

Korrekte Software: Grundlagen und Methoden  
Vorlesung 9 vom 05.06.18: Verifikationsbedingungen

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2018



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



Idee

- ▶ Hier ist ein einfaches Programm:

```
// {X = x ∧ Y = y}
z = y;
// {X = x ∧ Y = z}
y = x;
// {X = y ∧ Y = z}
x = z;
// {X = y ∧ Y = x}
```

- ▶ Wir sehen:

- 1 Die Verifikation erfolgt **rückwärts** (von hinten nach vorne).
- 2 Die Verifikation kann **berechnet** werden.

- ▶ Geht das immer?



Rückwärtsanwendung der Regeln

- ▶ Zuweisungsregel kann **rückwärts** angewandt werden, weil die Nachbedingung eine offene Variable ist —  $P$  passt auf jede beliebige Nachbedingung.

$$\frac{}{\vdash \{P[e/x]\} x = e \{P\}}$$

- ▶ Was ist mit den anderen Regeln? Nur **while** macht Probleme!

$$\frac{}{\vdash \{A\} \{ \} \{A\}} \quad \frac{\vdash \{A \wedge b\} c_0 \{B\} \quad \vdash \{A \wedge \neg b\} c_1 \{B\}}{\vdash \{A\} \text{if } (b) c_0 \text{ else } c_1 \{B\}}$$

$$\frac{\vdash \{A\} c_1 \{B\} \quad \vdash \{B\} c_2 \{C\}}{\vdash \{A\} c_1; c_2 \{C\}} \quad \frac{\vdash \{A \wedge b\} c \{A\}}{\vdash \{A\} \text{while } (b) c \{A \wedge \neg b\}}$$

$$\frac{A' \implies A \quad \vdash \{A\} c \{B\} \quad B \implies B'}{\vdash \{A'\} c \{B'\}}$$



Berechnung von Vorbedingungen

- ▶ Die Rückwärtsrechnung von einer gegebenen Nachbedingung entspricht der Berechnung einer Vorbedingung.
- ▶ Gegeben C0-Programm  $c$ , Prädikat  $Q$ , dann ist
  - ▶  $wp(c, Q)$  die **schwächste Vorbedingung**  $P$  so dass  $\models \{P\} c \{Q\}$ ;
  - ▶ Prädikat  $P$  **schwächer** als  $P'$  wenn  $P' \implies P$

- ▶ Semantische Charakterisierung:

Schwächste Vorbedingung

Gegeben Zusicherung  $Q \in \text{Assn}$  und Programm  $c \in \text{Stmt}$ , dann

$$\models \{P\} c \{Q\} \iff P \implies wp(c, Q)$$

- ▶ Wie können wir  $wp(c, Q)$  berechnen?



Berechnung von  $wp(c, Q)$

- ▶ Einfach für Programme ohne Schleifen:

$$\begin{aligned} wp(\{ \}, P) &\stackrel{\text{def}}{=} P \\ wp(x = e, P) &\stackrel{\text{def}}{=} P[e/x] \\ wp(c_1; c_2, P) &\stackrel{\text{def}}{=} wp(c_1, wp(c_2, P)) \\ wp(\text{if } (b) c_0 \text{ else } c_1, P) &\stackrel{\text{def}}{=} (b \wedge wp(c_0, P)) \vee (\neg b \wedge wp(c_1, P)) \end{aligned}$$

- ▶ Für Schleifen: nicht entscheidbar.
  - ▶ "Cannot in general compute a **finite** formula" (Mike Gordon)
- ▶ Wir können rekursive Formulierung angeben:

$$wp(\text{while } (c) .P) \stackrel{\text{def}}{=} (\neg b \wedge P) \vee (b \wedge wp(c, wp(\text{while } (b) c, P)))$$

- ▶ Hilft auch nicht weiter...



Lösung: Annotierte Programme

- ▶ Wir helfen dem Rechner weiter und **annotieren** die Schleifeninvariante am Programm.
- ▶ Damit berechnen wir:
  - ▶ die **approximative** schwächste Vorbedingung  $awp(c, Q)$
  - ▶ zusammen mit einer Menge von **Verifikationsbedingungen**  $wvc(c, Q)$
- ▶ Die Verifikationsbedingungen treten dort auf, wo die Weakening-Regel angewandt wird.
- ▶ Es gilt:

$$\bigwedge wvc(c, Q) \implies \models \{awp(c, Q)\} c \{Q\}$$



Approximative schwächste Vorbedingung

- ▶ Für die **while**-Schleife:

$$\begin{aligned} awp(\text{while } (b) /** \text{inv } i */ c, P) &\stackrel{\text{def}}{=} i \\ wvc(\text{while } (b) /** \text{inv } i */ c, P) &\stackrel{\text{def}}{=} wvc(c, i, \\ &\quad \cup \{i \wedge b \implies awp(c, i)\} \\ &\quad \cup \{i \wedge \neg b \implies P\} \end{aligned}$$

- ▶ Entspricht der **while**-Regel (1) mit Weakening (2):

$$\frac{\vdash \{A \wedge b\} c \{A\}}{\vdash \{A\} \text{while } (b) c \{A \wedge \neg b\}} \quad (1)$$

$$\frac{A \wedge b \implies C \quad \vdash \{C\} c \{A\} \quad A \wedge \neg b \implies B}{\vdash \{A\} \text{while } (b) c \{B\}} \quad (2)$$



## Überblick: Approximative schwächste Vorbedingung

$$\begin{aligned} \text{awp}(\{\}, P) &\stackrel{\text{def}}{=} P \\ \text{awp}(x = e, P) &\stackrel{\text{def}}{=} P[e/x] \\ \text{awp}(c_1; c_2, P) &\stackrel{\text{def}}{=} \text{awp}(c_1, \text{awp}(c_2, P)) \\ \text{awp}(\text{if } (b) \ c_0 \ \text{else } \ c_1, P) &\stackrel{\text{def}}{=} (b \wedge \text{awp}(c_0, P)) \vee (\neg b \wedge \text{awp}(c_1, P)) \\ \text{awp}(\text{while } (b) \ \text{/** inv } \ i \ \text{*/} \ c, P) &\stackrel{\text{def}}{=} i \\ \\ \text{wvc}(\{\}, P) &\stackrel{\text{def}}{=} \emptyset \\ \text{wvc}(x = e, P) &\stackrel{\text{def}}{=} \emptyset \\ \text{wvc}(c_1; c_2, P) &\stackrel{\text{def}}{=} \text{wvc}(c_1, \text{awp}(c_2, P)) \cup \text{wvc}(c_2, P) \\ \text{wvc}(\text{if } (b) \ c_0 \ \text{else } \ c_1, P) &\stackrel{\text{def}}{=} \text{wvc}(c_0, P) \cup \text{wvc}(c_1, P) \\ \text{wvc}(\text{while } (b) \ \text{/** inv } \ i \ \text{*/} \ c, P) &\stackrel{\text{def}}{=} \text{wvc}(c, i) \cup \{i \wedge b \rightarrow \text{awp}(c, i)\} \\ &\quad \cup \{i \wedge \neg b \rightarrow P\} \\ \\ \text{wvc}(\{P\} \ c \ \{Q\}) &\stackrel{\text{def}}{=} \{P \rightarrow \text{awp}(c, Q)\} \cup \text{wvc}(c, Q) \end{aligned}$$


## Beispiel: das Fakultätsprogramm

► In der Praxis sind Vorbedingung gegeben, und nur die Verifikationsbedingungen relevant.

► Sei  $F$  das annotierte Fakultätsprogramm:

```

1 // {0 ≤ n}
2 p = 1;
3 c = 1;
4 while (c ≤ n) /** inv {p = (c-1)! ∧ c-1 ≤ n}; */ {
5     p = p * c;
6     c = c + 1;
7 }
8 // {p = n!}
    
```

► Berechnung der Verifikationsbedingungen zur Nachbedingung.



## Notation für Verifikationsbedingungen

```

1 // {0 ≤ n}
2 p = 1;
3 c = 1;
4 while (c ≤ n) /** inv {p = (c-1)! ∧ c-1 ≤ n}; */ {
5     p = p * c;
6     c = c + 1;
7 }
8 // {p = n!}
    
```

**AWP**

$$\begin{aligned} 6 \quad &p = ((c+1)-1)! \wedge ((c-1)+1) \leq n \\ 5 \quad &p \cdot c = ((c+1)-1)! \wedge ((c-1)+1) \leq n \\ 3 \quad &p = (1-1)! \wedge (1-1) \leq n \\ 2 \quad &1 = (1-1)! \wedge (1-1) \leq n \end{aligned}$$

**VC**

$$\begin{aligned} 4 \quad &p = (c-1)! \wedge c-1 \leq n \wedge \neg(c \leq n) \rightarrow \\ &\quad p \cdot c = ((c+1)-1)! \wedge ((c-1)+1) \leq n \\ 4 \quad &p = (c-1)! \wedge c-1 \leq n \wedge \neg(c \leq n) \rightarrow p = n! \\ 1 \quad &0 \leq n \rightarrow 1 = (1-1)! \wedge (1-1) \leq n \end{aligned}$$


## Vereinfachung von Verifikationsbedingungen

Wir nehmen folgende **strukturellen Vereinfachungen** an den generierten Verifikationsbedingungen vor:

① Auswertung konstanter arithmetischer Ausdrücke, einfache arithmetische Gesetze

► Bsp.  $(x+1) - 1 \rightsquigarrow x$ ,  $1 - 1 \rightsquigarrow 0$

② Normalisierung der Relationen (zu  $<$ ,  $\leq$ ,  $=$ ,  $\neq$ ) und Vereinfachung

► Bsp.  $\neg(x \leq y) \rightsquigarrow x > y \rightsquigarrow y < x$

③ Konjunktionen in der Konklusion werden zu einzelnen Verifikationsbedingungen

► Bsp.  $A_1 \wedge A_2 \wedge A_3 \rightarrow P \wedge Q \rightsquigarrow A_1 \wedge A_2 \wedge A_3 \rightarrow P, A_1 \wedge A_2 \wedge A_3 \rightarrow Q$

④ Alle Bedingungen mit einer Prämisse *false* oder einer Konklusion *true* sind trivial erfüllt.



## Weiteres Beispiel: Maximales Element

```

1 // {0 < n}
2 i = 0;
3 r = 0;
4 while (i != n) /** inv (∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ r < n */
5 {
6     if (a[r] < a[i]) {
7         r = i;
8     }
9     else {
10    }
11    i = i + 1;
12 }
13 // {(∀j. 0 ≤ j < n → a[j] ≤ a[r]) ∧ 0 ≤ r < n}
    
```

► Sehr lange Verifikationsbedingungen (u.a. wegen Fallunterscheidung)

► Wie können wir das beheben?



## Spracherweiterung: Explizite Spezifikationen

► Erweiterung der Sprache C0 um Invarianten für Schleifen und **explizite Zusicherung**

**Assn**  $a ::= \dots$  — Zusicherungen

**Stmt**  $c ::= l = e \mid c_1; c_2 \mid \{\} \mid \text{if } (b) \ c_1 \ \text{else } \ c_2$   
 $\quad \mid \text{while } (b) \ \text{/** inv } \ a \ \text{*/} \ c$   
 $\quad \mid \text{/** } \{a\} \ \text{*/}$

► Zusicherungen haben **keine Semantik** (Kommentar!), sondern erzwingen eine neue Vorbedingung.

► Dazu vereinfachte Regel für Fallunterscheidung:

$$\text{awp}(\text{if } (b) \ c_0 \ \text{else } \ c_1, \stackrel{\text{def}}{=} (b \wedge \text{awp}(c_0, P)) \vee (\neg b \wedge \text{awp}(c_1, P)))$$

Wenn  $\text{awp}(c_0, P) = b \wedge P_0$ ,  $\text{awp}(c_1, P) = \neg b \wedge P_0$ , dann gilt

$$(b \wedge b \wedge P_0) \vee (\neg b \wedge \neg b \wedge P_0) = (b \wedge P_0) \vee (\neg b \wedge P_0) = (b \vee \neg b) \wedge P_0 = P_0$$


## Überblick: Approximative schwächste Vorbedingung

$$\begin{aligned} \text{awp}(\{\}, P) &\stackrel{\text{def}}{=} P \\ \text{awp}(x = e, P) &\stackrel{\text{def}}{=} P[e/x] \\ \text{awp}(c_1; c_2, P) &\stackrel{\text{def}}{=} \text{awp}(c_1, \text{awp}(c_2, P)) \\ \text{awp}(\text{if } (b) \ c_0 \ \text{else } \ c_1, P) &\stackrel{\text{def}}{=} Q \ \text{wenn} \ \text{awp}(c_0, P) = b \wedge Q, \\ &\quad \text{awp}(c_1, P) = \neg b \wedge Q \\ \text{awp}(\text{if } (b) \ c_0 \ \text{else } \ c_1, P) &\stackrel{\text{def}}{=} (b \wedge \text{awp}(c_0, P)) \vee (\neg b \wedge \text{awp}(c_1, P)) \\ \text{awp}(\text{/** } \{q\} \ \text{*/}, P) &\stackrel{\text{def}}{=} q \\ \text{awp}(\text{while } (b) \ \text{/** inv } \ i \ \text{*/} \ c, P) &\stackrel{\text{def}}{=} i \\ \\ \text{wvc}(\{\}, P) &\stackrel{\text{def}}{=} \emptyset \\ \text{wvc}(x = e, P) &\stackrel{\text{def}}{=} \emptyset \\ \text{wvc}(c_1; c_2, P) &\stackrel{\text{def}}{=} \text{wvc}(c_1, \text{awp}(c_2, P)) \cup \text{wvc}(c_2, P) \\ \text{wvc}(\text{if } (b) \ c_0 \ \text{else } \ c_1, P) &\stackrel{\text{def}}{=} \text{wvc}(c_0, P) \cup \text{wvc}(c_1, P) \\ \text{wvc}(\text{/** } \{q\} \ \text{*/}, P) &\stackrel{\text{def}}{=} \{q \rightarrow P\} \\ \text{wvc}(\text{while } (b) \ \text{/** inv } \ i \ \text{*/} \ c, P) &\stackrel{\text{def}}{=} \text{wvc}(c, i) \cup \{i \wedge b \rightarrow \text{awp}(c, i)\} \\ &\quad \cup \{i \wedge \neg b \rightarrow P\} \end{aligned}$$


## Maximales Element mit Zusicherung

```

1 // {0 < n}
2 i = 0;
3 r = 0;
4 while (i != n) /** inv (∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ r < n */
5 {
6     if (a[r] < a[i]) {
7         /** {∀j. 0 ≤ j < i → a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ a[r] < a[i]} */
8         r = i;
9     }
10    else {
11        /** {∀j. 0 ≤ j < i → a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])} */
12    }
13    i = i + 1;
14 }
15 // {(∀j. 0 ≤ j < n → a[j] ≤ a[r]) ∧ 0 ≤ r < n}
    
```

► Explizite Zusicherungen verkleinern Verifikationsbedingung



## Zusammenfassung

- ▶ Die Regeln des Floyd-Hoare-Kalküls lassen sich, weitgehend schematisch, rückwärts (vom Ende her) anwenden — nur Schleifen machen Probleme.
- ▶ Wir **annotieren** daher die Invarianten an Schleifen, und können dann die schwächste Vorbedingung und Verifikationsbedingungen automatisch berechnen.
  - ▶ Dabei sind die **Verifikationsbedingungen** das interessante.
- ▶ Um die Verifikationsbedingungen zu vereinfachen führen wir **explizite Zusicherungen** in C0 ein
- ▶ Die Generierung von Verifikationsbedingungen korrespondiert zur relativen Vollständigkeit der Floyd-Hoare-Logik.
- ▶ Nächste Woche: warum eigentlich immer **rückwärts**?