

Korrekte Software: Grundlagen und Methoden
Vorlesung 12 vom 26.06.17: Programmsicherheit und Frame Conditions

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2017

14.07.22 2017-07-03

1 [13]



Fahrplan

- ▶ Einführung
- ▶ Die Floyd-Hoare-Logik
- ▶ Operationale Semantik
- ▶ Denotationale Semantik
- ▶ Äquivalenz der Operationalen und Denotationalen Semantik
- ▶ Korrektheit des Hoare-Kalküls
- ▶ Vorwärts und Rückwärts mit Floyd und Hoare
- ▶ Funktionen und Prozeduren
- ▶ Referenzen und Speichermodelle
- ▶ Verifikationsbedingungen Revisited
- ▶ Vorwärtsrechnung Revisited
- ▶ **Programmsicherheit und Frame Conditions**
- ▶ Ausblick und Rückblick

Korrekte Software

2 [13]



Heute im Angebot

- ▶ Programmsicherheit Revisited:
 - ▶ Deferenzierung von Pointer und Arrays
 - ▶ Division durch 0
- ▶ Frame Conditions
 - ▶ Was ist das und wozu braucht man das?
 - ▶ Wie könnte man das in unserer Sprache behandeln?
 - ▶ Modification sets

Korrekte Software

3 [13]



Zur Erinnerung: Totale Korrektheit

- ▶ Partielle Korrektheit: wenn das Programm terminiert, erfüllt es die Nachbedingung.

Wie sinnvoll ist diese Aussage?

Mein Programm wäre richtig gewesen, wenn es nicht vorher abgestürzt wäre.

- ▶ Wir wollen **mindestens** ausschließen, dass Laufzeitfehler ("undefined behaviour" *C99 Standard*, §3.4.3) auftreten.
- ▶ Problem: wenn Pointer als Parameter übergeben werden, müssen sie **dereferenzierbar** sein.
- ▶ Dazu neue Annotationen: `\valid()` und `\array()`

Korrekte Software

4 [13]



Spezifikation von Zeigern und Feldern

Die Prädikate `\valid(x)` und `\array(a, n)`

`\valid(x)` für x Pointer-Typ \iff $*x$ ist definiert.
`\array(a, n)` für a Pointer-Typ \iff a ist ein Feld der Länge n .

- ▶ Abhängig vom Zustand (warum?)
- ▶ Felder als Parameter werden Zeigern konvertiert, deshalb müssen wir spezifizieren können, dass ein Zeiger "in Wirklichkeit" ein Feld ist.
- ▶ Formale Definition:

$$\begin{aligned} \text{\textbackslash valid}(x, S) &\stackrel{\text{def}}{=} \exists v. \text{read}(S, \text{read}(S, x)) = v \\ \text{\textbackslash array}(a, n, S) &\stackrel{\text{def}}{=} \forall i. 0 \leq i < n \implies \text{\textbackslash valid}(a[i], S) \end{aligned}$$

Korrekte Software

5 [13]



Wie beweisen wir Validität?

- ▶ Konvention: $\text{\textbackslash valid}(x) \stackrel{\text{def}}{=} \text{\textbackslash valid}(x, \sigma)$
- ▶ Herleitung von Validität:

$$\begin{aligned} \frac{x = \&e}{\text{\textbackslash valid}(x, S)} \quad \frac{S1 = \text{upd}(S, x, y) \quad \text{mit } y \in \text{Loc}, y \neq \text{read}(S, y')}{\text{\textbackslash valid}(x, S1)} \\ \frac{\text{\textbackslash valid}(x, S) \ \&\& \ S1 = \text{upd}(S, y, l) \ \&\& \ y \neq x}{\text{\textbackslash valid}(x, S1)} \\ \frac{\text{\textbackslash valid}(y, S) \ \&\& \ S1 = \text{upd}(S, x, \text{read}(S, y))}{\text{\textbackslash valid}(x, S1)} \\ \frac{\text{\textbackslash array}(a, n, S) \quad 0 \leq i < n}{\text{\textbackslash valid}(a[i], S)} \end{aligned}$$

Korrekte Software

6 [13]



Program Safety

- ▶ Hier: Dereferenzierungen definiert, keine Division durch 0
- ▶ Formal als **Verifikationsbedingungen**

$$\begin{aligned} \text{safe}(x, S) &= \emptyset \\ \text{safe}(*x, S) &= \{\text{\textbackslash valid}(x, S)\} \\ \text{safe}(a[i]) &= \{\text{\textbackslash array}(a, n) \ \&\& \ 0 \leq i \ \&\& \ i < n\} \\ \text{safe}(x + y, S) &= \text{safe}(x, S) \cup \text{safe}(y, S) \\ \text{safe}(x/y, S) &= \{y \neq 0\} \\ &\dots \end{aligned}$$

- ▶ Nicht ganz exakt: $y \neq 0$ muss im Zustand S ausgewertet werden,
 - ▶ D.h. $P \implies y \neq 0$ mit P Vorbedingung
 - ▶ Dazu muss safe in die Definition von asp/awp eingebunden werden
- ▶ Valid-Analysen bleiben rein schematisch
- ▶ Division durch 0 und Arrayzugriffe benötigen Auswertung

Korrekte Software

7 [13]



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.

Korrekte Software

8 [13]



Frame Rule

- ▶ Konstanzregel (Rule of Constancy):

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

- ▶ Nebenbedingung: keine in c **veränderte** Variable tritt in R auf
- ▶ Problem: gilt mit Pointern nur **ingeschränkt**, da c eventuell ohne direkte Zuweisung Teile des Zustands verändert, über den R Aussagen macht.



Modification Sets

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

- ▶ Syntax: modifies **Mexp**

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

- ▶ Mexp sind Lexp, die auch **Teile** von Feldern bezeichnen.
- ▶ Semantik: $\llbracket - \rrbracket : Env \rightarrow \mathbf{Mexp} \rightarrow \Sigma \rightarrow \mathbb{P}(\mathbf{Loc})$
- ▶ Modification Sets werden in die Hoare-Tripel **integriert**.



Erweiterung der Hoare-Tripel

- ▶ Hoare-Tripel mit Modification Sets:

$$\Lambda \models \{P\} c \{Q\} \iff \forall \sigma. P(\sigma) \wedge \exists \sigma'. \sigma' = c(\sigma) \implies Q(\sigma') \wedge \sigma \cong_L \sigma'$$

- ▶ wobei $\sigma \cong_L \tau$: Zustände σ und τ sind **gleich bis auf** die Adressen in L

$$\sigma \cong_L \tau \iff \forall l \in \text{dom}(\sigma) \cup \text{dom}(\tau) \setminus L. \sigma(l) = \tau(l)$$

bzw.

$$\sigma \cong_L \tau \iff \forall l. \sigma(l) \neq \tau(l) \implies l \in L$$



Erweiterung der Regeln

- ▶ Regeln werden mit Modification Set annotiert:

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

- ▶ Modification Set wird durchgereicht
- ▶ Zuweisungsregel wird ergänzt (vorwärts/rückwärts):

$$\frac{S \notin FV(P)}{\Gamma, \Lambda \vdash \{P\} x = e \{ \exists S. P[S/\sigma] \ \&\& \ x_S^\dagger \in \Lambda \ \&\& \ \sigma == \text{upd}(S, x_S^\dagger, e_S^\#) \}}$$

$$\frac{}{\Gamma, \Lambda \vdash \{Q[\text{upd}(\sigma, x_S^\dagger)/\sigma] \ \&\& \ x_\sigma^\dagger \in \Lambda\} x = e \{Q\}}$$



Zusammenfassung

- ▶ Programmsicherheit kann als zusätzliche **Verifikationsbedingung** formuliert werden
 - ▶ Nachteil: teilweise komplexe Verifikationsbedingungen
 - ▶ Vorteil: semantische Integrität
- ▶ Frame Rule: spezifiziert **unveränderte** Teile des Zustands
 - ▶ Essentiell für Skalierbarkeit
 - ▶ Bei Zeigern rein syntaktische Analyse (freie Variablen) nicht ausreichend, daher **modification sets**
 - ▶ Spezifizieren **veränderlichen** Teil des Zustandes
 - ▶ Werden bei Zuweisungen geprüft

