Korrekte Software: Grundlagen und Methoden
Vorlesung 10 vom 30.05.16: 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

Von Anweisungen zu Funktionen

- Erweiterung unserer Kernsprache:

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

- **Type**, **Init** (Initialisierer) s. letzte Vorlesung
- **FunSpec** später
- Abstrakte Syntax (vereinfacht, konkrete Syntax mischt **Type** und *Id*)

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

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

$$\begin{aligned}\mathcal{D}[\![x = e]\!] &= \{(\sigma, \sigma(c \mapsto a)) \mid (\sigma, c) \in \mathcal{L}[\![x]\!], (\sigma, a) \in \mathcal{E}[\![e]\!]\}\\ \mathcal{D}[\![\{c\ c_s\}]\!] &= \mathcal{D}[\![c_s]\!] \circ_S \mathcal{D}[\![c]\!] \quad \text{Komposition wie oben}\\ \mathcal{D}[\![\{\ \}]\!] &= \mathbf{Id} \qquad\qquad \mathbf{Id} := \{(\sigma, \sigma) | \sigma \in \Sigma\}\\ \mathcal{D}[\![\textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1]\!] &= \{(\sigma, \tau) \mid (\sigma, 1) \in \mathcal{B}[\![b]\!] \wedge (\sigma, \tau) \in \mathcal{D}[\![c_0]\!]\}\\ &\quad \cup \{(\sigma, \tau) \mid (\sigma, 0) \in \mathcal{B}[\![b]\!] \wedge (\sigma, \tau) \in \mathcal{D}[\![c_1]\!]\}\\ &\quad \text{mit } \tau \in \Sigma \cup (\Sigma \times \mathbf{V}_U)\\ \mathcal{D}[\![\textbf{return}\ e]\!] &= \{(\sigma, (\sigma, a)) \mid (\sigma, a) \in \mathcal{E}[\![e]\!]\}\\ \mathcal{D}[\![\textbf{return}]\!] &= \{(\sigma, (\sigma, *))\}\end{aligned}$$

$$\begin{aligned}\mathcal{D}[\![\textbf{while}\ (b)\ c]\!] &= \mathit{fix}(\Gamma)\\ \Gamma(\psi) &\stackrel{def}{=} \{(\sigma, \tau) \mid (\sigma, 1) \in \mathcal{B}[\![b]\!] \wedge (\sigma, \tau) \in \psi \circ_S \mathcal{D}[\![c]\!]\}\\ &\quad \cup \{(\sigma, \sigma) \mid (\sigma, 0) \in \mathcal{B}[\![b]\!]\}\end{aligned}$$

Semantik von Funktionsdefinitionen

$$\mathcal{D}_{fd}[\![.]\!] : \textbf{FunDef} \to \mathbf{V}^n \rightharpoonup \Sigma \rightharpoonup \Sigma \times \mathbf{V}_U$$

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

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

- 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.

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:

$$\mathcal{D}_{blk}[\![decls\ stmts]\!] = \mathcal{D}[\![stmts]\!] \circ_S \mathcal{D}_d[\![decls]\!]$$
$$\mathcal{D}_d[\![t\ i]\!] = \{(\sigma, \sigma[i \mapsto \bot])\}$$
$$\mathcal{D}_d[\![t\ i\ = init]\!] = \{(\sigma, \sigma[i \mapsto \mathcal{E}_{init}[\![init]\!]\}$$

- Verallgemeinerung auf Sequenz von Deklarationen
- $\mathcal{E}_{init}[\![.]\!]$ ist das Denotat von Initialisierungen

Funktionsaufrufe

- Aufruf einer Funktion: $f(t_1, \ldots, t_n)$:
  - Auswertung der Argumente $t_1, \ldots, t_n$
  - Einsetzen in die Semantik $\mathcal{D}_{fd}[\![f]\!]$
- Was ist mit Seiteneffekten?
  - Erst mal gar nichts...
- Call by name, call by value, call by reference...?
  - C kennt nur call by value (C-Standard 99, §6.9.1. (10))
  - Arrays werden als Referenzen übergeben (deshalb betrachten wir heute keine Arrays als Funktionsparameter).

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):

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

- Das Environment ist zusätzlicher Parameter für alle Definitionen
- Damit:

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

  - Aufruf einer nicht-definierten Funktion f oder mit falschen Anzahl n von Parametern ist nicht definiert
  - Wird durch statische Analyse verhindert

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;	$\Sigma \to \mathbf{T}$
Nachbedingung	**post** sp;	$\Sigma \times (\Sigma \times \mathbf{V}_U) \to \mathbf{T}$
	\old(e)	Wert von e im Vorzustand
	\result	Rückgabewert der Funktion

Semantik von Spezifikationen

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

$$\mathcal{B}_{sp}[\![.]\!] : Env \rightharpoonup (\Sigma \times (\Sigma \times \mathbf{V}_U)) \to \mathbf{T}$$
$$\mathcal{E}_{sp}[\![.]\!] : Env \rightharpoonup (\Sigma \times (\Sigma \times \mathbf{V}_U)) \to \mathbf{V}$$
$$\mathcal{B}_{sp}[\![!b]\!]\,\Gamma = \{(((\sigma, (\sigma', v)), 1) \mid (((\sigma, (\sigma', v)), 0) \in \mathcal{B}_{sp}[\![b]\!]\Gamma\} \cup \{(((\sigma, (\sigma', v)), 0) \mid (((\sigma, (\sigma', v)), 1) \in \mathcal{B}_{sp}[\![b]\!]\Gamma\}$$
$$\ldots$$
$$\mathcal{B}_{sp}[\![\backslash\mathbf{old}(e)]\!]\,\Gamma = \{(((\sigma, (\sigma', v)), b) \mid (\sigma, b) \in \mathcal{B}[\![e]\!]\,\Gamma\}$$
$$\mathcal{E}_{sp}[\![\backslash\mathbf{old}(e)]\!]\,\Gamma = \{(((\sigma, (\sigma', v)), a) \mid (\sigma, a) \in \mathcal{E}[\![e]\!]\,\Gamma\}$$
$$\mathcal{E}_{sp}[\![\backslash\mathbf{result}]\!]\,\Gamma = \{(((\sigma, (\sigma, v)), v)\}$$

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

Gültigkeit von Spezifikationen

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

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

  - Γ enthält globale Definitionen, insbesondere andere Funktionen.
  - Vgl. $\models \{P\}\, c\, \{Q\}$ für Hoare-Tripel
- Aber wie beweisen wir das? $\longrightarrow$ Nächste Vorlesung
  - Die Grenzen des Hoare-Kalküls sind erreicht.

Zusammenfassung

- 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