Korrekte Software: Grundlagen und Methoden

Vorlesung 11 vom 18.06.19

Spezifikation von Funktionen

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2019

Fahrplan

- Einführung
- Operationale Semantik
- Denotationale Semantik
- Äquivalenz der Operationalen und Denotationalen Semantik
- Der Floyd-Hoare-Kalkül
- Invarianten und die Korrektheit des Floyd-Hoare-Kalküls
- Strukturierte Datentypen
- Verifikationsbedingungen
- Vorwärts mit Floyd und Hoare
- Modellierung
- Spezifikation von Funktionen
- Referenzen und Speichermodelle
- Funktionsaufrufe und das Framing-Problem
- Ausblick und Rückblick

Funktionen & Prozeduren

- **Funktionen** sind das zentrale Modularisierungskonzept von C
  - Kleinste Einheit
  - NB. Prozeduren sind nur Funktionen vom Typ **void**
- In objektorientierten Sprachen: Methoden
  - Funktionen mit (implizitem) erstem Parameter **this**
- Wie behandeln wir Funktionen?

Modellierung und Spezifikation von Funktionen

Wir brauchen:

1. Von Anweisungen zu Funktionen: Deklarationen und Parameter

Modellierung und Spezifikation von Funktionen

Wir brauchen:

1. Von Anweisungen zu Funktionen: Deklarationen und Parameter
2. Semantik von Funktionsdefinitionen

Modellierung und Spezifikation von Funktionen

Wir brauchen:

1. Von Anweisungen zu Funktionen: Deklarationen und Parameter
2. Semantik von Funktionsdefinitionen
3. Spezifikation von Funktionsdefinitionen

Modellierung und Spezifikation von Funktionen

Wir brauchen:

1. Von Anweisungen zu Funktionen: Deklarationen und Parameter
2. Semantik von Funktionsdefinitionen
3. Spezifikation von Funktionsdefinitionen
4. Beweisregeln für Funktionsdefinitionen

Modellierung und Spezifikation von Funktionen

Wir brauchen:

1. Von Anweisungen zu Funktionen: Deklarationen und Parameter
2. Semantik von Funktionsdefinitionen
3. Spezifikation von Funktionsdefinitionen
4. Beweisregeln für Funktionsdefinitionen
5. Semantik des Funktionsaufrufs

Modellierung und Spezifikation von Funktionen

Wir brauchen:

1. Von Anweisungen zu Funktionen: Deklarationen und Parameter
2. Semantik von Funktionsdefinitionen
3. Spezifikation von Funktionsdefinitionen
4. Beweisregeln für Funktionsdefinitionen
5. Semantik des Funktionsaufrufs
6. Beweisregeln für Funktionsaufrufe

Von Anweisungen zu Funktionen

- Erweiterung unserer Kernsprache um Funktionsdefinition und Deklarationen:

$$
\begin{aligned}
\textbf{FunDef} &::= \textbf{FunHeader FunSpec}^{+}\ \textbf{Blk} \\
\textbf{FunHeader} &::= \textbf{Type Idt}(\textbf{Decl}^{*}) \\
\textbf{Decl} &::= \textbf{Type Idt} \\
\textbf{Blk} &::= \{\textbf{Decl}^{*}\ \textbf{Stmt}\} \\
\textbf{Type} &::= \textbf{char} \mid \textbf{int} \mid \textbf{Struct} \mid \textbf{Array} \\
\textbf{Struct} &::= \textbf{struct}\ \ \textbf{Idt}^{?}\ \{\textbf{Decl}^{+}\} \\
\textbf{Array} &::= \textbf{Type Idt}[\textbf{Aexp}]
\end{aligned}
$$

- Abstrakte Syntax
- Größe von Feldern: **konstanter** Ausdruck
- **FunSpec** wird später erläutert

Rückgaben

Neue Anweisungen: Return-Anweisung

Stmt $s ::= l = e \mid c_1; c_2 \mid \{\,\} \mid \textbf{if}\ (b)\ c_1\ \textbf{else}\ c_2$
$\mid \textbf{while}\ (b)\ //{**}\ \textbf{inv}\ a\ {*}/\ c \mid //{**}\ \{a\}\ {*}/$
$\mid \textbf{return}\ a^{?}$

Rückgabewerte

- Problem: **return** bricht sequentiellen Kontrollfluss:

```
if (x == 0) return -1;
y = y / x;      // Wird nicht immer erreicht
```

- Lösung 1: verbieten!
  - MISRA-C (Guidelines for the use of the C language in critical systems):

Rule 14.7 (required)

A function shall have a single point of exit at the end of the function.

  - Nicht immer möglich, unübersichtlicher Code ...

- Lösung 2: Erweiterung der Semantik von $\Sigma \rightharpoonup \Sigma$ zu $\Sigma \rightharpoonup (\Sigma + \Sigma \times \mathbf{V})$

Erweiterte Semantik

- Denotat einer Anweisung: $\Sigma \rightharpoonup (\Sigma \cup \Sigma \times \mathbf{V})$
- Abbildung von Ausgangszustand Σ auf:
  - Sequentieller Folgezustand, oder
  - Rückgabewert und Rückgabezustand;
  - Σ und $\Sigma \times \mathbf{V}$ sind **disjunkt**.
- Was ist mit **void**?

Erweiterte Semantik

- Denotat einer Anweisung: $\Sigma \rightharpoonup (\Sigma \cup \Sigma \times \mathbf{V}_U)$
- Abbildung von Ausgangszustand Σ auf:
  - Sequentieller Folgezustand, oder
  - Rückgabewert und Rückgabezustand;
  - Σ und $\Sigma \times \mathbf{V}$ sind **disjunkt**.
- Was ist mit **void**?
  - Erweiterte Werte: $\mathbf{V}_U \stackrel{def}{=} \mathbf{V} + \{*\}$
- Komposition zweier Anweisungen $f, g : \Sigma \rightharpoonup (\Sigma \cup \Sigma \times \mathbf{V}_U)$:

$$g \circ_S f(\sigma) \stackrel{def}{=} \begin{cases} g(\sigma') & f(\sigma) = \sigma' \\ (\sigma', v) & f(\sigma) = (\sigma', v) \end{cases}$$

Semantik von Anweisungen

$$\mathcal{C}[\![.]\!] : \textbf{Stmt} \to \Sigma \rightharpoonup (\Sigma \cup \Sigma \times \mathbf{V}_U)$$

$$\begin{aligned}
\mathcal{C}[\![x = e]\!] &= \{(\sigma, \sigma[a/l]) \mid (\sigma, l) \in \mathcal{L}[\![x]\!], (\sigma, a) \in \mathcal{A}[\![e]\!]\} \\
\mathcal{C}[\![c_1; c_2]\!] &= \mathcal{C}[\![c_2]\!] \circ_S \mathcal{C}[\![c_1]\!] \qquad \text{Komposition wie oben} \\
\mathcal{C}[\![\{\ \}]\!] &= \mathbf{Id}_\Sigma \qquad\qquad \mathbf{Id}_\Sigma := \{(\sigma, \sigma) | \sigma \in \Sigma\} \\
\mathcal{C}[\![\textbf{if } (b)\ c_0\ \textbf{ else }\ c_1]\!] &= \{(\sigma, \rho') \mid (\sigma, \mathit{true}) \in \mathcal{B}[\![b]\!] \wedge (\sigma, \rho') \in \mathcal{C}[\![c_0]\!]\} \\
&\quad \cup \{(\sigma, \rho') \mid (\sigma, \mathit{false}) \in \mathcal{B}[\![b]\!] \wedge (\sigma, \rho') \in \mathcal{C}[\![c_1]\!]\} \\
&\quad \text{mit } \rho' \in \Sigma \cup \Sigma \times \mathbf{V}_U \\
\mathcal{C}[\![\textbf{return }\ e]\!] &= \{(\sigma, (\sigma, a)) \mid (\sigma, a) \in \mathcal{A}[\![e]\!]\} \\
\mathcal{C}[\![\textbf{return}]\!] &= \{(\sigma, (\sigma, *))\}
\end{aligned}$$

$$\mathcal{C}[\![\textbf{while } (b)\ c]\!] = \mathit{fix}(\Gamma)$$

$$\begin{aligned}
\Gamma(\psi) \stackrel{\mathit{def}}{=} &\{(\sigma, \rho') \mid (\sigma, \mathit{true}) \in \mathcal{B}[\![b]\!] \wedge (\sigma, \rho') \in \psi \circ_S \mathcal{C}[\![c]\!]\} \\
&\cup \{(\sigma, \sigma) \mid (\sigma, \mathit{false}) \in \mathcal{B}[\![b]\!]\}
\end{aligned}$$

Semantik von Funktionsdefinitionen

$$\mathcal{D}_{fd}[\![.]\!] : \mathbf{FunDef} \to \mathbf{V}^n \rightharpoonup \Sigma \rightharpoonup \Sigma \times \mathbf{V}_U$$

Das Denotat einer Funktion ist eine Anweisung, die über den tatsächlichen Werten für die Funktionsargumente parametriert ist.

$$\mathcal{D}_{fd}[\![f(t_1\ p_1, t_2\ p_2, \ldots, t_n\ p_n)\ blk]\!]v_1, \ldots, v_n = \{(\sigma, (\sigma', v)) \mid (\sigma[v_1/p_1, \ldots, v_n/p_n], (\sigma', v)) \in \mathcal{D}_{blk}[\![blk]\!]\}\}$$

- Die Funktionsargumente sind lokale Deklarationen, die mit den Aufrufwerten initialisiert werden.
  - Insbesondere können sie lokal in der Funktion verändert werden.

Semantik von Blöcken und Deklarationen

Blöcke bestehen aus Deklarationen und einer Anweisung.

$$\mathcal{D}_{blk}[\![.]\!] : \textbf{Blk} \to \Sigma \rightharpoonup (\Sigma \times V_U)$$

$$\mathcal{D}_{blk}[\![decls\ stmts]\!] \stackrel{def}{=} \{(\sigma, (\sigma', v)) \mid (\sigma, (\sigma', v)) \in \mathcal{C}[\![stmts]\!]\}$$

- Von $\mathcal{C}[\![stmts]\!]$ sind nur **Rückgabezustände** interessant.
  - Kein „fall-through“
  - Was passiert ohne **return** am Ende?
- Keine Initialisierungen, Deklarationen haben (noch) keine Semantik.

Spezifikation von Funktionen

- Wir **spezifizieren** Funktionen durch **Vor-** und **Nachbedingungen**
  - Ähnlich den Hoare-Tripeln, aber vereinfachte Syntax
  - **Behavioural specification**, angelehnt an JML, OCL, ACSL (Frama-C)
- Syntaktisch:

$$\textbf{FunSpec} ::= \texttt{/**}\ \texttt{pre}\ \textbf{Assn}\ \texttt{post}\ \textbf{Assn}\ \texttt{*/}$$

Vorbedingung	pre sp;	$\Sigma \to \mathbb{B}$
Nachbedingung	post sp;	$\Sigma \times (\Sigma \times \mathbf{V}_U) \to \mathbb{B}$
	\old(e)	Wert von e im **Vorzustand**
	\ result	**Rückgabewert** der Funktion

Beispiel: Fakultät

```
int fac(int n)
/** pre 0 ≤ n;
    post \result == n!;
 */
{
 int p;
 int c;

 p= 1;
 c= 1;
 while (c<= n) /** inv p == (c − 1)! ∧ c ≤ n + 1 ∧ 0 < c */ {
    p= p*c;
    c= c+1;
    }
 return p;
}
```

Beispiel: Suche

```
int findmax(int a[], int a_len)
  /** pre  \array(a, a_len) ∧ 0 < a_len;
       post ∀i. 0 ≤ i < a_len ⟶ a[i] ≤ \result;  */
{
  int x; int j;

  x= INT_MIN; j= 0;
  while (j< a_len)
    /** inv (∀i. 0 ≤ i < j ⟶ a[i] ≤ x) ∧ j ≤ a_len;  */
    {
    if (a[j]> x) x= a[j];
    j= j+1;
    }
  return x;
}
```

Beispiel: Suche

```
int findmax(int a[], int a_len)
  /** pre  0 < a_len;
      post \result = max(seq(a, a_len)); */
{
  int x; int j;

  x= INT_MIN; j= 0;
  while (j< a_len)
    /** inv j > 0 ⟶ x = max(Seq(a, j)) ∧ j ≤ a_len; */
    {
    if (a[j]> x) x= a[j];
    j= j+1;
    }
  return x;
}
```

Semantik von Spezifikationen

- Vorbedingung: Auswertung als $\mathcal{B}[\![sp]\!]\,\Gamma$ über dem Vorzustand
- Nachbedingung: Erweiterung von $\mathcal{B}[\![.]\!]$ und $\mathcal{A}[\![.]\!]$
  - Ausdrücke können in Vor- oder Nachzustand ausgewertet werden.
  - **result** kann nicht in Funktionen vom Typ **void** auftreten.

$$\mathcal{B}_{sp}[\![.]\!] : \mathbf{Env} \to \mathbf{Assn} \rightharpoonup (\Sigma \times (\Sigma \times \mathbf{V}_U)) \to \mathbb{B}$$

$$\mathcal{A}_{sp}[\![.]\!] : \mathbf{Env} \to \mathbf{Aexpv} \rightharpoonup (\Sigma \times (\Sigma \times \mathbf{V}_U)) \to \mathbf{V}$$

$$\mathcal{B}_{sp}[\![!b]\!]\,\Gamma = \{((\sigma,(\sigma',v)),\mathit{true}) \mid ((\sigma,(\sigma',v)),\mathit{false}) \in \mathcal{B}_{sp}[\![b]\!]\Gamma\} \cup \{((\sigma,(\sigma',v)),\mathit{false}) \mid ((\sigma,(\sigma',v)),\mathit{true}) \in \mathcal{B}_{sp}[\![b]\!]\Gamma$$

$$\ldots$$

$$\mathcal{B}_{sp}[\![\backslash\mathsf{old}(e)]\!]\,\Gamma = \{((\sigma,(\sigma',v)),b) \mid (\sigma,b) \in \mathcal{B}[\![e]\!]\,\Gamma\}$$

$$\mathcal{A}_{sp}[\![\backslash\mathsf{old}(e)]\!]\,\Gamma = \{((\sigma,(\sigma',v)),a) \mid (\sigma,a) \in \mathcal{A}[\![e]\!]\,\Gamma\}$$

$$\mathcal{A}_{sp}[\![\backslash\,\mathsf{result}\,]\!]\,\Gamma = \{((\sigma,(\sigma,v)),v)\}$$

$$\mathcal{B}_{sp}[\![\mathsf{pre}\ p\ \mathsf{post}\ q]\!]\,\Gamma = \{(\sigma,(\sigma',v)) \mid \sigma \in \mathcal{B}[\![p]\!]\,\Gamma \wedge (\sigma',(\sigma,v)) \in \mathcal{B}_{sp}[\![p]\!]\,\Gamma\}$$

Gültigkeit von Spezifikationen

- Die Semantik von Spezifikationen erlaubt uns die Definition der **semantischen Gültigkeit**.

$$\text{pre } p \text{ post } q \models fd$$
$$\iff \forall v_1, \ldots, v_n.\, \mathcal{D}_{fd}[\![fd]\!]\ \Gamma\ v_1 \ldots v_n \in \mathcal{B}_{sp}[\![\text{pre } p \text{ post } q]\!]\ \Gamma$$

  - Γ enthält globale Definitionen, insbesondere andere Funktionen.

- Wie passt das zu den Hoare-Tripeln $\models \{P\}\, c\, \{Q\}$?

- Wie **beweisen** wir das?

Gültigkeit von Spezifikationen

- Die Semantik von Spezifikationen erlaubt uns die Definition der **semantischen Gültigkeit**.

$$\text{pre } p \text{ post } q \models fd$$
$$\iff \forall v_1, \ldots, v_n.\, \mathcal{D}_{fd}[\![fd]\!]\ \Gamma\ v_1 \ldots v_n \in \mathcal{B}_{sp}[\![\text{pre } p \text{ post } q]\!]\ \Gamma$$

  - Γ enthält globale Definitionen, insbesondere andere Funktionen.

- Wie passt das zu den Hoare-Tripeln $\models \{P\}\, c\, \{Q\}$?

- Wie **beweisen** wir das? **Erweiterung** des Hoare-Kalküls

Erweiterung des Floyd-Hoare-Kalküls

$$\mathcal{C}[\![.]\!] : \mathbf{Stmt} \to \Sigma \rightharpoonup (\Sigma \cup \Sigma \times \mathbf{V}_U)$$

Hoare-Tripel: zusätzliche Spezifikation für **Rückgabewert**.

Partielle Korrektheit ($\models \{P\}\, c\, \{Q|Q_R\}$)

c ist **partiell korrekt**, wenn für alle Zustände σ, die P erfüllen:

- die Ausführung von c mit σ in σ' regulär terminiert, so dass σ' die Spezifikation Q erfüllt,
- oder die Ausführung von c in σ' mit dem Rückgabewert v terminiert, so dass (σ', v) die Rückgabespezifikation Q_R erfüllt.

$$\Gamma \models \{P\}\, c\, \{Q|Q_R\} \iff$$
$$\forall \sigma.\, (\sigma, \mathit{true}) \in \mathcal{B}[\![P]\!] \implies \exists \sigma'.\, (\sigma, \sigma') \in \mathcal{C}[\![c]\!] \wedge (\sigma', \mathit{true}) \in \mathcal{B}[\![Q]\!]$$
$$\vee$$
$$\exists \sigma', v.\, (\sigma, (\sigma', v)) \in \mathcal{C}[\![c]\!] \wedge ((\sigma', v), \mathit{true}) \in \mathcal{B}[\![Q_R]\!]$$

Erweiterung des Floyd-Hoare-Kalküls: return

$$\frac{}{\Gamma \vdash \{Q\}\ \textbf{return}\ \{P|Q\}} \qquad \frac{}{\Gamma \vdash \{Q[e/\backslash\textbf{result}]\}\ \textbf{return}\ e\ \{P|Q\}}$$

- Bei **return** wird die Rückgabespezifikation Q zur Vorbedingung, die reguläre Nachfolgespezifikation wird ignoriert, da die Ausführung von **return** kein Nachfolgezustand hat.
- **return** ohne Argument darf nur bei einer Nachbedingung Q auftreten, die kein **result** enthält.
- Bei **return** mit Argument ersetzt der Rückgabewert den **result** in der Rückgabespezifikation.

Erweiterung des Floyd-Hoare-Kalküls: Spezifikation

$$\frac{P \Longrightarrow P'[y_i/\,\backslash\mathbf{old}(y_i)] \quad \Gamma \vdash \{P'\}\, c\, \{false|Q\}}{\Gamma \vdash f(x_1, \ldots, x_n)/{*}{*}\ \text{pre}\ P\ \text{post}\ Q\ {*}/\ \{ds\ c\}}$$

- Die Parameter x_i werden per Konvention nur als x_i referenziert, aber es ist immer der Wert im **Vorzustand** gemeint (eigentlich $\backslash\mathbf{old}(x_i)$).
- Variablen unterhalb von $\backslash\mathbf{old}(y)$ werden bei der Substitution (Zuweisungsregel) **nicht ersetzt!**
- $\backslash\mathbf{old}(y)$ wird beim Weakening von der Vorbedingung P ersetzt
- Sequentielle Nachbedingung von c ist *false*

Zusammenfassung: Erweiterter Floyd-Hoare-Kalkül

$$\frac{}{\Gamma \vdash \{P\}\,\{\}\,\{P|Q_R\}} \qquad \frac{\Gamma \vdash \{P\}\,c_1\,\{R|Q_R\} \quad \Gamma \vdash \{R\}\,c_2\,\{Q|Q_R\}}{\Gamma \vdash \{P\}\,c_1; c_2\,\{Q|Q_R\}}$$

$$\frac{}{\Gamma \vdash \{Q[e/x]\}\,x = e\,\{Q|Q_R\}} \qquad \frac{\Gamma \vdash \{P \wedge b\}\,c\,\{P|Q_R\}}{\Gamma \vdash \{P\}\ \textbf{while}\ (b)\ c\,\{P \wedge \neg b|Q_R\}}$$

$$\frac{\Gamma \vdash \{P \wedge b\}\,c_1\,\{Q|Q_R\} \quad \Gamma \vdash \{P \wedge \neg b\}\,c_2\,\{Q|Q_R\}}{\Gamma \vdash \{P\}\ \textbf{if}\ (b)\ c_1\ \textbf{else}\ c_2\,\{Q|Q_R\}}$$

$$\frac{P \longrightarrow P' \quad \Gamma \vdash \{P'\}\,c\,\{Q'|R'\} \quad Q' \longrightarrow Q \quad R' \longrightarrow R}{\Gamma \vdash \{P\}\,c\,\{Q|R\}}$$

Erweiterter Floyd-Hoare-Kalkül II

$$\frac{}{\Gamma \vdash \{Q\}\ \textbf{return}\ \{P|Q\}} \qquad \frac{}{\Gamma \vdash \{Q[e/\,\backslash\textbf{result}]\}\ \textbf{return}\ \ e\,\{P|Q\}}$$

$$\frac{P \Longrightarrow P'[y_i/\,\backslash\textbf{old}(y_i)] \quad \Gamma \vdash \{P'\}\,c\,\{\mathit{false}|Q\}}{\Gamma \vdash f(x_1,\ldots,x_n)/{*}{*}\ \text{pre}\ P\ \text{post}\ Q\ {*}/\ \{ds\ c\}}$$

Approximative schwächste Vorbedingung

- Erweiterung zu $\mathrm{awp}(\Gamma, c, Q, Q_R)$ und $\mathrm{wvc}(\Gamma, c, Q, Q_R)$ analog zu der Erweiterung der Floyd-Hoare-Regeln.
  - Es werden der **Kontext** Γ und eine **Rückgabespezifikation** Q_R benötigt.
- Es gilt:

$$\bigwedge \mathrm{wvc}(\Gamma, c, Q, Q_R) \Longrightarrow \Gamma \models \{\mathrm{awp}(c, Q, Q_R)\}\, c\, \{Q | Q_R\}$$

- Berechnung von awp und wvc:

$$\mathrm{awp}(\Gamma, f(x_1, \ldots, x_n)/{**}\ \mathrm{pre}\ P\ \mathrm{post}\ Q\ {*}/\ \{ds\ blk\}) \stackrel{def}{=} \mathrm{awp}(\Gamma', blk, false, Q)$$

$$\mathrm{wvc}(\Gamma, f(x_1, \ldots, x_n)/{**}\ \mathrm{pre}\ P\ \mathrm{post}\ Q\ {*}/\ \{ds\ blk\}) \stackrel{def}{=}$$

$$\{P \Longrightarrow \mathrm{awp}(\Gamma', blk, Q, Q)[y_j / \backslash\mathbf{old}(y_j)]\} \cup \mathrm{wvc}(\Gamma', blk, false, Q)$$

$$\Gamma' \stackrel{def}{=} \Gamma[f \mapsto \forall x_1, \ldots, x_n.\, (P, Q)]$$

Approximative schwächste Vorbedingung (Revisited)

$$
\begin{aligned}
\mathsf{awp}(\Gamma, \{\,\}, Q, Q_R) &\stackrel{def}{=} Q \\
\mathsf{awp}(\Gamma, l = e, Q, Q_R) &\stackrel{def}{=} P[e/l] \\
\mathsf{awp}(\Gamma, c_1; c_2, Q, Q_R) &\stackrel{def}{=} \mathsf{awp}(\Gamma, c_1, \mathsf{awp}(c_2, Q, Q_R), Q_R) \\
\mathsf{awp}(\Gamma, \textbf{if } (b)\ c_0\ \textbf{ else }\ c_1, Q, Q_R) &\stackrel{def}{=} (b \wedge \mathsf{awp}(\Gamma, c_0, Q, Q_R)) \\
&\quad \vee (\neg b \wedge \mathsf{awp}(\Gamma, c_1, Q, Q_R)) \\
\mathsf{awp}(\Gamma, //{**}\ \{q\}\ {*}/, Q, Q_R) &\stackrel{def}{=} q \\
\mathsf{awp}(\Gamma, \textbf{while } (b)\ //{**}\ \textbf{inv }\ i\ {*}/\ c, Q_R) &\stackrel{def}{=} i \\
\mathsf{awp}(\Gamma, \textbf{return }\ e, Q, Q_R) &\stackrel{def}{=} Q_R[e/\backslash\textbf{result}] \\
\mathsf{awp}(\Gamma, \textbf{return}, Q, Q_R) &\stackrel{def}{=} Q_R
\end{aligned}
$$

Approximative Verifikationsbedingungen (Revisited)

$$
\begin{aligned}
\mathrm{wvc}(\Gamma, \{\,\}, Q, Q_R) &\stackrel{def}{=} \emptyset \\
\mathrm{wvc}(\Gamma, x = e, Q, Q_R) &\stackrel{def}{=} \emptyset \\
\mathrm{wvc}(\Gamma, c_1; c_2, Q, Q_R) &\stackrel{def}{=} \mathrm{wvc}(\Gamma, c_1, \mathrm{awp}(c_2, Q, Q_R), Q_R) \\
&\quad \cup\ \mathrm{wvc}(\Gamma, c_2, Q, Q_R) \\
\mathrm{wvc}(\Gamma, \textbf{if}\ (b)\ c_1\ \textbf{else}\ c_2, Q, Q_R) &\stackrel{def}{=} \mathrm{wvc}(\Gamma, c_1, Q, Q_R) \cup \mathrm{wvc}(\Gamma, c_2, Q, Q_R) \\
\mathrm{wvc}(\Gamma, //{**}\ \{q\}\ {*}/, Q, Q_R) &\stackrel{def}{=} \{q \Longrightarrow Q\} \\
\mathrm{wvc}(\Gamma, \textbf{while}\ (b)\ //{**}\ \textbf{inv}\ i\ {*}/\ c, Q, Q_R) &\stackrel{def}{=} \mathrm{wvc}(\Gamma, c, i, Q_R) \\
&\quad \cup \{i \wedge b \Longrightarrow \mathrm{awp}(\Gamma, c, i, Q_R)\} \\
&\quad \cup \{i \wedge \neg b \Longrightarrow Q\} \\
\mathrm{wvc}(\Gamma, \textbf{return}\ e, Q, Q_R) &\stackrel{def}{=} \emptyset
\end{aligned}
$$

Beispiel: Fakultät

```
 1  int fac(int n)
 2  /** pre 0 ≤ n;
 3      post \result == n!;
 4   */
 5  {
 6   int p;
 7   int c;
 8
 9   p= 1;
10   c= 1;
11   while (1) /** inv p == (c − 1)! */ {
12       if (c == n) return p;
13       p= p∗c;
14       c= c+1;
15       }
16  }
```

Beispiel: Fakultät (berichtigt)

```
 1  int fac(int n)
 2  /** pre 0 ≤ n;
 3      post \result == n!;
 4   */
 5  {
 6   int p;
 7   int c;
 8
 9   p= 1;
10   c= 0;
11   while (1) /** inv p == c! */ {
12       if (c == n) return p;
13       c= c+1;
14       p= p*c;
15       }
16  }
```

Zusammenfassung

- Funktionen sind **zentrales Modularisierungskonzept**
- Wir müssen Funktionen **modular** verifizieren können
- Erweiterung der **Semantik:**
  - Semantik von Deklarationen und Parameter — straightforward
  - Semantik von **Rückgabewerten** — Erweiterung der Semantik
- Erweiterung der **Spezifikationen:**
  - Spezifikation von Funktionen: **Vor-/Nachzustand** statt logischer Variablen
- Erweiterung des Hoare-Kalküls:
  - Environment, um andere Funktionen zu nutzen
  - Gesonderte Nachbedingung für Rückgabewert/Endzustand
- Es fehlt: **Funktionsaufruf** und **Parameterübergabe**