

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:

**FunDef** ::= **Type** *Id*(**Param**<sup>\*</sup>) **FunSpec**<sup>+</sup> **Blk**

**Param** ::= **Type** *Id*

**Blk** ::= {**Decl**<sup>\*</sup> **Stmt**}

**Decl** ::= **Type** *Id* = **Aexp** | **Type** *Id*

**Aexp** ::= ... | *Id*(**Aexp**<sup>\*</sup>)

**Stmt** ::= ... | *Id*(**Aexp**<sup>\*</sup>) **return** (**Aexp**)?

- ▶ **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  $\Sigma \rightarrow \Sigma$  zu  $\Sigma \rightarrow (\Sigma + \Sigma \times \mathbf{V})$



## Erweiterte Semantik

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

$$g \cdot s f(\sigma) \stackrel{\text{def}}{=} \begin{cases} g(\sigma') & f(\sigma) = \sigma' \\ (\sigma', \nu) & f(\sigma) = (\sigma', \nu) \end{cases}$$



## Semantik von Anweisungen

$\mathcal{C}[\cdot] : \mathbf{Stmt} \rightarrow \Sigma \rightarrow (\Sigma + \Sigma \times \mathbf{V}_U)$

$\mathcal{C}[x = e] = \{(\sigma, \sigma[a/x]) \mid (\sigma, a) \in \mathcal{A}[e]\}$

$\mathcal{C}[\{c \ c_s\}] = \mathcal{C}[c_s] \cdot s \ \mathcal{C}[c]$  Komposition wie oben

$\mathcal{C}[\{\}] = \mathbf{Id}$   $\mathbf{Id} := \{(\sigma, \sigma) \mid \sigma \in \Sigma\}$

$\mathcal{C}[\mathbf{if} (b) \ c_0 \ \mathbf{else} \ c_1] = \{(\sigma, \tau) \mid (\sigma, 1) \in \mathcal{B}[b] \wedge (\sigma, \tau) \in \mathcal{C}[c_0]\} \cup \{(\sigma, \tau) \mid (\sigma, 0) \in \mathcal{B}[b] \wedge (\sigma, \tau) \in \mathcal{C}[c_1]\}$   
mit  $\tau \in \Sigma \cup (\Sigma \times \mathbf{V}_U)$

$\mathcal{C}[\mathbf{return} \ e] = \{(\sigma, (\sigma, a)) \mid (\sigma, a) \in \mathcal{A}[e]\}$

$\mathcal{C}[\mathbf{return}] = \{(\sigma, (\sigma, *))\}$

$\mathcal{C}[\mathbf{while} (b) \ c] = \text{fix}(\Gamma)$

$\Gamma(\psi) \stackrel{\text{def}}{=} \{(\sigma, \tau) \mid (\sigma, 1) \in \mathcal{B}[b] \wedge (\sigma, \tau) \in \psi \cdot s \ \mathcal{C}[c]\} \cup \{(\sigma, \sigma) \mid (\sigma, 0) \in \mathcal{B}[b]\}$



## Semantik von Funktionsdefinitionen

$$\mathcal{D}_{fd}[\cdot] : \text{FunDef} \rightarrow \mathbf{V}^n \rightarrow \Sigma \rightarrow \Sigma \times \mathbf{V}_U$$

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

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

- 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 Rückgebustände interessant.



## Semantik von Blöcken und Deklarationen

$$\mathcal{D}_{blk}[\cdot] : \text{Blk} \rightarrow \Sigma \rightarrow (\Sigma + \Sigma \times \mathbf{V}_U)$$

$$\mathcal{D}_d[\cdot] : \text{Decl} \rightarrow \Sigma \rightarrow (\Sigma + \Sigma \times \mathbf{V}_U)$$

Blöcke bestehen aus Deklarationen und einer Anweisung:

$$\mathcal{D}_{blk}[\text{decls stmts}] = \mathcal{C}[\text{stmts}] \cdot \mathcal{S} \mathcal{D}_d[\text{decls}]$$

$$\mathcal{D}_d[t i] = \{(\sigma, \sigma[\perp/i])\}$$

$$\mathcal{D}_d[t i = \text{init}] = \{(\sigma, \sigma[\mathcal{A}_{init}[\text{init}]/i])\}$$

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



## Funktionsaufrufe

Aufruf einer Funktion:  $f(t_1, \dots, t_n)$ :

- Auswertung der Argumente  $t_1, \dots, t_n$
- 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):

$$\text{Env} = \text{Id} \rightarrow \llbracket \text{FunDef} \rrbracket$$

$$= \text{Id} \rightarrow \mathbf{V}^N \rightarrow \Sigma \rightarrow (\Sigma \times \mathbf{V}_U)$$

Das Environment ist zusätzlicher Parameter für alle Definitionen



## Semantik von Funktionsaufrufen

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

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

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

- 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  $\mathcal{A}[f(t_1, \dots, t_n)]$  ignoriert Endzustand
- Aufruf von Prozedur  $\mathcal{C}[f(t_1, \dots, t_n)]$  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:

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

Vorbedingung **pre** sp;  $\Sigma \rightarrow \mathbf{T}$

Nachbedingung **post** sp;  $\Sigma \times (\Sigma \times \mathbf{V}_U) \rightarrow \mathbf{T}$

$\backslash \text{old}(e)$  Wert von  $e$  im Vorzustand

$\backslash \text{result}$  Rückgabewert der Funktion



## Semantik von Spezifikationen

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

$$\mathcal{B}_{sp}[\cdot] : \text{Env} \rightarrow \text{Bexp} \rightarrow (\Sigma \times (\Sigma \times \mathbf{V}_U)) \rightarrow \mathbf{T}$$

$$\mathcal{A}_{sp}[\cdot] : \text{Env} \rightarrow \text{Aexp} \rightarrow (\Sigma \times (\Sigma \times \mathbf{V}_U)) \rightarrow \mathbf{V}$$

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

...

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

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

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

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



## Gültigkeit von Spezifikationen

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

$$\text{pre } p \text{ post } q \models \text{FunDef}$$

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

- $\Gamma$  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

$$\mathcal{C}[\cdot] : \text{Stmt} \rightarrow \Sigma \rightarrow (\Sigma + \Sigma \times \mathbf{V}_U)$$

Hoare-Tripel: zusätzliche Spezifikation für Rückgabewert.

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

$c$  ist **partiell korrekt**, wenn für alle Zustände  $\sigma$ , die  $P$  erfüllen:

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

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



## Kontext

- ▶ Wir benötigen ferner einen **Kontext**  $\Gamma$ , der Funktionsbezeichnern ihre **Spezifikation** (Vor/Nachbedingung) zuordnet.
- ▶  $\Gamma(f) = \forall x_1, \dots, x_n. (P, Q)$ , für Funktion  $f(x_1, \dots, x_n)$  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

$$\Gamma \vdash \{Q\} \text{return } \{P|Q\} \quad \Gamma \vdash \{Q[e/\backslash\text{result}]\} \text{return } e \{P|Q\}$$

- ▶ 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

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

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



## Erweiterung des Floyd-Hoare-Kalküls: Aufruf

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

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



## Erweiterter Floyd-Hoare-Kalkül I

$$\frac{\Gamma \vdash \{P\} c \{R|Q_R\} \quad \Gamma \vdash \{R\} cs \{Q|Q_R\}}{\Gamma \vdash \{P\} \{c\} \{P|Q_R\}} \quad \frac{\Gamma \vdash \{P\} c cs \{Q|Q_R\}}{\Gamma \vdash \{Q[e/x]\} x = e \{Q|Q_R\}} \quad \frac{\Gamma \vdash \{P \ \&\& \ b\} c \{P|Q_R\}}{\Gamma \vdash \{P\} \text{while } (b) \ c \{P \ \&\& \ !b|Q_R\}}$$

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

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



## Erweiterter Floyd-Hoare-Kalkül II

$$\Gamma \vdash \{Q\} \text{return } \{P|Q\} \quad \Gamma \vdash \{Q[e/\backslash\text{result}]\} \text{return } e \{P|Q\}$$

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

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



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

$$\bigwedge \text{wvc}(\Gamma, c, Q, Q_R) \implies \Gamma \models \{\text{awp}(c, Q, Q_R)\} c \{Q \mid Q_R\}$$

- Berechnung von  $\text{awp}$  und  $\text{wvc}$ :

$$\begin{aligned} \text{awp}(\Gamma, f(x_1, \dots, x_n) / ** \text{pre } P \text{ post } Q * / \{ds \text{ blk}\}) &\stackrel{\text{def}}{=} \text{awp}(\Gamma', \text{blk}, Q, Q) \\ \text{wvc}(\Gamma, f(x_1, \dots, x_n) / ** \text{pre } P \text{ post } Q * / \{ds \text{ blk}\}) &\stackrel{\text{def}}{=} \\ \{P \implies \text{awp}(\Gamma', \text{blk}, Q, Q)[y_j / \backslash \text{old}(y_j)]\} \cup \text{wvc}(\Gamma', \text{blk}, Q, Q) & \\ \Gamma' &\stackrel{\text{def}}{=} \Gamma[f \mapsto \forall x_1, \dots, x_n. (P, Q)] \end{aligned}$$



## Approximative schwächste Vorbedingung (Revisited)

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



## Approximative Verifikationsbedingungen (Revisited)

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



## 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

