Korrekte Software: Grundlagen und Methoden
Vorlesung 1 vom 02.04.19
Einführung

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2019

Organisatorisches

- Veranstalter:

Christoph Lüth
christoph.lueth@dfki.de
MZH 4186, Tel. 59830

Serge Autexier
serge.autexier@dfki.de
Cartesium 1.49, Tel. 59834

- Termine:
  - Vorlesung: Dienstag, 12 – 14, MZH 1100
  - Übung: Donnerstag, 12 – 14, MZH 1450
- Webseite:

  `http://www.informatik.uni-bremen.de/~cxl/lehre/ksgm.ss19`

Übungsbetrieb

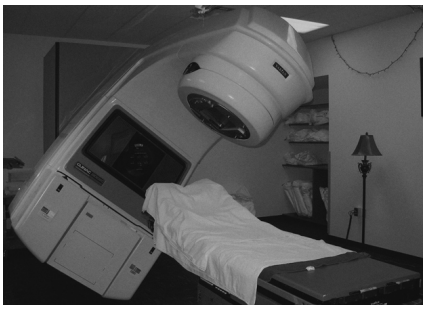
- "Leichtgewichtige" Übungsblätter, die **in der Übung** bearbeitet und **schnell** korrigiert werden können.
- Übungsblätter **vertiefen** Vorlesungsstoff, Bewertung gibt Feedback.
- Übungsbetrieb:
  - Gruppen bis zu drei Studierende
  - Ausgabe: Donnerstag in der Übung
  - Bearbeitung: in der Übung
  - Abgabe: Donnerstag abend

Prüfungsform und Übungsbetrieb

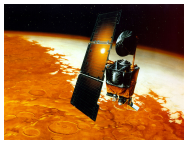
- 10 Übungsblätter (geplant)
- Bewertung:
  - A (sehr gut, 1.3) — nichts zu meckern, keine/kaum Fehler
  - B (gut, 2.3) — kleine Fehler, sonst gut
  - C (befriedigend, 3.3) — größere Fehler oder Mängel
  - Nicht bearbeitet — oder zu viele Fehler
- Prüfungsleistung:
  - Mündliche Prüfung
    - Einzelprüfung ca. 20– 30 Minuten
  - Übungsbetrieb (bis zu 20% Bonuspunkte, keine Voraussetzung)

Warum Korrekte Software?

Software-Disaster I: Therac-25



Software-Disasters II: Space



Mariner 1 (27.08.1962), Mars Climate Orbiter (1999), Ariane 5 (04.06.1996)

Software-Disaster III: AT&T (15.01.1990)

```
while (! empty(ring_rcv_buffer)
       && ! empty(side_buffer empty)) {
 initialize pointer to first message buffer;
 get copy of buffer;
 switch (message) {
   case (incoming_message):
     if (sender is out_of_service) {
       if (empty(ring_wrt_buffer)) {
         send "in service" to status map;
       } else {
         break;
       }
       process incoming message, set up pointers;
       break;
       }
   }
 do optional parameter work;
 }
```

Software-Disaster IV: Airbus A400M

Sevilla, 09.05.2015

Inhalt der Vorlesung

Themen

Korrekte Software im Lehrbuch:

- Spielzeugsprache
- Wenig Konstrukte
- Kleine Beispiele

Korrekte Software im Einsatz:

- Richtige Programmiersprache
- Mehr als nur ganze Zahlen
- Skalierbarkeit — wie können große Programme verifiziert werden?

Inhalt

- Grundlagen:
  - Beweis der **Korrektheit** von Programmen: der **Floyd-Hoare-Kalkül**
  - **Bedeutung** von Programmen: **Semantik**
- Betrachtete Programmiersprache: "C0" (erweiterte Untermenge von C)
- Erweiterung der Programmkonstrukte und des Hoare-Kalküls:
  1. Referenzen (Zeiger)
  2. Funktion und Prozeduren (Modularität)
  3. Reiche **Datenstrukturen** (Felder, `struct`)

Fahrplan

- Einführung
- Operationale Semantik
- Denotationale Semantik
- Äquivalenz der Operationalen und Denotationalen Semantik
- Der Floyd-Hoare-Kalkül
- Invarianten und die Korrektheit des Floyd-Hoare-Kalküls
- Strukturierte Datentypen
- Verifikationsbedingungen
- Vorwärts mit Floyd und Hoare
- Modellierung
- Spezifikation von Funktionen
- Referenzen und Speichermodelle
- Funktionsaufrufe und das Framing-Problem
- Ausblick und Rückblick

Warum Semantik?

Idee

- Was wird hier berechnet? $p = n!$
- Warum? Wie können wir das **beweisen**?
- Wir berechnen symbolisch, welche Werte Variablen über den Programmverlauf annehmen.

```
p= 1;
c= 1;
while (c <= n) {
  p = p * c;
  c = c + 1;
}
```


Semantik von Programmiersprachen

Drei wesentliche Möglichkeiten:

- **Operationale Semantik**:
  Ausführung auf einer **abstrakten** Maschine
- **Denotationale Semantik**:
  Abbildung in ein **mathematisches Objekt**
- **Axiomatische Semantik**:
  Beschreibung durch eines Programmes durch seine **Eigenschaften**

Unsere Sprache C0

- C0 ist eine **Untermenge** der Sprache C
- C0-Programme sind **ausführbare** C-Programme
- Erste Ausbaustufe:
  - Zuweisungen, Fallunterscheidungen, Schleifen
  - Datentypen: ganze Zahlen mit Arithmetik
  - Relationen: Vergleich ($=$, $\leq$)
  - Boolsche Operatoren: Konjunktion, Disjunktion, Negation
- 1. Ausbaustufe: Felder und Strukturen
- 2. Ausbaustufe: Funktionen und Prozeduren (nur Ausblick)
- 3. Ausbaustufe: Referenzen (nur Ausblick)
- Fehlt: **union**, **goto**, ...

Operationale Semantik

- Kernkonzept: Zustandstandsübergänge einer abstrakten Maschine
- Abstrakte Maschine hat **impliziten Zustand**
- Zustand ordnet **Adressen** veränderliche **Werte** zu
- Konkretes Beispiel: $n \mapsto 3$, p und c undefiniert

```
p = 1;
c = 1;
while(c <= n){
  p = p * c;
  c = c + 1; }
```

p	?
c	?
n	3

$\rightsquigarrow$

p	1
c	?
n	3

$\rightsquigarrow$

p	1
c	1
n	3

$\rightsquigarrow$

p	1
c	1
n	3

$\rightsquigarrow$

p	1
c	2
n	3

$\rightsquigarrow$

p	2
c	2
n	3

$\rightsquigarrow$

p	2
c	3
n	3

$\rightsquigarrow$

p	6
c	3
n	3

$\rightsquigarrow$

p	6
c	4
n	3

Denotationale Semantik

- Kernkonzept: Abbildung von Programmen auf mathematisches Gegenstück (**Denotat**)
- **Partielle** Funktionen zwischen Zuständen $[\![c]\!] : \sigma \rightharpoonup \sigma$
- Beispiel:

```
p = 1;
c = 1; // p1
while (c <= n) {
  p = p * c;
  c = c + 1; // p2
  }
// p3
```

$$[\![p_1]\!](\sigma) = \sigma(p \mapsto 1)(c \mapsto 1)$$

$$[\![p_2]\!](\sigma) = \sigma(p \mapsto \sigma(p) * \sigma(c))(c \mapsto \sigma(c) + 1)$$

$$[\![p_3]\!](\sigma)[\![p_3]\!] = ??? \; fix(\Gamma([\![c \leq n]\!])[\![p_2]\!]))([\![p_1]\!](\sigma)) \; fix(\Gamma([\![c \leq n]\!])([\![p_2]\!])) \circ [\![p_1]\!]$$

$$\Gamma([\![c \leq n]\!])([\![p_2]\!])(\varphi)(\sigma) = \begin{cases} \sigma & \text{if } [\![c \leq n]\!](\sigma) = 0 \\ (\varphi \circ [\![p_2]\!])(\sigma) & \text{if } [\![c \leq n]\!](\sigma) = 1 \end{cases}$$

$$\Gamma(\beta)(\rho)(\varphi)(\sigma) = \begin{cases} \sigma & \text{if } \beta(\sigma) = 0 \\ (\varphi \circ \rho)(\sigma) & \text{if } \beta(\sigma) = 1 \end{cases}$$

Axiomatische Semantik

- Kernkonzept: Charakterisierung von Programmen durch **Zusicherungen**
- Zusicherungen sind zustandsabhängige Prädikate
- Beispiel (mit $n = 3$)

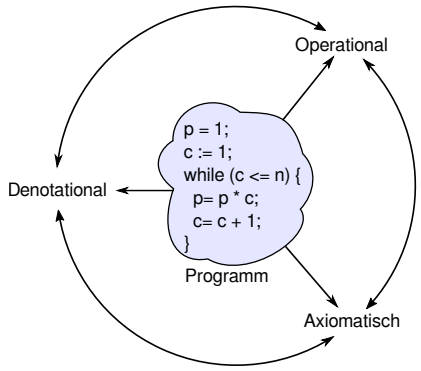
```
// (1)
p = 1; // (2)
c = 1; // (3)
while(c <= n){
  // (4)
  p = p * c;
  c = c + 1; }
// (5)
```

(1) $\quad$ (2) $\quad$ (3)

(4) $\quad (p = 1 \wedge c = 1 \vee p = 1 \wedge c = 2 \vee \; p = (c-1)! \wedge n = p = 2 \wedge c = 3 \vee p = 6 \wedge c = 4) \wedge n = 3$

(5)

Drei Semantiken — Eine Sicht



Zusammenfassung

- Wir wollen die **Bedeutung** (Semantik) von Programmen beschreiben, um ihre Korrektheit beweisen zu können.
- Dazu gibt es verschiedene Ansätze, die wir betrachten werden.
- Nächste Woche geht es mit dem ersten los: **operationale** Semantik