Korrekte Software: Grundlagen und Methoden
Vorlesung 11 vom 06.06.16: Funktionen und Prozeduren

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2016

Fahrplan

- Einführung
- Die Floyd-Hoare-Logik
- Operationale Semantik
- Denotationale Semantik
- Äquivalenz der Semantiken
- Verifikation: Vorwärts oder Rückwärts?
- Korrektheit des Hoare-Kalküls
- Einführung in Isabelle/HOL
- Weitere Datentypen: Strukturen und Felder
- Funktionen und Prozeduren
- Referenzen und Zeiger
- Frame Conditions & Modification Clauses
- 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)
- 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

Motivation

- Funktionen sind zentrales Modularisierungskonzept
- Wir müssen Funktionen modular verifizieren können
- Semantik von Deklarationen und Parameter — straightforward
- Semantik von Rückgabewerten — Erweiterung der Semantik
- Funktionsaufrufe — Environment, um Funktionsbezeichnern eine Semantik zu geben
  - C kennt nur call by value
- Spezifikation von Funktionen: Vor-/Nachzustand statt logischer Variablen

Hoare-Kalkül für Funktionspezifikationen

$$\begin{aligned}
\textbf{FunDef} &::= \textbf{Type}\ \mathit{Id}(\textbf{Param}^*)\ \textbf{FunSpec}^+\ \textbf{Blk}\\
\textbf{Param} &::= \textbf{Type}\ \mathit{Id}\\
\textbf{FunSpec} &::= \texttt{/**}\ \textbf{pre}\ \textbf{Bexpr}\ \textbf{post}\ \textbf{Bexpr}\ \texttt{*/}\\
\textbf{Blk} &::= \{\textbf{Decl}^*\ \textbf{Stmt}\}\\
\textbf{Decl} &::= \textbf{Type}\ \mathit{Id}\ = \textbf{Init} \mid \textbf{Type}\ \mathit{Id}
\end{aligned}$$

- Hoare-Tripel:

$$\{P\}\ c\ \{Q_L \mid Q_G\}$$

Beispiel

```
int factorial(int x)
/** pre x >= 0
    post \result = \old(x)! */ {
 int r = 0;
 if (x = 0) {
    return 1;
 }
 else {
    r = factorial(x - 1);
 }
 return r * x;
}
```

$$\frac{P \longrightarrow P'_{\backslash old(\$)\to\$} \qquad \{P'\}c\{Q|Q\}}{\tau_0\ f(\tau_1\ v_1,\ldots,\tau_n\ v_n)\ \texttt{/**}\ \text{pre}\ P\ \text{post}\ Q\ \texttt{*/}\ c\ \mathit{int\ factorial(int\ x)}\ \texttt{/**}\ \text{pre}\ x >= 0\ \text{pos}} \qquad \frac{x >= 0 \longrightarrow P'_{\backslash old(}}{\{P'\}c\{\backslash result = \backslash old(x)! \mid \backslash re}$$

Hoare-Kalkül mit return

$$\frac{}{\{P\}\{\}\{P|Q\}}$$

$$\frac{\{P\}c\{Q_1'|Q_2\} \qquad \{Q_1'\}cs\{Q_1|Q_2\}}{\{P\}c\ cs\{Q_1|Q_2\}}$$

$$\frac{}{\{Q_1[e/l]\}l = e\{Q_1|Q_2\}}$$

$$\frac{\{P \wedge b\}c_1\{Q_1|Q_2\} \qquad \{P \wedge \neg b\}c_2\{Q_1|Q_2\}}{\{P\}\text{if } b\ c_1 \text{ else } c_2\{Q_1|Q_2\}}$$

While, Weakening

$$\frac{\{P \wedge b\}c\{P|Q\}}{\{P\}\text{while } (b)\ c\{P \wedge \neg(b)|Q\}}$$

$$\frac{P \longrightarrow P' \qquad \{P'\}c\{Q_1'|Q_2'\} \qquad Q_1' \longrightarrow Q_1 \qquad Q_2' \longrightarrow Q_2}{\{P\}c\{Q_1|Q_2\}}$$

Hoare-Kalkül mit return

$$\frac{}{\{Q[^{e}/_{\backslash result}]\}\,return\ e\,\{P|Q\}}$$

$$\frac{Q \text{ enthält kein } \backslash result}{\{Q\}return\{P|Q\}}$$

Funktionsaufruf

$$\frac{\begin{array}{c}(V_1 = e_1 \wedge \ldots V_n = e_n \wedge Q_1) \longrightarrow f.pre(e_1, \ldots e_n) \\ P := f.post(e_1, \ldots, e_n)_{\backslash result \to l, \backslash old(v_i) \to V_i} \\ \{(V_1 = e_1 \wedge \ldots V_n = e_n \wedge Q_1 \wedge P)[f(e_1, \ldots, e_n)/l]\} \\ l = f(e_1, \ldots, e_n) \\ \{V_1 = e_1 \wedge \ldots V_n = e_n \wedge Q_1 \wedge P | Q_2\}\end{array}}{}$$

f mit formalen Parametern $v_1, \ldots, v_n$; $V_1, \ldots V_n$ logische Variablen


```
int factorial(int x)
/** pre x >= 0
    post \result = \old(x)! */ {
/** { COND: x >= 0 ——> x >= 0 ∧ x! = x!
      x >= 0 ∧ x! = \old(x)! } */
 int r = 0;
/** {x >= 0 ∧ x! = \old(x)!} */
 if (x = 0) {
/** {x >= 0 ∧ x = 0 ∧ x! = \old(x)!} */
/** {x >= 0 ∧ x = 0 ∧ 1 = \old(x)!} */
    return 1;
/** { r * x = \old(i)! |  \result = \old(x)!} */
 }
 else { ... }
/** { r * x = \old(i)! | \result =\old(i)! } */
 return r * x;
/** {\result =\old(i)! | \result =\old(i)!} */
```



```
 else {
/** {x >= 0 ∧ ¬(x = 0) ∧ x! = \old(x)!} */
/** {COND: x >= 0 ∧ ¬(x = 0) ∧
          factorial(x−1) * x = \old(x)! ∧ V = x —> x − 1 >= 0
      x >= 0 ∧ ¬(x = 0) ∧ factorial(x−1) * x = \old(x)! ∧
      V = x ∧ factorial(x−1) = (V−1)!} */
    r = factorial(x − 1);
/** {x >= 0 ∧ ¬(x = 0) ∧ r * x = \old(x)! ∧
     V = x ∧ r = (V−1)!} */
/** { x >= 0 ∧ ¬(x = 0) ∧ r * x = \old(x)! } */
/** { r * x = \old(i)! } */
 }
/** { r * x = \old(i)! | \result =\old(i)! } */
 return r * x;
/** {\result = \old(i)! | \result =\old(i)!} */
}
```


Approximative schwächste Vorbedingung (Revisited)

$$\begin{array}{rcl}
\text{awp}(\Gamma, \{\,\}, P) & \stackrel{def}{=} & P \\
\text{awp}(\Gamma, l = f(e_1, \ldots, e_n), P) & \stackrel{def}{=} & P[F([\![e_1]\!], \ldots, [\![e_n]\!])/[\![l]\!]] \\
\text{mit } post(\Gamma!f) = (\forall v_1, \ldots, v_n.\textbf{result} = F(v_1, \ldots, v_n)) & & \\
\text{awp}(\Gamma, l = e, P) & \stackrel{def}{=} & P[[\![e]\!]/[\![l]\!]] \\
\text{awp}(\Gamma, \{c\ c_s\}, P) & \stackrel{def}{=} & \text{awp}(\Gamma, c, \text{awp}(\{c_s\}, P)) \\
\text{awp}(\Gamma, \textbf{if } (b)\ \{c_0\}\ \textbf{else}\ \{c_1\}, P) & \stackrel{def}{=} & (b \wedge \text{awp}(\Gamma, c_0, P)) \\
 & & \vee(\neg b \wedge \text{awp}(\Gamma, c_1, P)) \\
\text{awp}(\Gamma, /{*}{*}\ \{q\}\ {*}/, P) & \stackrel{def}{=} & [\![q]\!] \\
\text{awp}(\Gamma, \textbf{while } (b)\ /{*}{*}\ \textbf{inv}\ i\ {*}/\ c, P) & \stackrel{def}{=} & [\![i]\!] \\
\text{awp}(\Gamma, \textbf{return}\ e, P) & \stackrel{def}{=} & post(\Gamma)[[\![e]\!]/\textbf{result}] \\
\text{awp}(\Gamma, \textbf{return}, P) & \stackrel{def}{=} & post(\Gamma)
\end{array}$$

Approximative schwächste Vorbedingung (Revisited)

$$\begin{array}{rcl}
\text{wvc}(\Gamma, \{\,\}, P) & \stackrel{def}{=} & \emptyset \\
\text{wvc}(\Gamma, x = e, P) & \stackrel{def}{=} & \emptyset \\
\text{wvc}(\Gamma, x = f(e_1, \ldots, e_n), P) & \stackrel{def}{=} & P \longrightarrow pre(\Gamma!f)([\![e_1]\!], \ldots, [\![e_n]\!]) \\
\text{wvc}(\Gamma, \{c\ c_s\}, P) & \stackrel{def}{=} & \text{wvc}(\Gamma, c, \text{awp}(\{c_s\}, P)) \\
 & & \cup\, \text{wvc}(\Gamma, \{c_s\},, P) \\
\text{wvc}(\Gamma, \textbf{if } (b)\ c_0\ \textbf{else}\ c_1, P) & \stackrel{def}{=} & \text{wvc}(\Gamma, c_0, P) \cup \text{wvc}(\Gamma, c_1, P) \\
\text{wvc}(\Gamma, /{*}{*}\ \{q\}\ {*}/, P) & \stackrel{def}{=} & \{[\![q]\!] \longrightarrow P\} \\
\text{wvc}(\Gamma, \textbf{while } (b)\ /{*}{*}\ \textbf{inv}\ i\ {*}/\ c, P) & \stackrel{def}{=} & \text{wvc}(\Gamma, c, [\![i]\!]) \\
 & & \cup \{[\![i]\!] \wedge b \longrightarrow \text{awp}(\Gamma, c, [\![i]\!])\} \\
 & & \cup \{[\![i]\!] \wedge \neg b \longrightarrow P\} \\
\text{wvc}(\Gamma, \textbf{return}\ e, P) & \stackrel{def}{=} & \emptyset
\end{array}$$