Korrekte Software: Grundlagen und Methoden
Vorlesung 11 vom 19.06.18: Funktionen und Prozeduren

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2018

Fahrplan

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

(2) Semantik von Funktionsdefinition und Funktionsaufruf

(3) Spezifikation von Funktionen

(4) Beweisregeln für Funktionsdefinition und Funktionsaufruf

Von Anweisungen zu Funktionen

- Erweiterung unserer Kernsprache um Funktionsdefinition und Deklarationen:

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

- Abstrakte Syntax (konkrete Syntax mischt **Type** und **Idt**, Kommata bei Argumenten, ...)
- Größe von Feldern: **konstanter** Ausdruck
- **FunSpec** später

Funktionsaufrufe und Rückgaben

Neue Ausdrücke und Anweisungen:

- Funktionsaufrufe
- Return-Anweisung

$$\begin{aligned}
\textbf{Aexp}\ a ::=\ & \textbf{Z} \mid \textbf{C} \mid \textbf{Lexp} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \mid a_1/a_2\\
& \mid \textbf{Idt}(\textbf{Exp}^*)\\
\textbf{Bexp}\ b ::= & \ \textbf{1} \mid \textbf{0} \mid a_1 == a_2 \mid a_1 < a_2 \mid !\, b \mid b_1\ \&\&\ b_2 \mid b_1 \mid\mid b_2\\
\textbf{Exp}\ e :=\ & \textbf{Aexp} \mid \textbf{Bexp}\\
\textbf{Stmt}\ c ::=\ & l = e \mid c_1; c_2 \mid \{\,\} \mid \textbf{if}\ (b)\ c_1\ \textbf{else}\ c_2\\
& \mid \textbf{while}\ (b)\ /{**}\,\textbf{inv}\ a\,{*}/\ c \mid /{**}\{a\}\,{*}/\\
& \mid \textbf{Idt}(a^*)\\
& \mid \textbf{return}\ a^?
\end{aligned}$$

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: $\Sigma \rightharpoonup (\Sigma + \Sigma \times \mathbf{V})$ $\Sigma \rightharpoonup (\Sigma + \Sigma \times \mathbf{V}_U)$
- 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 $f, g : \Sigma \rightharpoonup (\Sigma + \Sigma \times \mathbf{V}_U)$:

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

Semantik von Anweisungen

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

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

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

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.

$$\begin{aligned}
&\mathcal{D}_{fd}[\![f(t_1\ p_1, t_2\ p_2, \ldots, t_n\ p_n)\ \mathit{blk}]\!] = \\
&\quad \lambda v_1, \ldots, v_n. \{(\sigma, (\sigma', v)) \mid \\
&\qquad (\sigma, (\sigma', v)) \in \mathcal{D}_{blk}[\![\mathit{blk}]\!] \circ_S \{(\sigma, \sigma[v_1/p_1, \ldots, v_n/p_n])\}\}
\end{aligned}$$

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

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:

$$\begin{aligned}
\mathcal{D}_{blk}[\![\mathit{decls}\ \mathit{stmts}]\!] &= \mathcal{C}[\![\mathit{stmts}]\!] \circ_S \mathcal{D}_d[\![\mathit{decls}]\!] \\
\mathcal{D}_d[\![t\ i]\!] &= \{(\sigma, \sigma[\perp/i])\}
\end{aligned}$$

- Verallgemeinerung auf Sequenz von Deklarationen

Funktionsaufrufe

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

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):

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

- Das Environment ist **zusätzlicher Parameter** für alle Definitionen

Semantik von Funktionsaufrufen

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

- 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:

FunSpec ::= /** **pre** **Bexp** **post** **Bexp** */

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); */
  /** 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;
}
```


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.
  - \resultkann nicht in Funktionen vom Typ **void** auftreten.

$$\mathcal{B}_{sp}[\![.]\!] : \mathbf{Env} \to \mathbf{Bexp} \rightharpoonup (\Sigma \times (\Sigma \times \mathbf{V}_U)) \to \mathbb{B}$$
$$\mathcal{A}_{sp}[\![.]\!] : \mathbf{Env} \to \mathbf{Aexp} \rightharpoonup (\Sigma \times (\Sigma \times \mathbf{V}_U)) \to \mathbf{V}$$
$$\mathcal{B}_{sp}[\![!b]\!]\,\Gamma = \{((\sigma,(\sigma',v)),1) \mid ((\sigma,(\sigma',v)),\mathit{false}) \in \mathcal{B}_{sp}[\![b]\!]\Gamma\} \cup \{((\sigma,(\sigma',v)),0) \mid ((\sigma,(\sigma',v)),\mathit{true}) \in \mathcal{B}_{sp}[\![b]\!]\Gamma\}$$
$$\ldots$$
$$\mathcal{B}_{sp}[\![\backslash\mathbf{old}(e)]\!]\,\Gamma = \{((\sigma,(\sigma',v)),b) \mid (\sigma,b) \in \mathcal{B}[\![e]\!]\,\Gamma\}$$
$$\mathcal{A}_{sp}[\![\backslash\mathbf{old}(e)]\!]\,\Gamma = \{((\sigma,(\sigma',v)),a) \mid (\sigma,a) \in \mathcal{A}[\![e]\!]\,\Gamma\}$$
$$\mathcal{A}_{sp}[\![\backslash\mathbf{result}]\!]\,\Gamma = \{((\sigma,(\sigma,v)),v)\}$$

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

Gültigkeit von Spezifikationen

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

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

  - 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.

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

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

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

- 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

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

- 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

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

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

- 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 $\backslash\mathbf{old}(y_j)$ 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 \result

Erweiterter Floyd-Hoare-Kalkül I

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

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

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

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

Erweiterter Floyd-Hoare-Kalkül II

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

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

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

Beispiel: die Fakultätsfunktion, rekursiv

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

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


Beobachtungen

- Der Aufruf einer Funktion **ersetzt** die momentane Nachbedingung — das ist ein Problem
- Wir brauchen keine Invariante mehr — ist durch die Nachbedingung gegeben
- Rekursion benötigt keine Extrabehandlung
  - Termination von rekursiven Funktionen wird extra gezeigt

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