Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Dr. Olaf Müller

WS 1998/99
Lösungsvorschläge zu Blatt 7
23. 12. 1998

Übungen zur Semantik von Programmiersprachen

Aufgabe 1 (Lösungsvorschlag)

Seien $x > 0$ und $y > 0$.

Eine Schleifeninvariante, mit der sich die Behauptung beweisen läßt, ist

$$I: \qquad X \cdot U + Y \cdot V = 2 \cdot x \cdot y \wedge gcd(X,Y) = gcd(x,y) \wedge X > 0 \wedge Y > 0 \, .$$

Beim Beweis, daß I tatsächlich eine Invariante ist, konzentrieren wir uns auf die erste Teilformel der Konjunktion, da wir den Beweis für den Rest der Konjunktion in Blatt 6/Aufgabe 2 bereits geführt haben.

Wegen

$$(X - Y) \cdot U + Y \cdot (V + U) = X \cdot U + Y \cdot V$$

folgt für den *then*-Zweig mit den Regeln für Zuweisung, sequentielle Komposition und Verstärkung der Vorbedingung:

$$\begin{array}{c} \{X > Y \wedge I\} \\ X := X - Y \; ; \; V := V + U \\ \{I\} \end{array}$$

Wegen

$$X \cdot (U + V) + (Y - X) \cdot V = X \cdot U + Y \cdot V$$

folgt für den *else*-Zweig mit den Regeln für Zuweisung, sequentielle Komposition und Verstärkung der Vorbedingung:

$$\begin{array}{c} \{X < Y \wedge I\} \\ Y := Y - X \; ; \; U := U + V \\ \{I\} \end{array}$$

Mit der Regel für bedingte Anweisungen erhalten wir:

$$\begin{array}{c} \{\neg(X = Y) \wedge I\} \\ \textbf{if } X > Y \textbf{ then } (X := X - Y \; ; \; V := V + U) \\ \textbf{else } (Y := Y - X \; ; \; U := U + V) \\ \{I\} \end{array}$$

Mit der Schleifenregel folgt:

$$\begin{array}{l} \qquad\qquad\quad \{I\} \\ \textbf{while } \neg(X = Y) \textbf{ do} \\ (\textbf{if } X > Y \textbf{ then } (X := X - Y \; ; \; V := V + U) \\ \qquad\quad \textbf{else } (Y := Y - X \; ; \; U := U + V)) \\ \qquad\quad \{I \wedge X = Y\} \end{array}$$

Mit den Regeln für die Zuweisung und die sequentielle Komposition folgt, daß nach der Initialisierung die Schleifeninvariante gilt:

$$\begin{array}{c} \{true\} \\ X := x \; ; \; Y := y \; ; \; U := y \; ; \; V := x \\ \{I\} \end{array}$$

Nun betrachten wir das Programmstück hinter der Schleife und zeigen, daß $I \wedge X = Y$ die Formel $(U+V)/2 = lcm(x,y)$ impliziert:

$$\begin{array}{ll}
 & I \wedge X = Y \\
\Rightarrow & \{ \text{ Leibniz-Regel, d.h. "Substitution von Gleichem für Gleiches";} \\
 & \quad \text{Eigenschaft von } gcd\} \\
 & X \cdot U + X \cdot V = 2 \cdot x \cdot y \wedge gcd(x,y) = X \\
\Rightarrow & \{ \text{ Arithmetik und Leibniz-Regel}\} \\
 & gcd(x,y) \cdot (U+V) = 2 \cdot x \cdot y \\
= & \{ \; gcd(x,y) > 0\} \\
 & \frac{U+V}{2} = \frac{x \cdot y}{gcd(x,y)} \\
= & \{ \text{ Hinweis aus der Angabe}\} \\
 & \frac{U+V}{2} = lcm(x,y)
\end{array}$$

Mit den Regeln für Zuweisung, sequentielle Komposition und Verstärkung der Vorbedingung erhalten wir:

$$\begin{array}{c} \{I \wedge X = Y\} \\ A := (X+Y)/2 \; ; \; B := (U+V)/2 \\ \{B = lcm(x,y)\} \end{array}$$

Damit ist für das gesamte gegebene Programm P gezeigt:

$$\begin{array}{c} \{true\} \\ P \\ \{B = lcm(x,y)\} \end{array}$$

Aufgabe 2 (Lösungsvorschlag)

(a) *Zu zeigen:* $\forall ac \in \mathbf{Acom}, B, C \in \mathbf{Assn} : (B \Rightarrow C) \Rightarrow (wp(ac,B) \Rightarrow wp(ac,C))$

Beweis: Wir zeigen

$$\forall B, C \in \mathbf{Assn} : (B \Rightarrow C) \Rightarrow (wp(ac,B) \Rightarrow wp(ac,C))$$

mittels struktureller Induktion über ac.

$\underline{ac = \mathbf{skip}}$: Gelte $B \Rightarrow C$.

$$\begin{array}{ll}
 & wp(\mathbf{skip}, B) \\
= & \{ \text{ Def. von } wp\} \\
 & B \\
\Rightarrow & \{ \text{ Voraussetzung}\} \\
 & C \\
= & \{ \text{ Def. von } wp\} \\
 & wp(\mathbf{skip}, C)
\end{array}$$

$\underline{ac = (X := a)\text{:}}$ Gelte $B \Rightarrow C$.

$$
\begin{array}{cl}
 & wp(X := a, B) \\
= & \{ \text{ Def. von } wp\} \\
 & B[a/X] \\
\Rightarrow & \{ \ B \Rightarrow C, \text{ und Substitution ist im 1. Argument monoton bzgl. } \Rightarrow\} \\
 & C[a/X] \\
= & \{ \text{ Def. von } wp\} \\
 & wp(X := a, C)
\end{array}
$$

$\underline{ac = (ac_0; ac_1)\text{:}}$ Gelte $B \Rightarrow C$.

$$
\begin{array}{cl}
 & wp(ac_0; ac_1, B) \\
= & \{ \text{ Def. von } wp\} \\
 & wp(ac_0, wp(ac_1, B)) \\
\Rightarrow & \{ \text{ Ind.vor. für } ac_1 \text{ liefert } wp(ac_1, B) \Rightarrow wp(ac_1, C); \\
 & \quad \text{Ind.vor. für } ac_0 \text{ mit } wp(ac_1, B) \text{ für } B \text{ und } wp(ac_1, C) \text{ für } C\} \\
 & wp(ac_0, wp(ac_1, C)) \\
= & \{ \text{ Def. von } wp\} \\
 & wp(ac_0; ac_1, C)
\end{array}
$$

$\underline{ac = \textbf{if } b \textbf{ then } ac_0 \textbf{ else } ac_1\text{:}}$ Gelte $B \Rightarrow C$.

$$
\begin{array}{cl}
 & wp(\textbf{if } b \textbf{ then } ac_0 \textbf{ else } ac_1, B) \\
= & \{ \text{ Def. von } wp\} \\
 & (b \wedge wp(ac_0, B)) \vee (\neg b \wedge wp(ac_1, B)) \\
\Rightarrow & \{ \text{ Ind.vor. für } ac_0 \text{ und } ac_1\} \\
 & (b \wedge wp(ac_0, C)) \vee (\neg b \wedge wp(ac_1, C)) \\
= & \{ \text{ Def. von } wp\} \\
 & wp(\textbf{if } b \textbf{ then } ac_0 \textbf{ else } ac_1, C)
\end{array}
$$

$\underline{ac = \textbf{while } b \textbf{ do } \{I\}\ ac_0\text{:}}$ Gelte $B \Rightarrow C$.

$$
\begin{array}{cl}
 & wp(\textbf{while } b \textbf{ do } \{I\}\ ac_0, B) \\
= & \{ \text{ Def. von } wp\} \\
 & I \\
= & \{ \text{ Def. von } wp\} \\
 & wp(\textbf{while } b \textbf{ do } \{I\}\ ac_0, C)
\end{array}
$$

$\square$

(b) *Zu zeigen:* $\forall ac \in \textbf{Acom}, B, C \in \textbf{Assn} : (B \Rightarrow C) \Rightarrow (vc(ac, B) \Rightarrow vc(ac, C))$
Beweis: Wir zeigen

$$\forall B, C \in \textbf{Assn} : (B \Rightarrow C) \Rightarrow (vc(ac, B) \Rightarrow vc(ac, C))$$

mittels struktureller Induktion über ac.

$ac = \textbf{skip}$ oder $ac = (X := a)$:

$$\begin{array}{cl} & vc(ac, B) \\ = & \{\ \text{Def. von } vc\} \\ & true \\ = & \{\ \text{Def. von } vc\} \\ & vc(ac, C) \end{array}$$

$ac = (ac_0; ac_1)$: Gelte $B \Rightarrow C$.

$$\begin{array}{cl} & vc(ac_0; ac_1, B) \\ = & \{\ \text{Def. von } vc\} \\ & vc(ac_0, wp(ac_1, B)) \wedge vc(ac_1, B) \\ \Rightarrow & \{\ \text{Ind.vor. für } ac_1\} \\ & vc(ac_0, wp(ac_1, B)) \wedge vc(ac_1, C) \\ \Rightarrow & \{\ \text{Ind.vor. für } ac_0 \text{ mit } wp(ac_1, B) \text{ für } B \text{ und } wp(ac_1, C) \text{ für } C; \\ & \quad \text{nach (a) gilt } wp(ac_1, B) \Rightarrow wp(ac_1, C)\} \\ & vc(ac_0, wp(ac_1, C)) \wedge vc(ac_1, C) \\ = & \{\ \text{Def. von } vc\} \\ & vc(ac_0; ac_1, C) \end{array}$$

$ac = \textbf{if } b \textbf{ then } ac_0 \textbf{ else } ac_1$: Gelte $B \Rightarrow C$.

$$\begin{array}{cl} & vc(\textbf{if } b \textbf{ then } ac_0 \textbf{ else } ac_1, B) \\ = & \{\ \text{Def. von } vc\} \\ & vc(ac_0, B) \wedge vc(ac_1, B) \\ \Rightarrow & \{\ \text{Ind.vor. für } ac_0 \text{ und } ac_1\} \\ & vc(ac_0, C) \wedge vc(ac_1, C) \\ = & \{\ \text{Def. von } vc\} \\ & vc(\textbf{if } b \textbf{ then } ac_0 \textbf{ else } ac_1, C) \end{array}$$

$ac = \textbf{while } b \textbf{ do } \{I\}\ ac_0$: Gelte $B \Rightarrow C$.

$$\begin{array}{cl} & vc(\textbf{while } b \textbf{ do } \{I\}\ ac_0, B) \\ = & \{\ \text{Def. von } vc\} \\ & (I \wedge \neg b \Rightarrow B) \wedge (I \wedge b \Rightarrow wp(ac_0, I)) \wedge vc(ac_0, I) \\ \Rightarrow & \{\ B \Rightarrow C; \Rightarrow \text{ist transitiv}\} \\ & (I \wedge \neg b \Rightarrow C) \wedge (I \wedge b \Rightarrow wp(ac_0, I)) \wedge vc(ac_0, I) \\ = & \{\ \text{Def. von } vc\} \\ & vc(\textbf{while } b \textbf{ do } \{I\}\ ac_0, C) \end{array}$$

□

Aufgabe 3 (Lösungsvorschlag)

Wir erweitern den Hoare-Kalkül aus der Vorlesung um folgende Regel für das *repeat*-Konstrukt:

$$\frac{\{A\}\ c\ \{B\} \qquad \{B \wedge \neg b\}\ c\ \{B\}}{\{A\}\ \textbf{repeat}\ c\ \textbf{until}\ b\ \{B \wedge b\}}$$

Wir zeigen, daß die Regel korrekt ist, d.h. daß gilt (vgl. Bemerkungen unten):

$$(\models \{A\}\ c\ \{B\}) \wedge (\models \{B \wedge \neg b\}\ c\ \{B\}) \Rightarrow (\models \{A\}\ \textbf{repeat}\ c\ \textbf{until}\ b\ \{B \wedge b\})$$

Gelten

$$\models \{A\}\ c\ \{B\} \text{ und} \tag{0}$$

$$\models \{B \wedge \neg b\}\ c\ \{B\}\ . \tag{1}$$

Wir rechnen:

$$
\begin{array}{cl}
 & \models \{A\}\ \textbf{repeat}\ c\ \textbf{until}\ b\ \{B \wedge b\} \\
= & \{\ \text{Definition von} \models\} \\
 & \forall \text{Interpretationen } I, \sigma \in \Sigma_\perp\ .\ (\sigma \models^I \{A\}\ \textbf{repeat}\ c\ \textbf{until}\ b\ \{B \wedge b\}) \\
= & \{\ \text{Definition von} \models^I\} \\
 & \forall I, \sigma \in \Sigma_\perp\ .\ (\sigma \models^I A) \Rightarrow (\mathcal{C}[\![\textbf{repeat}\ c\ \textbf{until}\ b]\!]\sigma \models^I B \wedge b) \\
= & \{\ \text{Definition von } \mathcal{C} \text{ gemäß Blatt 4/Aufgabe 3}\} \\
 & \forall I, \sigma \in \Sigma_\perp\ .\ (\sigma \models^I A) \Rightarrow ((\bigsqcup_{n\in\omega} \Gamma^n(\perp))\sigma \models^I B \wedge b) \\
 & \text{mit } \Gamma : (\Sigma_\perp \nrightarrow \Sigma_\perp) \to (\Sigma_\perp \nrightarrow \Sigma_\perp) \\
 & \Gamma = \lambda\varphi.\lambda\rho. \begin{cases} \varphi(\mathcal{C}[\![c]\!]\rho), & \text{falls } \mathcal{B}[\![b]\!](\mathcal{C}[\![c]\!]\rho) = \textbf{false} \\ \mathcal{C}[\![c]\!]\rho, & \text{falls } \mathcal{B}[\![b]\!](\mathcal{C}[\![c]\!]\rho) = \textbf{true} \\ \perp, & \text{falls } \mathcal{C}[\![c]\!]\rho = \perp \end{cases} \\
 & \text{und } \perp \text{ auch für die überall undefinierte Funktion} \\
= & \{\ \text{die kleinste obere Schranke von Funktionen: punktweise}\} \\
 & \forall I, \sigma \in \Sigma_\perp\ .\ (\sigma \models^I A) \Rightarrow (\bigsqcup_{n\in\omega}(\Gamma^n(\perp)(\sigma)) \models^I B \wedge b) \\
= & \{\ \Sigma_\perp \text{ flach geordnet; } \perp \models^I B \wedge b\} \\
 & \forall I, \sigma \in \Sigma_\perp\ .\ (\sigma \models^I A) \Rightarrow (\forall n \in \omega.(\Gamma^n(\perp)(\sigma) \models^I B \wedge b)) \\
= & \{\ \text{Prädikatenkalkül}\} \\
 & \forall n \in \omega.\forall I, \sigma \in \Sigma_\perp\ .\ (\sigma \models^I A) \Rightarrow (\Gamma^n(\perp)(\sigma) \models^I B \wedge b) \\
\Leftarrow & \{\ \text{nach (0) gilt für alle } I,\ \sigma\text{: } (\sigma \models^I A) \Rightarrow (\mathcal{C}[\![c]\!]\sigma \models^I B)\} \\
 & \forall n \in \omega.\forall I, \sigma \in \Sigma_\perp\ .\ (\mathcal{C}[\![c]\!]\sigma \models^I B) \Rightarrow (\Gamma^n(\perp)(\sigma) \models^I B \wedge b)
\end{array}
$$

Die letzte Formel beweisen wir mittels Induktion über n:

$n = 0$:

Für alle I und σ gilt:

$$
\begin{array}{cl}
 & \Gamma^0(\perp)(\sigma) \models^I B \wedge b \\
= & \{\ \Gamma^0 = id\} \\
 & \perp(\sigma) \models^I B \wedge b \\
= & \{\ \perp \text{ ist die überall undefinierte Funktion}\} \\
 & \perp \models^I B \wedge b \\
= & \{\ \text{Def. von} \models^I\} \\
 & \mathit{true}
\end{array}
$$

$n \leadsto n+1$:

Induktionsvoraussetzung:

$$\forall I, \sigma \in \Sigma_\perp\ .\ (\mathcal{C}[\![c]\!]\sigma \models^I B) \Rightarrow (\Gamma^n(\perp)(\sigma) \models^I B \wedge b) \tag{2}$$

Seien I und σ beliebig mit

$$\mathcal{C}[\![c]\!]\sigma \models^I B\ . \tag{3}$$

Zu zeigen ist: $\Gamma^{n+1}(\bot)(\sigma) \models^I B \wedge b$

Es gilt:

$$
\begin{array}{ll}
 & \Gamma^{n+1}(\bot)(\sigma) \\
= & \{ \quad \Gamma^{n+1} = \Gamma \circ \Gamma^n \} \\
 & \Gamma(\Gamma^n(\bot))(\sigma) \\
= & \{ \quad \text{Definition von } \Gamma \text{, siehe oben} \} \\
 & \begin{cases} (\Gamma^n(\bot))(\mathcal{C}[\![c]\!]\sigma), & \text{falls } \mathcal{B}[\![b]\!](\mathcal{C}[\![c]\!]\sigma) = \mathbf{false} \\ \mathcal{C}[\![c]\!]\sigma, & \text{falls } \mathcal{B}[\![b]\!](\mathcal{C}[\![c]\!]\sigma) = \mathbf{true} \\ \bot, & \text{falls } \mathcal{C}[\![c]\!]\sigma = \bot \end{cases}
\end{array}
$$

Zum 3. Fall: $\bot \models^I B \wedge b$ gilt nach Definition von $\models^I$.

Im 2. Fall ist $\mathcal{C}[\![c]\!]\sigma \models^I B \wedge b$ zu zeigen. Nach (3) gilt $\mathcal{C}[\![c]\!]\sigma \models^I B$. Aus der Bedingung des 2. Falls folgt mit einem Satz der Vorlesung $\mathcal{C}[\![c]\!]\sigma \models^I b$. Daraus folgt nach einem Satz der Vorlesung $\mathcal{C}[\![c]\!]\sigma \models^I B \wedge b$.

Im 1. Fall ist $(\Gamma^n(\bot))(\mathcal{C}[\![c]\!]\sigma) \models^I B \wedge b$ zu zeigen. Nach Induktionsvor. (2) genügt es, $\mathcal{C}[\![c]\!](\mathcal{C}[\![c]\!]\sigma) \models^I B$ zu zeigen. Nach (3) gilt $\mathcal{C}[\![c]\!]\sigma \models^I B$. Aus der Bedingung des 1. Falls folgt mit einem Satz der Vorlesung $\mathcal{C}[\![c]\!]\sigma \models^I \neg b$. Daraus folgt nach einem Satz der Vorlesung $\mathcal{C}[\![c]\!]\sigma \models^I B \wedge \neg b$. Mit (1) folgt $\mathcal{C}[\![c]\!](\mathcal{C}[\![c]\!]\sigma) \models^I B$.

Da die Fallunterscheidung vollständig ist, ist die Behauptung gezeigt.

Bemerkungen:

(a) (*Korrektheit von Regeln*)

Eine Regel

$$\frac{P_1 \ \ldots \ P_n}{Q}$$

(mit partiellen Korrektheitsformeln $P_1, \ldots, P_n, Q$) eines Hoare-Kalküls heißt "korrekt", wenn gilt:

$$(\models P_1) \wedge \ldots \wedge (\models P_n) \ \Rightarrow \ (\models Q)$$

Hat man alle Regeln eines Hoare-Kalküls als korrekt bewiesen, dann folgt die Korrektheit des Hoare-Kalküls mittels Regelinduktion: Zu zeigen ist für alle partiellen Korrektheitsformeln Q, daß gilt:

$$(\vdash Q) \ \Rightarrow \ (\models Q)$$

Ist die in einer Herleitung zuletzt angewandte Regel

$$\frac{P_1 \ \ldots \ P_n}{Q}$$

dann folgt nach Induktionsvoraussetzung der Regelinduktion

$$(\models P_1) \wedge \ldots \wedge (\models P_n) \ .$$

Zu zeigen bleibt

$$\models Q \ ,$$

was unmittelbar aus der Korrektheit der Regel folgt.

D.h. die Korrektheit der einzelnen Regeln impliziert die Korrektheit eines Hoare-Kalküls.

(b) (*Notation*)

Wir haben die logischen Junktoren $\Rightarrow$, $\wedge$ usw. sowohl in der Zusicherungssprache **Assn** (z.B. $B \wedge b$) als auch auf der Meta-Ebene unserer Beweise (z.B. $(\models \{A\}\ c\ \{B\}) \ \wedge\ (\models \{B \wedge \neg b\}\ c\ \{B\})$) verwendet.

(c) (*Umwandlung in totale Funktion*)

In Blatt 4/Aufgabe 3 wurde $\mathcal{C}[\![\textbf{repeat}\ c\ \textbf{until}\ b]\!]$ als partielle Funktion aus $\Sigma \rightharpoonup \Sigma$ angegeben. Hier haben wir $\mathcal{C}[\![\textbf{repeat}\ c\ \textbf{until}\ b]\!]$ als totale Funktion aus $\Sigma_\perp \to \Sigma_\perp$ aufgefaßt, wobei $\perp$ für "undefiniert" steht. Deshalb ergibt sich obige Form von Γ.

Aufgabe 4 (Lösungsvorschlag)

```
-- Generator fuer Verifikationsbedingungen

-- Typdeklarationen fuer Loc, Aexp, Bexp, Mvar, Aexpv, Assn, Acom

type  Loc   = String

data  Aexp  = Num Int | Var Loc | Aexp :+ Aexp | Aexp :- Aexp | Aexp :* Aexp

data  Bexp  = Truth Bool | Aexp :== Aexp | Aexp :<= Aexp | Non Bexp |
              Bexp :/\ Bexp | Bexp :\/ Bexp

data  Mvar  = String

data  Aexpv = Numv Int | Varv Loc | Mvarv Mvar | Aexpv :+: Aexpv |
              Aexpv :-: Aexpv | Aexpv :*: Aexpv

data  Assn  = Atruth Bool | Aexpv :==: Aexpv | Aexpv :<=: Aexpv |
              Assn :/\: Assn | Assn :\/: Assn | Anon Assn | Assn :=>: Assn |
              Forall Mvar Assn | Exists Mvar Assn

-- Operatoren duerfen kein ";" enthalten. Deshalb schreiben wir sequentielle
-- Komposition mit "."

data  Acom  = Skip | Loc := Aexp | Acom :. Acom | If Bexp Acom Acom |
              While Bexp Assn Acom


-- Inklusion von Aexp in Aexpv

mkaexpv:: Aexp -> Aexpv

mkaexpv (Num n) = Numv n
mkaexpv (Var x) = Varv x
mkaexpv (a0 :+ a1) = mkaexpv a0 :+: mkaexpv a1
mkaexpv (a0 :- a1) = mkaexpv a0 :-: mkaexpv a1
mkaexpv (a0 :* a1) = mkaexpv a0 :*: mkaexpv a1
```

```
-- Inklusion von Bexp in Assn

mkassn:: Bexp -> Assn

mkassn (Truth t) = Atruth t
mkassn (a0 :== a1) = mkaexpv a0 :==: mkaexpv a1
mkassn (a0 :<= a1) = mkaexpv a0 :<=: mkaexpv a1
mkassn (Non b) = Anon (mkassn b)
mkassn (b0 :/\ b1) = mkassn b0 :/\: mkassn b1
mkassn (b0 :\/ b1) = mkassn b0 :\/: mkassn b1


-- Substitution auf arithmetischen Ausdruecken

substaexpv:: (Aexpv, Aexpv, Loc) -> Aexpv

substaexpv (Numv n, a, x) = Numv n
substaexpv (Varv y, a, x) = if x == y then a else Varv y
substaexpv (Mvarv i, a, x) = Mvarv i
substaexpv (a0 :+: a1, a, x) = substaexpv (a0, a, x) :+: substaexpv (a1, a, x)
substaexpv (a0 :-: a1, a, x) = substaexpv (a0, a, x) :-: substaexpv (a1, a, x)
substaexpv (a0 :*: a1, a, x) = substaexpv (a0, a, x) :*: substaexpv (a1, a, x)


-- Substitution auf Zusicherungen

substassn:: (Assn, Aexpv, Loc) -> Assn

substassn (Atruth t, a, x) = Atruth t
substassn (a0 :==: a1, a, x) = substaexpv (a0, a, x) :==: substaexpv (a1, a, x)
substassn (a0 :<=: a1, a, x) = substaexpv (a0, a, x) :<=: substaexpv (a1, a, x)
substassn (as0 :/\: as1, a, x) = substassn (as0, a, x) :/\: substassn (as1, a, x)
substassn (as0 :\/: as1, a, x) = substassn (as0, a, x) :\/: substassn (as1, a, x)
substassn (Anon as, a, x) = Anon (substassn (as, a, x))
substassn (as0 :=>: as1, a, x) = substassn (as0, a, x) :=>: substassn (as1, a, x)
substassn (Forall i as, a, x) = Forall i (substassn (as, a, x))
substassn (Exists i as, a, x) = Exists i (substassn (as, a, x))


-- wp aus der Vorlesung

wp:: (Acom, Assn) -> Assn

wp (Skip, as) = as
wp (x := a, as) = substassn (as, mkaexpv a, x)
wp (ac0 :. ac1, as) = wp (ac0, wp (ac1, as))
wp (If b ac0 ac1, as) =
   (mkassn b :/\: wp (ac0, as)) :\/: (mkassn (Non b) :/\: wp (ac1, as))
wp (While b i ac, as) = i
```

```
-- vc aus der Vorlesung

vc:: (Acom, Assn) -> Assn

vc (Skip, as) = Atruth True
vc (x := a, as) = Atruth True
vc (ac0 :. ac1, as) = vc (ac0, wp (ac1, as)) :/\: vc (ac1, as)
vc (If b ac0 ac1, as) = vc (ac0, as) :/\: vc (ac1, as)
vc (While b i ac, as) =
   (((i :/\: Anon (mkassn b)) :=>: as) :/\: ((i :/\: mkassn b) :=>: wp (ac, i)))
   :/\: vc (ac, i)
```