

## Korrekte Software: Grundlagen und Methoden Vorlesung 1 vom 04.04.18: Einführung

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2018

11.50.47 2018-06-05

1 [22]



## Organisatorisches

### ▶ Veranstalter:

Christoph Lüth                      Serge Autexier  
christoph.lueth@dfki.de        serge.autexier@dfki.de  
MZH 4186, Tel. 59830            Cartesium 2.11, Tel. 59834

### ▶ Termine:

- ▶ Vorlesung: Dienstag, 12 – 14, MZH 6210
- ▶ Übung: Donnerstag, 12 – 14, MZH 1110

### ▶ Webseite:

<http://www.informatik.uni-bremen.de/~cx1/lehre/ksgm.ss18>

Korrekte Software

2 [22]



## 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

Korrekte Software

3 [22]



## 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)

Korrekte Software

4 [22]



## Warum Korrekte Software?

Korrekte Software

5 [22]



## Software-Disaster I: Therac-25



Korrekte Software

6 [22]



## Software-Disasters II: Space



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

Korrekte Software

7 [22]



## 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;
}
```

Korrekte Software

8 [22]



## 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
- ▶ Die Floyd-Hoare-Logik
- ▶ Invarianten und die Korrektheit des Floyd-Hoare-Kalküls
- ▶ Strukturierte Datentypen
- ▶ Modellierung und Spezifikation
- ▶ Verifikationsbedingungen
- ▶ Vorwärts mit Floyd und Hoare
- ▶ Funktionen und Prozeduren
- ▶ Referenzen
- ▶ 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**



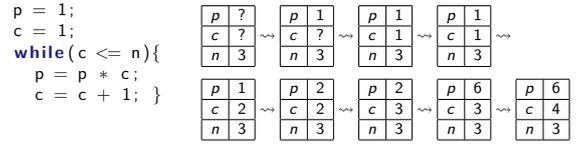
## Unsere Sprache C0

- ▶ 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 ( $=$ ,  $\leq$ )
  - ▶ Boolesche Operatoren: Konjunktion, Disjunktion, Negation
- ▶ 1. Ausbaustufe: Funktionen und Prozeduren
- ▶ 2. Ausbaustufe: Felder und Strukturen
- ▶ 3. Ausbaustufe: Referenzen (Pointer)
- ▶ Fehlt: **union**, **goto**, ...



## Operationale Semantik

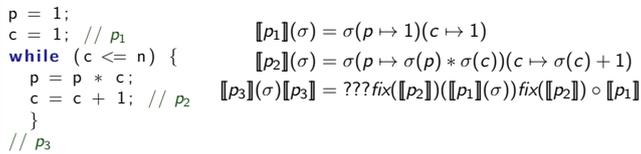
- ▶ Kernkonzept: Zustandsübergänge einer abstrakten Maschine
- ▶ Abstrakte Maschine hat **impliziten Zustand**
- ▶ Zustand ordnet **Adressen** veränderliche **Werten** zu
- ▶ Konkretes Beispiel:  $n \mapsto 3$ ,  $p$  und  $c$  undefiniert



## Denotationale Semantik

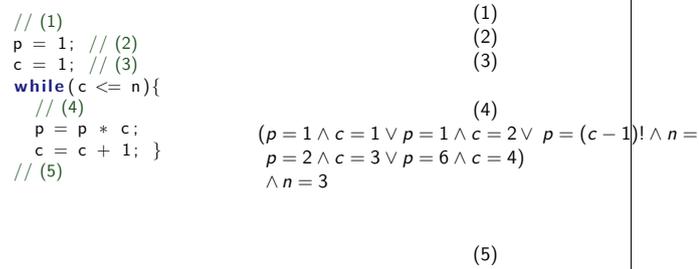
- ▶ Kernkonzept: Abbildung von Programmen auf mathematisches Gegenstück (**Denotat**)
- ▶ **Partielle** Funktionen zwischen Zuständen  $\llbracket c \rrbracket : \sigma \rightarrow \sigma$

▶ Beispiel:

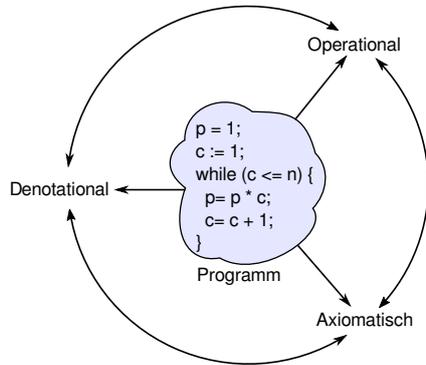


## Axiomatische Semantik

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



## 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

