Korrekte Software: Grundlagen und Methoden
Vorlesung 13 vom 02.07.19
Funktionsaufrufe und das Framing-Problem

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2019

Fahrplan

Modellierung und Spezifikation von Funktionen

Wir brauchen:

1. Deklarationen und Parameter ✓
2. Semantik von Funktionsdefinitionen ✓
3. Spezifikation von Funktionsdefinitionen ✓
4. Beweisregeln für Funktionsdefinitionen ✓
5. Semantik des Funktionsaufrufs
6. Beweisregeln für Funktionsaufrufe

Funktionsaufrufe und Rückgaben

Neue Ausdrücke und Anweisungen:

- Funktionsaufrufe
- Prozeduraufrufe (mit Zuweisung eines Rückgabewertes)

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

Zur Erinnerung: 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.

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

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

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**?
    - Durch Übergabe von **Referenzen** als **Werten**

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):

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

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

Nebenbedingungen von Funktionsaufrufen

- Aufruf einer nicht-definierten Funktion f oder mit falscher Anzahl n von Parametern ist nicht definiert
  - Muss durch **statische Analyse** verhindert werden
- Aufruf einer Funktion mit **Seiteneffekt** in einem größeren Ausdruck
  - Reihenfolge der Seiteneffekte unspezifiziert
  - Daher hier nur für reine Funktionen
- **Reine Funktion** (pure function):
  - keine (sichtbaren) Seiteneffekte und Spezifikation der Form $\backslash$ result $=$ ...

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 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

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.
- Korrektheit gilt immer nur im **Kontext**, dadurch kann jede Funktion separat verifiziert werden (**Modularität**)

Erweiterung des Floyd-Hoare-Kalküls: Aufruf

$$\frac{\Gamma(f) = \forall x_1,\ldots,x_n.\,(P,Q),\ f \text{ vom Typ } \textbf{void}}{\Gamma \vdash \{\tau = \sigma \wedge P[t_i{}^{\#}/x_i]\}\ f(t_1,\ldots,t_n)\ \{Q[\tau/\rho][t_i{}^{\#}/x_i] \mid Q_R\}}$$

- Γ muss f mit der Vor-/Nachbedingung P, Q enthalten
- In P und Q werden Parameter x_i durch Argumente t_i ersetzt.
- In Q wird der Vorzustand ρ durch τ ersetzt
- Q darf kein **\result** enthalten

Erweiterung des Floyd-Hoare-Kalküls: Aufruf

$$\frac{\Gamma(f) = \forall x_1,\ldots,x_n.\,(P,Q)}{\Gamma \vdash \{\tau = \sigma \wedge P[t_i{}^{\#}/x_i]\}\ I = f(t_1,\ldots,t_n)\ \{Q[\tau/\rho][t_i{}^{\#}/x_i][I^{\#}/\backslash\textbf{result}] \mid Q_R\}}$$

- Γ muss f mit der Vor-/Nachbedingung P, Q enthalten
- In P und Q werden Parameter x_i durch Argumente t_i ersetzt
- In Q wird der Vorzustand ρ durch τ ersetzt
- **\result** in Q wird durch I ersetzt

Zwei Beispiele

```
void incr(int *x)
    /** post \old(*x)+1 == *x  */

int a;

// x == 4
incr(&x);
// x == 5

void swap (int *x, int *y)
    /** post \old(*y) == *x && \old(*x) == *y */

int a, b;

a= 3;
b= 5;
swap(&a, &b);
// b == 3 && a == 5
```


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;
}
```


Beobachtung

- 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

Funktionsparameter und Frame Conditions

- Problem: Funktionen können **beliebige** Änderungen im Speicher vornehmen.

```
int x, y, z;

z= x+ y;
swap(&x, &y);
/** { z == \old(x)+ \old(y) } */
```

- Vor/Nach dem Funktionsaufruf (hier swap) muss die Nachbedingung/Vorbedingung noch gelten.

Frame Rule

- Konstanzregel (Rule of Constancy):

$$\frac{\vdash \{P\}\, c\, \{Q\}}{\vdash \{P \wedge R\}\, c\, \{Q \wedge R\}}$$

- Nebenbedingung: c verändert keine Variablen in R
- Problem: mit Pointern können wir (syntaktisch) keine Aussagen über den Teil des Zustandes machen, den c verändert macht (**framing problem**)

Modification Sets

- Idee: Spezifiziere, welcher Teil des Zustands verändert werden darf.
  - ... denn wir können **nicht** spezifizieren, was gleich bleibt.
- Syntax: modifies **Mexp**

$$\textbf{Mexp} ::= \textbf{Loc} \mid \textbf{Mexp}\ [*] \mid \textbf{Mexp}\ [i:j] \mid \textbf{Mexp}\ .\ \textbf{name}$$

  - Mexp sind Lexp, die auch **Teile** von Feldern bezeichnen.
- Semantik: $[\![-]\!] : \textbf{Env} \to \textbf{Mexp} \to \Sigma \to \mathbb{P}(\textbf{Loc})$
- Modification Sets werden in die Hoare-Tripel **integriert**.

Semantik mit Modification Sets

- Hoare-Tripel mit Modification Sets:

$$\Lambda \models \{P\}\, c\, \{Q\} \iff \forall \sigma.\, P(\sigma) \wedge \exists \sigma'.\, \sigma' = c(\sigma) \Longrightarrow Q(\sigma') \wedge \sigma \cong_\Lambda \sigma'$$

- wobei $\sigma \cong_L \tau \iff \forall l \in dom(\sigma) \cup dom(\tau) \setminus L.\, \sigma(l) = \tau(l)$
- oder alternativ $\sigma \cong_L \tau \iff \forall l.\, \sigma(l) \neq \tau(l) \Longrightarrow l \in L$

Regeln mit Modification Sets

- Regeln werden mit Modification Set annotiert:

$$\Gamma, \Lambda \vdash \{P\}\, c\, \{Q_1 | Q_2\}$$

- Modification Set wird durchgereicht, aber:

$$\frac{\begin{array}{l}\Gamma, \Lambda \vdash \{\lambda\sigma.\, [\![l]\!]\Gamma \in dom(\sigma) \wedge [\![l]\!]\Gamma \in \Lambda \wedge Q(upd(\sigma, [\![l]\!]\Gamma, [\![e]\!]\Gamma))\} \\ \quad l = e \\ \quad \{Q | R\}\end{array}}{}$$

Zusammenfassung

- Aufruf von Funktionen:
  - Funktionen mit Seiteneffekt in Kombination mit Zuweisung
- Aufruf einer Funktion **ersetzt** Vor/Nachbedingung
- **Framing-Problem**: Beschränkung der Zustandsänderung durch eine Funktion
- Erfordert weitere Erweiterung der Sprache (modification sets)
- Fazit: Funktionen sind nicht ganz so straightforward