

Korrekte Software: Grundlagen und Methoden

Vorlesung 1 vom 21.04.20

Einführung

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2020



Organisatorisches

▶ Veranstalter:

Christoph Lüth
christoph.lueth@dfki.de
MZH 4186¹, Tel. 59830²

Serge Autexier
serge.autexier@dfki.de
Cartesium 1.49¹, Tel. 59834²

▶ Termine:

- ▶ Dienstag, 12 – 14
- ▶ Donnerstag, 8 – 10 ← **Verlegen?**

▶ Webseite:

<http://www.informatik.uni-bremen.de/~cx1/lehre/ksgm.ss20>

¹Zur Zeit im Home-Office

²Wird weitergeleitet.



Online-Konzept in Corona-Zeiten

- ▶ Keine lange Vorlesung, lieber integrierte Veranstaltung
- ▶ Kürzere Vortragseinheiten (Folie/Lifestream), dazwischen *Arbeitsfragen* (Kurzübungen)
 - ▶ Kein asynchrones Angebot (Aufzeichnung der Meetings?)
- ▶ Wöchentliche Übungsaufgaben zur Vertiefung
- ▶ Technisch:
 - ▶ Nutzung von GotoMeeting:
<https://www.gotomeet.me/DFKI-BAALL/ksgmss20>
 - ▶ Fragen/Kurzübungen in CodiMD:
<http://hackmd.informatik.uni-bremen.de/>
 - ▶ Übungsblätter als ausfüllbare PDFs.



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)



Arbeitsblatt 1.1: Jetzt seid ihr dran!

- ▶ Gruppirt euch in Gruppen zu drei Teilnehmenden! Nutzt dazu folgenden Doodle:
<https://www.doodle.com/poll/utp4mg5yikbfta8d>
- ▶ Zu jeder Gruppe gibt es ein Arbeitsblatt:
https://hackmd.informatik.uni-bremen.de/s/SkVLK1Q_I
- ▶ Auf diesem Arbeitsblatt bearbeitet ihr die Arbeitsfragen im Laufe des Kurses.
- ▶ Bitte nur in "eurem" Arbeitsblatt arbeiten
- ▶ Die Arbeitsblätter sind nicht notenrelevant.



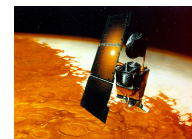
Warum Korrekte Software?



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



Arbeitsblatt 1.2: Jetzt seid ihr dran!

- ▶ Sucht im Netz nach weiteren Software-Disastern:

- 1 Was ist passiert?
- 2 Wie ist es passiert?
- 3 Was war der Softwarefehler?

- ▶ Quellen: Suchmaschine nach Wahl ("software disasters"), The Risks Digest, <https://catless.ncl.ac.uk/Risks/>



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
- ▶ 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**



Arbeitsblatt 1.3: Maschinen und Funktionen

Was genau kann man sich unter "abstrakten Maschine" vorstellen?

Betrachtet als Beispiel die Summe einer Liste von ganzen Zahlen:

- ▶ Wie könnte man eine abstrakte Maschine definieren, welche Listen von Zahlen summiert?
- ▶ Wie könnte man ein mathematisches Objekt definieren, welches Listen von Zahlen summiert?



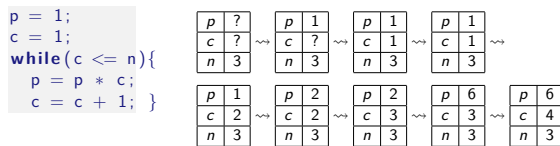
Unsere Sprache C0

- ▶ C0 ist eine **Untermenge** der Sprache C
- ▶ C0-Programme sind **ausführbare** C-Programme
- ▶ Grundausbaustufe:
 - ▶ Zuweisungen, Fallunterscheidungen, Schleifen
 - ▶ Datentypen: ganze Zahlen mit Arithmetik
 - ▶ Relationen: Vergleich ($=, \leq$)
 - ▶ Boolesche 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: Zustandsübergänge einer abstrakten Maschine
- ▶ Abstrakte Maschine hat **impliziten Zustand**
- ▶ Zustand ordnet **Adressen** veränderliche **Werte** zu
- ▶ Konkretes Beispiel: $n \mapsto 3, p$ und c undefiniert

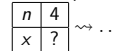


Arbeitsblatt 1.4: Operationale Semantik

Gegeben folgendes C0-Programm:

```
1 x = 0;
2 while (n > 0) {
3   x = x + n * n;
4   n = n - 1;
5 }
```

Entwickeln Sie die ersten zehn Schritte der operationalen Semantik wie im Beispiel oben für den initialen Zustand



Denotationale Semantik

- ▶ Kernkonzept: Abbildung von Programmen auf mathematisches Gegenstück (**Denotat**)
- ▶ **Partielle** Funktionen zwischen Zuständen $\llbracket c \rrbracket : \sigma \rightarrow \sigma$
- ▶ Beispiel:

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

$$\llbracket p_1 \rrbracket(\sigma) = \sigma(p \mapsto 1)(c \mapsto 1)$$

$$\llbracket p_2 \rrbracket(\sigma) = \sigma(p \mapsto \sigma(p) * \sigma(c))(c \mapsto \sigma(c) + 1)$$

$$\llbracket p_3 \rrbracket(\sigma) \llbracket p_3 \rrbracket = ??? \text{fix}(\Gamma(\llbracket c \leq n \rrbracket)(\llbracket p_2 \rrbracket))(\llbracket p_1 \rrbracket(\sigma)) \text{fix}(\Gamma(\llbracket c \leq n \rrbracket)(\llbracket p_2 \rrbracket)) \circ \llbracket p_1 \rrbracket$$

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

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



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)
```

$$(p = 1 \wedge c = 1 \vee p = 1 \wedge c = 2 \vee p = (c - 1)) \wedge n = 3$$

$$(p = 2 \wedge c = 3 \vee p = 6 \wedge c = 4) \wedge n = 3$$

(5)



Arbeitsblatt 1.5: Zusicherungen

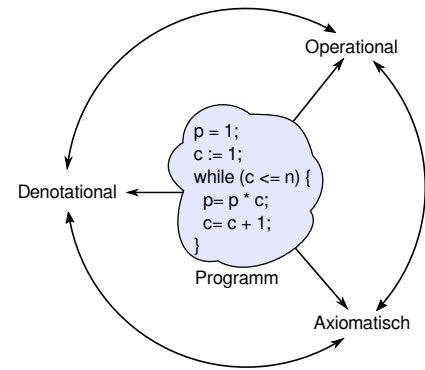
Betrachten Sie folgende Variation des Programms von oben:

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

- ▶ Welche der Zusicherungen (1) – (5) von oben gelten noch?
- ▶ Welche nicht?
- ▶ Was gilt stattdessen?



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

