

Korrekte Software: Grundlagen und Methoden

Vorlesung 11 vom 18.06.19

Spezifikation von Funktionen

Serge Autexier, Christoph L  th

Universit  t Bremen

Sommersemester 2019

11.27.30 2019-07-04

1 [28]



Fahrplan

- Einf  hrung
- Operationale Semantik
- Denotationale Semantik
-   quivalenz der Operationalen und Denotationalen Semantik
- Der Floyd-Hoare-Kalk  l
- Invarianten und die Korrektheit des Floyd-Hoare-Kalk  ls
- Strukturierte Datentypen
- Verifikationsbedingungen
- Vorw  rts mit Floyd und Hoare
- Modellierung
- Spezifikation von Funktionen
- Referenzen und Speichermodelle
- Funktionsaufrufe und das Framing-Problem
- Ausblick und R  ckblick

Korrekte Software

2 [28]



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?

Korrekte Software

3 [28]



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 f  r Funktionsdefinitionen
- 5 Semantik des Funktionsaufrufs
- 6 Beweisregeln f  r Funktionsaufrufe

Korrekte Software

4 [28]



Von Anweisungen zu Funktionen

- Erweiterung unserer Kernsprache um Funktionsdefinition und Deklarationen:

```
FunDef ::= FunHeader FunSpec+ Blk
FunHeader ::= Type Idt(Decl*)
Decl ::= Type Idt
Blk ::= {Decl* Stmt}
Type ::= char | int | Struct | Array
Struct ::= struct Idt? {Decl+}
Array ::= Type Idt[Aexp]
```

- Abstrakte Syntax
- Gr   e von Feldern: **konstanter** Ausdruck
- **FunSpec** wird sp  ter erl  utert

Korrekte Software

5 [28]



R  ckgaben

Neue Anweisungen: Return-Anweisung

```
Stmt      s ::= l = e | c1; c2 | { } | if (b) c1 else c2
           | while (b) /** inv a */ c | /** {a} */
           | return a?
```

Korrekte Software

6 [28]



R  ckgabewerte

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

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

- L  sung 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 m  glich, un  bersichtlicher Code ...
- L  sung 2: Erweiterung der Semantik von $\Sigma \rightarrow \Sigma$ zu $\Sigma \rightarrow (\Sigma + \Sigma \times \mathbf{V})$

Korrekte Software

7 [28]



Erweiterte Semantik

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

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

Korrekte Software

8 [28]



Semantik von Anweisungen

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

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

$$\mathcal{C}[c_1; c_2] = \mathcal{C}[c_2] \circ_S \mathcal{C}[c_1] \quad \text{Komposition wie oben}$$

$$\mathcal{C}[\{\cdot\}] = \text{Id}_\Sigma \quad \text{Id}_\Sigma := \{(\sigma, \sigma) \mid \sigma \in \Sigma\}$$

$$\mathcal{C}[\text{if } (b) \ c_0 \ \text{else} \ c_1] = \{(\sigma, \rho') \mid (\sigma, \text{true}) \in \mathcal{B}[b] \wedge (\sigma, \rho') \in \mathcal{C}[c_0]\} \\ \cup \{(\sigma, \rho') \mid (\sigma, \text{false}) \in \mathcal{B}[b] \wedge (\sigma, \rho') \in \mathcal{C}[c_1]\} \\ \text{mit } \rho' \in \Sigma \cup \Sigma \times \mathbf{V}_U$$

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

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

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

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



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.

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

- 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.

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

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

- Von $\mathcal{C}[\text{stmts}]$ 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:

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

$$\text{Vorbedingung} \quad \text{pre } sp; \quad \Sigma \rightarrow \mathbb{B}$$

$$\text{Nachbedingung} \quad \text{post } sp; \quad \Sigma \times (\Sigma \times \mathbf{V}_U) \rightarrow \mathbb{B}$$

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

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



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 $\mathcal{B}[\cdot]$ und $\mathcal{A}[\cdot]$
 - Ausdrücke können in Vor- oder Nachzustand ausgewertet werden.
 - $\backslash \text{result}$ kann nicht in Funktionen vom Typ **void** auftreten.

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

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

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

...

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

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

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

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



Gültigkeit von Spezifikationen

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

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

- 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

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

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.

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



Erweiterung des Floyd-Hoare-Kalküls: return

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

- 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

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

- Die Parameter x_i werden per Konvention nur als x_i referenziert, aber es ist immer der Wert im **Vorzustand** gemeint (eigentlich $\backslash \text{old}(x_i)$).
- 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
- Sequentielle Nachbedingung von c ist **false**



Zusammenfassung: Erweiterter Floyd-Hoare-Kalkül

$$\begin{aligned} &\frac{}{\Gamma \vdash \{P\} \{ \} \{P|Q_R\}} \quad \frac{\Gamma \vdash \{P\} c_1 \{R|Q_R\} \quad \Gamma \vdash \{R\} c_2 \{Q|Q_R\}}{\Gamma \vdash \{P\} c_1; c_2 \{Q|Q_R\}} \\ &\frac{}{\Gamma \vdash \{Q[e/x]\} x = e \{Q|Q_R\}} \quad \frac{\Gamma \vdash \{P \wedge b\} c \{P|Q_R\}}{\Gamma \vdash \{P\} \text{ while } (b) \ c \{P \wedge \neg b|Q_R\}} \\ &\frac{\Gamma \vdash \{P \wedge b\} c_1 \{Q|Q_R\} \quad \Gamma \vdash \{P \wedge \neg b\} c_2 \{Q|Q_R\}}{\Gamma \vdash \{P\} \text{ if } (b) \ c_1 \ \text{else } c_2 \{Q|Q_R\}} \\ &\frac{P \longrightarrow P' \quad \Gamma \vdash \{P'\} c \{Q'|R'\} \quad Q' \longrightarrow Q \quad R' \longrightarrow R}{\Gamma \vdash \{P\} c \{Q|R\}} \end{aligned}$$



Erweiterter Floyd-Hoare-Kalkül II

$$\frac{}{\Gamma \vdash \{Q\} \text{ return } \{P|Q\}} \quad \frac{}{\Gamma \vdash \{Q[e/\backslash \text{result}]\} \text{ return } e \{P|Q\}} \quad \frac{P \implies P'[y_i/\backslash \text{old}(y_i)] \quad \Gamma \vdash \{P'\} c \{false|Q\}}{\Gamma \vdash f(x_1, \dots, x_n)/** \text{pre } P \text{ post } Q */ \{ds \ c\}}$$



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** Γ und eine **Rückgabespezifikation** Q_R benötigt.
- Es gilt:

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

- Berechnung von **awp** und **wvc**:

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



Approximative schwächste Vorbedingung (Revisited)

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



Approximative Verifikationsbedingungen (Revised)

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



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**

