herz-Karte gesteckt.

## Ein Beispiel mit Zahlen

Die folgende Tabelle zeigt die Sortierschritte zum Sortieren der Folge

5 7 0 3 4 2 6 1

5	7	0	3	4	2	6	1	(0)
5	7	0	3	4	2	6	1	(0)
0	5	7	3	4	2	6	1	(2)
0	3	5	7	4	2	6	1	(2)
0	3	4	5	7	2	6	1	(2)
0	2	3	4	5	7	6	1	(4)
0	2	3	4	5	6	7	1	(1)
0	1	2	3	4	5	6	7	(6)

Auf der linken Seite rot dargestellt befindet sich jeweils der bereits sortierte Teil der Folge.

Ganz rechts steht in Klammern die Anzahl der Positionen, um die das eingefügte Element nach links gewandert ist.

Einzelheiten, einschließlich einer hübschen Simulation, finden Sie unter: <http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/insert/insertion.htm>

## Ein wenig Programmcode

```
insertionsort ()
{
    int i, j, t;
    for (i=1; i<n; i++)
    {
        j=i;
        t=a[j];
        while (j>0 && a[j-1]>t)
        {
            a[j]=a[j-1];
            j--;
        }
        a[j]=t;
    }
}
```

---

**Algorithm 1** Sortieren durch Einfügen: insertionsort

**Input(s):** an array  $A : \mathbb{Z}[1..n]$

**Output(s):**  $A$  becomes a sorted array

**for**  $i \leftarrow 2$  **to**  $n$  **do**

$t \leftarrow A[i]$  {einzufügendes Element}

**for**  $j \leftarrow i$  **downto** 2 **do**

**if**  $A[j - 1] \geq t$  **then**

**exit for**

**else**

$A[j] \leftarrow A[j - 1]$  {Verschiebe}

$A[j] \leftarrow t$  {Einfügestelle gefunden}

---

---

**Algorithm 1** Sortieren durch Einfügen: Korrektheitsbetrachtungen

---

**Input(s):** an array  $A : \mathbb{Z}[1..n]$

**Output(s):**  $A$  becomes a sorted array

**for**  $i \leftarrow 2$  **to**  $n$  **do**

{ $A[1]$  bis  $A[i - 1]$  ist bereits aufsteigend sortiert.}

$t \leftarrow A[i]$  {einzufügendes Element}

**for**  $j \leftarrow i$  **downto**  $2$  **do**

{ $A[1]$  bis  $A[j - 1]$  und  $A[j + 1]$  bis  $A[i]$  ist aufsteigend sortiert.}

{ $\forall k : j + 1 \leq k \leq i \implies t \leq A[k]$ .}

{ $A[1..j - 1]$  ist unverändert gegenüber Schleifenbeginn;}

{ $A[j + 1..i]$  entspricht  $A[j..i - 1]$  vor Schleifenbeginn.}

**if**  $A[j - 1] \geq t$  **then**

**exit for**

**else**

$A[j] \leftarrow A[j - 1]$  {Verschiebe}

$A[j] \leftarrow t$  {Einfügestelle gefunden, d.h.:  $A[j - 1] \leq t \leq A[j + 1]$ .}

**Hoare-Kalkül**: eine Reflexion

Offenbar ist der Hoare-Kalkül  
anwendbar in praktisch relevanten Programmiersituationen,  
durchaus empfehlenswert bei sicherheitsrelevanten Anwendungen,  
obwohl selbst dort oft sorgsam entworfene Tests bevorzugt werden.

**ABER**: Ist der Hoare-Kalkül korrekt??

In der vorigen VL haben wir Beweise stets über die Semantik Hoarescher Ausdrücke geführt.

Das sah immer recht länglich und umständlich aus.

Ist dies überflüssig geworden? Und wenn ja, warum?

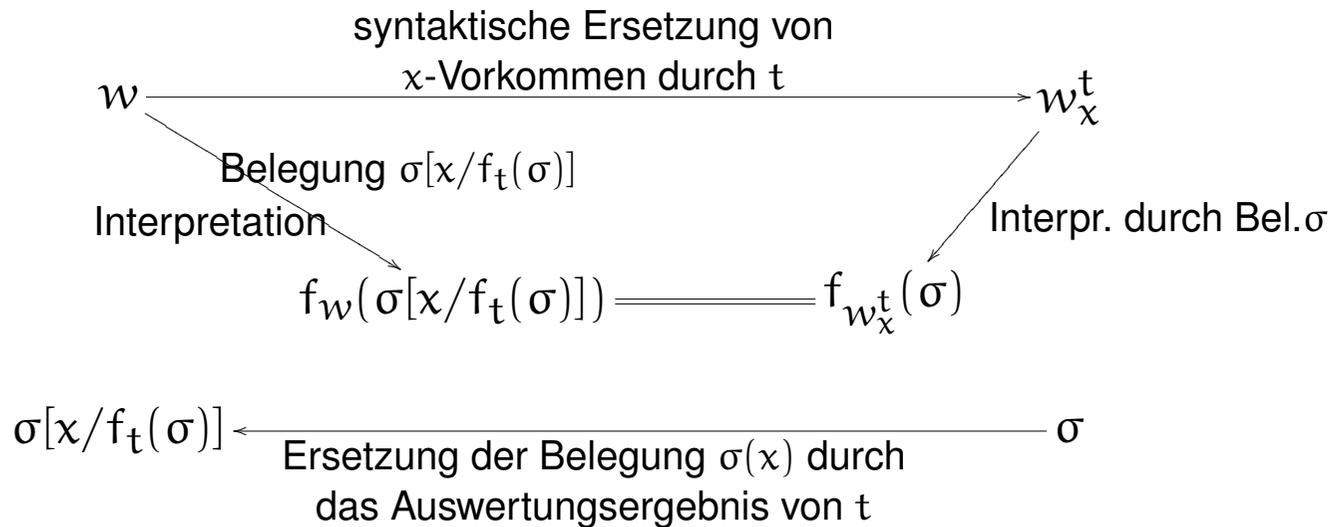
## Das Substitutionstheorem

Ist  $w$  ein WAA und  $x$  eine in  $w$  vorkommende Variable und  $t$  ein weiterer WAA, so gilt für alle Belegungen  $\sigma$ :

$$f_{w_x^t}(\sigma) = f_w(\sigma[x/f_t(\sigma)]).$$

Entsprechende Aussagen gelten für WBTs.

Graphische Darstellung als *Diagramm*:



## **Der Hoare-Kalkül ist korrekt.**

Das heißt: Können wir  $\{p\}S\{q\}$  im Hoare-Kalkül beweisen, so gilt für alle Belegungen  $\sigma$ , die die Vorbedingung  $p$  erfüllen, dass die Belegung  $M \langle S \rangle (\sigma)$  die Nachbedingung  $q$  erfüllt (sofern  $S$  terminiert).

Was ist zu zeigen:

- (1) Die Axiomenschemata sind korrekt.
- (2) Die Schlussregeln sind korrekt.

Daraus folgt dann die Behauptung durch einfache Induktion über die Länge einer Ableitung im Hoare-Kalkül.

## Der Hoare-Kalkül ist korrekt. Beweisteile (1)

Aus  $M \langle \mathbf{skip} \rangle (\sigma) = \sigma$  folgt sofort das Skip-Axiomenschema (S).  
 $\{p\} \mathbf{skip} \{p\}$  für alle WBT  $p$ .

Das Substitutionstheorem liefert unmittelbar die Korrektheit für das Zuweisungs-Axiomenschema (Z).

$\{p_x^t\} x \leftarrow t \{p\}$  für alle WBT  $p$  und (alle  $x \in V_{\mathbb{B},p}$  und alle WBT  $t$ ) sowie (alle  $x \in V_{\mathbb{N},p}$  und alle WAA  $t$ )

Beachte beim Nachweis der Korrektheit der Schlussregeln, dass wir immer von der Korrektheit der Prämissen ausgehen; man könnte dies auch als strukturelle Induktion begreifen.

**Der Hoare-Kalkül ist korrekt.** Beweisteile (2)(L)

Die Linearregel (L) ist korrekt: Betrachte

$$\frac{\{p\} S_1 \{q\}, \{q\} S_2 \{r\}}{\{p\} S_1; S_2 \{r\}}$$

Sei  $\sigma$  eine Belegung, die  $p$  erfüllt.

Wenn  $S_1; S_2$  terminiert,

so gibt es Belegungen  $\rho = M \langle S_1 \rangle (\sigma)$  und  $\tau = M \langle S_2 \rangle (\rho)$ .

Aus der angenommenen Korrektheit der Prämissen der Schlussregel folgt:

$\rho$  erfüllt  $q$  und daher erfüllt  $\tau$  die Nachbedingung  $r$ .

Da  $\tau = M \langle S_1; S_2 \rangle (\sigma) = M \langle S_2 \rangle (\rho)$ , folgt die Behauptung.

Beweisteil (2)(F) ist “trivial richtig.”

**Der Hoare-Kalkül ist korrekt.** Beweisteile (2)(I)

Betrachte die Bedingungsregel (I):

$$\frac{\{p \wedge e\} S_1 \{q\}, \{p \wedge \neg e\} S_2 \{q\}}{\{p\} \mathbf{if\ } e \mathbf{\ then\ } S_1 \mathbf{\ else\ } S_2 \mathbf{\ fi\ } \{q\}}$$

Ist  $\sigma$  eine Belegung, die  $p \wedge e$  erfüllt, so bedeutet die Korrektheit der Prämisse, dass  $M \langle S_1 \rangle (\sigma)$  dann  $q$  erfüllt.

Da dann aber insbesondere  $f_e(\sigma) = 1$  gilt, folgt  $M \langle \mathbf{if\ } e \mathbf{\ then\ } S_1 \mathbf{\ else\ } S_2 \mathbf{\ fi} \rangle (\sigma) = M \langle S_1 \rangle (\sigma)$ , d.h.  $q$  ist erfüllt nach Ausführung der Verzweigung.

Der Fall einer Belegung  $\sigma$ , welche  $p \wedge \neg p$  erfüllt, ist analog zu behandeln.

**Der Hoare-Kalkül ist korrekt.** Beweisteile (2)(W)

Betrachte die Schleifenregel (W):

$$\frac{\{p \wedge e\} S \{p\}}{\{p\} \mathbf{while} \ e \ \mathbf{do} \ S \ \mathbf{od} \ \{p \wedge \neg e\}}$$

Setze  $f = M \langle \mathbf{while} \ e \ \mathbf{do} \ S \ \mathbf{od} \rangle$  und  $g = M \langle S \rangle$ .

Nach Lemma aus VL 3 gilt im Terminierungsfall:

$f(\sigma) = g^k(\sigma)$  für ein  $k \geq 0$  mit  $f_e(g^k(\sigma)) = 0$ ;  $f_e(g^i(\sigma)) = 1$  für alle  $0 \leq i < k$ .

Daraus ergibt sich für alle  $i = 0, \dots, k$  durch Induktion  $f_p(g^i(\sigma)) = 1$ .

Speziell für  $i = k$  folgt  $f_p(\tau) = 1$  für  $\tau = f(\sigma)$ .

Wegen  $f_e(\tau) = 0$  folgt die Behauptung  $f_{p \wedge \neg e}(\tau) = 1$ .

## **Der Hoare-Kalkül ist vollständig.**

Diese Aussage wollen wir nicht beweisen, aber etwas verstehen:

Es gibt m.a.W. keine Folgerung, die wir auf der semantischen Ebene durchführen könnten, aber nicht innerhalb des Hoare-Kalküls durchzuführen wäre.

Streng genommen beruht das auch auf der Vollständigkeit des zugrunde gelegten logischen Kalküls.

Da wir diesen nicht streng formal betrachtet haben, müsste ein Beweis der Vollständigkeitsaussage notwendig deutliche Lücken aufweisen.

Das Auffüllen dieser Lücken würde 3-4 Vorlesungen beanspruchen und entfällt daher hier.