Korrekte Software: Grundlagen und Methoden
Vorlesung 8 vom 22.05.17: Funktionen und Prozeduren

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2017

Fahrplan

- Einführung
- Die Floyd-Hoare-Logik
- Operationale Semantik
- Denotationale Semantik
- Äquivalenz der Operationalen und Denotationalen Semantik
- Korrektheit des Hoare-Kalküls
- Vorwärts und Rückwärts mit Floyd und Hoare
- Funktionen und Prozeduren
- Referenzen und Speichermodelle
- Verifikationsbedingungen Revisited
- Vorwärtsrechnung Revisited
- Programmsicherheit und Frame Conditions
- Ausblick und Rückblick

Funktionen & Prozeduren

- Funktionen sind das zentrale Modularisierungskonzept von C
  - Kleinste Einheit
  - NB. Prozeduren sind nur Funktionen vom Typ **void**
  - Auch in den meisten anderen Sprachen, meist mit Zustandsverkapselung (Methoden)
- Wie behandeln wir Funktionen?

Modellierung und Spezifikation von Funktionen

Wir brauchen:

(1) Von Anweisungen zu Funktionen: Deklarationen und Parameter

(2) Semantik von Funktionsdefinition und Funktionsaufruf

(3) Spezifikation von Funktionen

(4) Beweisregeln für Funktionsdefinition und Funktionsaufruf

Von Anweisungen zu Funktionen

- Erweiterung unserer Kernsprache:

$$\begin{aligned}
\textbf{FunDef} &::= \textbf{Type}\ \mathit{Id}(\textbf{Param}^*)\ \textbf{FunSpec}^+\ \textbf{Blk} \\
\textbf{Param} &::= \textbf{Type}\ \mathit{Id} \\
\textbf{Blk} &::= \{\textbf{Decl}^*\ \textbf{Stmt}\} \\
\textbf{Decl} &::= \textbf{Type}\ \mathit{Id}\ = \textbf{Aexp} \mid \textbf{Type}\ \mathit{Id} \\
\\
\textbf{Aexp} &::= \ldots \mid \mathit{Id}(\textbf{Aexp}^*) \\
\textbf{Stmt} &::= \ldots \mid \mathit{Id}(\textbf{Aexp}^*) \mid \textbf{return}\ (\textbf{Aexp})?
\end{aligned}$$

- **Type**: zur Zeit nur **int**; Initialisierer: konstanter Ausdruck
- **FunSpec** später
- Vereinfachte Syntax (konkrete Syntax mischt **Type** und *Id*, Kommata bei Argumenten, ...)

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: $\Sigma \rightharpoonup (\Sigma + \Sigma \times \mathbf{V})$
- Abbildung von Ausgangszustand Σ auf:
  - Sequentieller Folgezustand, oder
  - Rückgabewert und Rückgabezustand
- Was ist mit **void**?

Erweiterte Semantik

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

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

Semantik von Anweisungen

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

$$\begin{aligned}
\mathcal{C}[\![x = e]\!] &= \{(\sigma, \sigma[a/x]) \mid (\sigma, a) \in \mathcal{A}[\![e]\!]\} \\
\mathcal{C}[\![\{c\ c_s\}]\!] &= \mathcal{C}[\![c_s]\!] \cdot_S \mathcal{C}[\![c]\!] \qquad \text{Komposition wie oben} \\
\mathcal{C}[\![\{\ \}]\!] &= \mathbf{Id} \qquad\qquad \mathbf{Id} := \{(\sigma, \sigma) | \sigma \in \Sigma\} \\
\mathcal{C}[\![\textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1]\!] &= \{(\sigma, \tau) \mid (\sigma, 1) \in \mathcal{B}[\![b]\!] \wedge (\sigma, \tau) \in \mathcal{C}[\![c_0]\!]\} \\
&\quad \cup \{(\sigma, \tau) \mid (\sigma, 0) \in \mathcal{B}[\![b]\!] \wedge (\sigma, \tau) \in \mathcal{C}[\![c_1]\!]\} \\
&\quad \text{mit } \tau \in \Sigma \cup (\Sigma \times \mathbf{V}_U) \\
\mathcal{C}[\![\textbf{return}\ e]\!] &= \{(\sigma, (\sigma, a)) \mid (\sigma, a) \in \mathcal{A}[\![e]\!]\} \\
\mathcal{C}[\![\textbf{return}]\!] &= \{(\sigma, (\sigma, *))\}
\end{aligned}$$

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

$$\begin{aligned}
\Gamma(\psi) \stackrel{\mathit{def}}{=} &\{(\sigma, \tau) \mid (\sigma, 1) \in \mathcal{B}[\![b]\!] \wedge (\sigma, \tau) \in \psi \cdot_S \mathcal{C}[\![c]\!]\} \\
&\cup \{(\sigma, \sigma) \mid (\sigma, 0) \in \mathcal{B}[\![b]\!]\}
\end{aligned}$$

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.

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

- Die Funktionsargumente sind lokale Deklarationen, die mit den Aufrufwerten initialisiert werden.
  - Insbesondere können sie lokal in der Funktion verändert werden.
- Von $\mathcal{D}_{blk}[\![blk]\!]$ sind nur Rückgabezustände interessant.

Semantik von Blöcken und Deklarationen

$$\mathcal{D}_{blk}[\![.]\!] : \mathbf{Blk} \rightharpoonup \Sigma \rightharpoonup (\Sigma + \Sigma \times \mathbf{V}_U)$$
$$\mathcal{D}_{d}[\![.]\!] : \mathbf{Decl} \rightharpoonup \Sigma \rightharpoonup (\Sigma + \Sigma \times \mathbf{V}_U)$$

Blöcke bestehen aus Deklarationen und einer Anweisung:

$$\mathcal{D}_{blk}[\![decls\ stmts]\!] = \mathcal{C}[\![stmts]\!] \cdot_S \mathcal{D}_d[\![decls]\!]$$
$$\mathcal{D}_d[\![t\ i]\!] = \{(\sigma, \sigma[\perp/i])\}$$
$$\mathcal{D}_d[\![t\ i\ = init]\!] = \{(\sigma, \sigma[\mathcal{A}_{init}[\![init]\!]/i])\}$$

- Verallgemeinerung auf Sequenz von Deklarationen
- $\mathcal{A}_{init}[\![.]\!]$ ist das Denotat von Initialisierungen:
  - Nur für konstante Ausdrücke, daher nicht zustandsabhängig

Funktionsaufrufe

- Aufruf einer Funktion: $f(t_1, \ldots, t_n)$:
  - Auswertung der Argumente $t_1, \ldots, t_n$
  - Einsetzen in die Semantik $\mathcal{D}_{fd}[\![f]\!]$
- Was ist mit Seiteneffekten?
  - Erst mal gar nichts, unsere Sprache hat noch keine ...
- Call by name, call by value, call by reference...?
  - C kennt nur call by value (C-Standard 99, §6.9.1. (10))

Funktionsaufrufe

- Um eine Funktion f aufzurufen, müssen wir (statisch!) die Semantik der Definition von f dem Bezeichner f zuordnen.

- Deshalb brauchen wir eine Umgebung (Environment):

$$\begin{aligned} Env &= Id \rightharpoonup [\![\mathbf{FunDef}]\!] \\ &= Id \rightharpoonup \mathbf{V}^N \rightharpoonup \Sigma \rightharpoonup (\Sigma \times \mathbf{V}_u) \end{aligned}$$

- Das Environment ist zusätzlicher Parameter für alle Definitionen

Semantik von Funktionsaufrufen

$$\mathcal{A}[\![f(t_1,\ldots,t_n)]\!]\Gamma = \{(\sigma,v) \mid \exists \sigma', v.\, (\sigma,(\sigma',v)) \in \Gamma(f)(v_1,\ldots v_n) \\ \wedge (\sigma, v_i) \in \mathcal{A}[\![t_i]\!]\Gamma\}$$

$$\mathcal{C}[\![f(t_1,\ldots,t_n]\!]\Gamma = \{(\sigma,\sigma') \mid \exists \sigma'.\, (\sigma,(\sigma',*)) \in \Gamma(f)(v_1,\ldots v_n) \\ \wedge (\sigma, v_i) \in \mathcal{A}[\![t_i]\!]\Gamma\}$$

- Aufruf einer nicht-definierten Funktion f oder mit falscher Anzahl n von Parametern ist nicht definiert
  - Muss durch statische Analyse verhindert werden
- Aufruf von Funktion $\mathcal{A}[\![f(t_1,\ldots,t_n)]\!]$ ignoriert Endzustand
- Aufruf von Prozedur $\mathcal{C}[\![f(t_1,\ldots,t_n)]\!]$ ignoriert Rückgabewert

Semantik von Funktionsaufrufen

$$\mathcal{A}[\![f(t_1,\ldots,t_n)]\!]\Gamma = \{(\sigma,v) \mid \exists \sigma', v.\, (\sigma,(\sigma',v)) \in \Gamma(f)(v_1,\ldots v_n) \\ \wedge (\sigma, v_i) \in \mathcal{A}[\![t_i]\!]\Gamma\}$$

$$\mathcal{C}[\![f(t_1,\ldots,t_n]\!]\Gamma = \{(\sigma,\sigma') \mid \exists \sigma'.\, (\sigma,(\sigma',*)) \in \Gamma(f)(v_1,\ldots v_n) \\ \wedge (\sigma, v_i) \in \mathcal{A}[\![t_i]\!]\Gamma\}$$

$$\mathcal{C}[\![x = f(t_1,\ldots,t_n]\!]\Gamma = \{(\sigma,\sigma'[v/x]) \mid \exists \sigma', v.\, (\sigma,(\sigma',v)) \in \Gamma(f)(v_1,\ldots v_n) \\ \wedge (\sigma, v_i) \in \mathcal{A}[\![t_i]\!]\Gamma\}$$

- Aufruf einer nicht-definierten Funktion f oder mit falscher Anzahl n von Parametern ist nicht definiert
  - Muss durch statische Analyse verhindert werden
- Aufruf von Funktion $\mathcal{A}[\![f(t_1,\ldots,t_n)]\!]$ ignoriert Endzustand
- Aufruf von Prozedur $\mathcal{C}[\![f(t_1,\ldots,t_n)]\!]$ ignoriert Rückgabewert
- Besser: Kombination mit Zuweisung

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:

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

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

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.

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

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

$$\begin{aligned}\mathcal{B}_{sp}[\![!b]\!]\,\Gamma = &\{((\sigma,(\sigma',v)),1) \mid ((\sigma,(\sigma',v)),0) \in \mathcal{B}_{sp}[\![b]\!]\Gamma\} \\ &\cup \{((\sigma,(\sigma',v)),0) \mid ((\sigma,(\sigma',v)),1) \in \mathcal{B}_{sp}[\![b]\!]\Gamma\}\end{aligned}$$

$$\ldots$$

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

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

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

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

Gültigkeit von Spezifikationen

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

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

  - Γ enthält globale Definitionen, insbesondere andere Funktionen.

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

- Wie beweisen wir das?

Gültigkeit von Spezifikationen

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

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

  - Γ enthält globale Definitionen, insbesondere andere Funktionen.

- Wie passt das zu $\models \{P\}\, c\, \{Q\}$ für Hoare-Tripel?
- Wie beweisen wir das? Erweiterung des Hoare-Kalküls

Erweiterung des Floyd-Hoare-Kalküls

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

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.

$$\begin{aligned}
&\models \{P\}\, c\, \{Q|Q_R\} \iff \\
&\quad \forall \sigma.\, \sigma \models \mathcal{B}[\![P]\!]\Gamma \implies \exists \sigma'.\, (\sigma, \sigma') \in \mathcal{C}[\![c]\!]\Gamma \wedge \sigma' \models \mathcal{B}[\![Q]\!]\Gamma \\
&\qquad\qquad\qquad\qquad \vee \\
&\qquad\qquad\qquad\qquad \exists \sigma', v.\, (\sigma, (\sigma', v)) \in \mathcal{C}[\![c]\!]\Gamma \wedge (\sigma', v) \models \mathcal{B}[\![Q_R]\!]\Gamma
\end{aligned}$$

Kontext

- Wir benötigen ferner einen Kontext Γ, der Funktionsbezeichnern ihre Spezifikation (Vor/Nachbedingung) zuordnet.
- $\Gamma(f) = \forall x_1, \ldots, x_n. (P, Q)$, für Funktion $f(x_1, \ldots, x_n)$ mit Vorbedingung P und Nachbedingung Q.
- Notation: $\Gamma \models \{P\}\, c\, \{Q|Q_R\}$ und $\Gamma \vdash \{P\}\, c\, \{Q|Q_R\}$
- Korrektheit gilt immer nur im Kontext, dadurch kann jede Funktion separat verifiziert werden (Modularität)

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

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

- 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

Erweiterung des Floyd-Hoare-Kalküls: Aufruf

$$\frac{\Gamma(f) = \forall x_1, \ldots, x_n.\,(P, Q), f \text{ vom Typ } \textbf{void}}{\begin{array}{l}\Gamma \vdash \{Y_j = y_j \;\&\&\; P[t_i/x_i]\} \\ \quad f(t_1, \ldots, t_n) \\ \quad \{Q[t_i/x_i][Y_j/\,\backslash\textbf{old}(y_j)] | Q_R\}\end{array}}$$

$$\frac{\Gamma(f) = \forall x_1, \ldots, x_n.\,(P, Q)}{\begin{array}{l}\Gamma \vdash \{Y_j = y_j \;\&\&\; P[t_i/x_i]\} \\ \quad x = f(t_1, \ldots, t_n) \\ \quad \{Q[t_i/x_i][Y_j/\,\backslash\textbf{old}(y_j)][x/\,\backslash\textbf{result}] | Q_R\}\end{array}}$$

- Γ muss f mit der Vor-/Nachbedingung P, Q enthalten
- In P und Q werden Parameter x_i durch Argumente t_i ersetzt.
- $y_1, \ldots, y_m$ sind die als $\backslash\textbf{old}(y_j)$ in Q auftretenden Variablen
- $Y_1, \ldots, Y_m$ dürfen nicht in P oder Q enthalten sein
- Im ersten Fall (Aufruf als Prozedur) enthält Q kein $\backslash\textbf{result}$

Erweiterter Floyd-Hoare-Kalkül I

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

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

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

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

Erweiterter Floyd-Hoare-Kalkül II

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

$$\frac{\begin{array}{l}\Gamma[f \mapsto (P,Q)] \vdash \{X_i = x_i \;\&\&\; Y_j = y_j \;\&\&\; P\} \\ \qquad\qquad blk \\ \qquad\qquad \{Q[X_i/x_i][y_j/\backslash\textbf{old}(y_j)]|Q[X_i/x_i][y_j/\backslash\textbf{old}(y_j)]\}\end{array}}{\Gamma \vdash f(x_1,\ldots,x_n)/{*}{*}\ \text{pre}\ P\ \text{post}\ Q\ {*}{*}/\ \{ds\ blk\}}$$

$$\frac{\Gamma(f) = \forall x_1,\ldots,x_n.(P,Q), f\ \text{vom Typ}\ \textbf{void}}{\begin{array}{l}\Gamma \vdash \{Y_j = y_j \;\&\&\; P[t_i/x_i]\} \\ \qquad f(t_1,\ldots,t_n) \\ \qquad \{Q[t_i/x_i][Y_j/\backslash\textbf{old}(y_j)]|Q_R\}\end{array}}$$

$$\frac{\Gamma(f) = \forall x_1,\ldots,x_n.(P,Q)}{\begin{array}{l}\Gamma \vdash \{Y_j = y_j \;\&\&\; P[t_i/x_i]\} \\ \qquad x = f(t_1,\ldots,t_n) \\ \qquad \{Q[t_i/x_i][Y_j/\backslash\textbf{old}(y_j)][x/\backslash\textbf{result}]|Q_R\}\end{array}}$$

Beispiel: die Fakultätsfunktion, rekursiv

```
int fac(int x)
/** pre x >= 0
    post \result = factorial(x) */
{
 int r = 0;

 if (x == 0) { return 1; }
 r = fac(x- 1);
 return r * x;
}
```

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:

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

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

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

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

Approximative schwächste Vorbedingung (Revisited)

$$
\begin{aligned}
\mathrm{awp}(\Gamma, \{\,\}, Q, Q_R) &\stackrel{def}{=} Q \\
\mathrm{awp}(\Gamma, l = f(e_1, \ldots, e_n), Q, Q_R) &\stackrel{def}{=} P[e_i/x_i] \\
&\quad \text{mit } \Gamma(f) = \forall x_1, \ldots, x_n.\,(P, R) \\
\mathrm{awp}(\Gamma, f(e_1, \ldots, e_n), Q, Q_R) &\stackrel{def}{=} P[e_i/x_i] \\
&\quad \text{mit } \Gamma(f) = \forall x_1, \ldots, x_n.\,(P, R) \\
\mathrm{awp}(\Gamma, l = e, Q, Q_R) &\stackrel{def}{=} P[e/l] \\
\mathrm{awp}(\Gamma, \{c\ c_s\}, Q, Q_R) &\stackrel{def}{=} \mathrm{awp}(\Gamma, c, \mathrm{awp}(\{c_s\}, Q, Q_R), Q_R) \\
\mathrm{awp}(\Gamma, \textbf{if}\ (b)\ \{c_0\}\ \textbf{else}\ \{c_1\}, Q, Q_R) &\stackrel{def}{=} (b\ \&\&\ \mathrm{awp}(\Gamma, c_0, Q, Q_R)) \\
&\quad \|\ (!\,b\ \&\&\ \mathrm{awp}(\Gamma, c_1, Q, Q_R)) \\
\mathrm{awp}(\Gamma, /{**}\ \{q\}\ {*}/, Q, Q_R) &\stackrel{def}{=} q \\
\mathrm{awp}(\Gamma, \textbf{while}\ (b)\ /{**}\ \textbf{inv}\ i\ {*}/\ c, Q, Q_R) &\stackrel{def}{=} i \\
\mathrm{awp}(\Gamma, \textbf{return}\ e, Q, Q_R) &\stackrel{def}{=} Q_R[e/\backslash\textbf{result}] \\
\mathrm{awp}(\Gamma, \textbf{return}, Q, Q_R) &\stackrel{def}{=} Q_R
\end{aligned}
$$

Approximative Verifikationsbedingungen (Revisited)

$$\begin{aligned}
\mathrm{wvc}(\Gamma, \{\,\}, Q, Q_R) &\stackrel{def}{=} \emptyset \\
\mathrm{wvc}(\Gamma, x = e, Q, Q_R) &\stackrel{def}{=} \emptyset \\
\mathrm{wvc}(\Gamma, x = f(e_1, \ldots, e_n), Q, Q_R) &\stackrel{def}{=} \{R[e_i/x_i][x/\backslash\textbf{result}] \Longrightarrow Q\} \\
&\quad \text{mit } \Gamma(f) = \forall x_1, \ldots, x_n.\,(P, R) \\
\mathrm{wvc}(\Gamma, f(e_1, \ldots, e_n), Q, Q_R) &\stackrel{def}{=} \{R[e_i/x_i] \Longrightarrow Q\} \\
&\quad \text{mit } \Gamma(f) = \forall x_1, \ldots, x_n.\,(P, R) \\
\mathrm{wvc}(\Gamma, \{c\ c_s\}, Q, Q_R) &\stackrel{def}{=} \mathrm{wvc}(\Gamma, c, \mathrm{awp}(\{c_s\}, Q, Q_R), Q_R) \\
&\quad \cup\ \mathrm{wvc}(\Gamma, \{c_s\}, , Q, Q_R) \\
\mathrm{wvc}(\Gamma, \textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1, Q, Q_R) &\stackrel{def}{=} \mathrm{wvc}(\Gamma, c_0, Q, Q_R) \\
&\quad \cup\ \mathrm{wvc}(\Gamma, c_1, Q, Q_R) \\
\mathrm{wvc}(\Gamma, /{**}\ \{q\}\ {*}/, Q, Q_R) &\stackrel{def}{=} \{q \Longrightarrow Q\} \\
\mathrm{wvc}(\Gamma, \textbf{while}\ (b)\ /{**}\ \textbf{inv}\ i\ {*}/\ c, Q, Q_R) &\stackrel{def}{=} \mathrm{wvc}(\Gamma, c, i, Q_R) \\
&\quad \cup \{i \wedge b \Longrightarrow \mathrm{awp}(\Gamma, c, i, Q_R)\} \\
&\quad \cup \{i \wedge \neg b \Longrightarrow Q\} \\
\mathrm{wvc}(\Gamma, \textbf{return}\ e, Q, Q_R) &\stackrel{def}{=} \emptyset
\end{aligned}$$

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
  - Funktionsaufrufe — Environment, um Funktionsbezeichnern eine Semantik zu geben
- 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