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;

y = x;

x = z;
// {X = y ∧ Y = x}
```

Idee

- Hier ist ein einfaches Programm:

$// \{X = x \wedge Y = y\}$
z = y ;

y = x ;
$// \{X = y \wedge Y = z\}$
x = z ;
$// \{X = y \wedge Y = x\}$

Idee

- Hier ist ein einfaches Programm:

$// \{X = x \wedge Y = y\}$
z = y ;
$// \{X = x \wedge Y = z\}$
y = x ;
$// \{X = y \wedge Y = z\}$
x = z ;
$// \{X = y \wedge Y = x\}$

Idee

- Hier ist ein einfaches Programm:

  // $\{X = x \wedge Y = y\}$
  z = y ;
  // $\{X = x \wedge Y = z\}$
  y = x ;
  // $\{X = y \wedge Y = z\}$
  x = z ;
  // $\{X = y \wedge Y = x\}$

- Wir sehen:

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

Idee

- Hier ist ein einfaches Programm:

  // $\{X = x \wedge Y = y\}$
  z = y ;
  // $\{X = x \wedge Y = z\}$
  y = x ;
  // $\{X = y \wedge Y = z\}$
  x = z ;
  // $\{X = y \wedge Y = x\}$

- 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.

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

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?

$$\frac{}{\vdash \{A\}\, \{\}\, \{A\}} \qquad \frac{\vdash \{A \wedge b\}\, c_0\, \{B\} \qquad \vdash \{A \wedge \neg b\}\, c_1\, \{B\}}{\vdash \{A\}\ \textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1\, \{B\}}$$

$$\frac{\vdash \{A\}\, c_1\, \{B\} \qquad \vdash \{B\}\, c_2\, \{C\}}{\vdash \{A\}\, c_1; c_2\, \{C\}} \qquad \frac{\vdash \{A \wedge b\}\, c\, \{A\}}{\vdash \{A\}\ \textbf{while}\ (b)\ c\, \{A \wedge \neg b\}}$$

$$\frac{A' \Longrightarrow A \qquad \vdash \{A\}\, c\, \{B\} \qquad B \Longrightarrow B'}{\vdash \{A'\}\, c\, \{B'\}}$$

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!

$$\frac{}{\vdash \{A\}\, \{\}\, \{A\}} \qquad \frac{\vdash \{A \wedge b\}\, c_0\, \{B\} \qquad \vdash \{A \wedge \neg b\}\, c_1\, \{B\}}{\vdash \{A\}\ \textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1\, \{B\}}$$

$$\frac{\vdash \{A\}\, c_1\, \{B\} \qquad \vdash \{B\}\, c_2\, \{C\}}{\vdash \{A\}\, c_1; c_2\, \{C\}} \qquad \frac{\vdash \{A \wedge b\}\, c\, \{A\}}{\vdash \{A\}\ \textbf{while}\ (b)\ c\, \{A \wedge \neg b\}}$$

$$\frac{A' \Longrightarrow A \qquad \vdash \{A\}\, c\, \{B\} \qquad B \Longrightarrow B'}{\vdash \{A'\}\, c\, \{B'\}}$$

Berechnung von Vorbedingungen

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

Schwächste Vorbedingung

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

$$\models \{P\}\, c\, \{Q\} \Longleftrightarrow P \Longrightarrow \mathrm{wp}(c, Q)$$

- Wie können wir $\mathrm{wp}(c, Q)$ berechnen?

Berechnung von $\mathrm{wp}(c, Q)$

- Einfach für Programme ohne Schleifen:

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

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

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

  - 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 $\mathsf{awp}(c, Q)$
  - zusammen mit einer Menge von **Verifikationsbedingungen** $\mathsf{wvc}(c, Q)$
- Die Verifikationsbedingungen treten dort auf, wo die Weakening-Regel angewandt wird.
- Es gilt:

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

Approximative schwächste Vorbedingung

- Für die **while**-Schleife:

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

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

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

$$\frac{A \wedge b \Longrightarrow C \qquad \vdash \{C\}\, c\, \{A\} \qquad A \wedge \neg b \Longrightarrow B}{\vdash \{A\}\ \textbf{while}\ (b)\ c\, \{B\}} \quad (2)$$

Überblick: Approximative schwächste Vorbedingung

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

$$
\begin{aligned}
\mathrm{wvc}(\{\,\}, P) &\stackrel{def}{=} \emptyset \\
\mathrm{wvc}(x = e, P) &\stackrel{def}{=} \emptyset \\
\mathrm{wvc}(c_1; c_2, P) &\stackrel{def}{=} \mathrm{wvc}(c_1, \mathrm{awp}(c_2, P)) \cup \mathrm{wvc}(c_2, P) \\
\mathrm{wvc}(\textbf{if } (b)\ c_0\ \textbf{else}\ c_1, P) &\stackrel{def}{=} \mathrm{wvc}(c_0, P) \cup \mathrm{wvc}(c_1, P) \\
\mathrm{wvc}(\textbf{while } (b)\ /{**}\ \textbf{inv}\ i\ {*}/\ c, P) &\stackrel{def}{=} \mathrm{wvc}(c, i) \cup \{i \wedge b \longrightarrow \mathrm{awp}(c, i)\} \\
&\qquad \cup \{i \wedge \neg b \longrightarrow P\}
\end{aligned}
$$

$$
wvc(\{P\}\, c\, \{Q\}) \stackrel{def}{=} \{P \longrightarrow \mathrm{awp}(c, Q)\} \cup \mathrm{wvc}(c, Q)
$$

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

6	$p = ((c+1)-1)! \wedge ((c-1)+1) \leq n$
5	$p \cdot c = ((c+1)-1)! \wedge ((c-1)+1) \leq n$
3	$p = (1-1)! \wedge (1-1) \leq n$
2	$1 = (1-1)! \wedge (1-1) \leq n$

VC

4	$p = (c-1)! \wedge c-1 \leq n \wedge \neg(c \leq n) \longrightarrow$ $p \cdot c = ((c+1)-1)! \wedge ((c-1)+1) \leq n$
4	$p = (c-1)! \wedge c-1 \leq n \wedge \neg(c \leq n) \longrightarrow p = n!$
1	$0 \leq n \longrightarrow 1 = (1-1)! \wedge (1-1) \leq n$

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: $A_1 \wedge A_2 \wedge A_3 \longrightarrow P \wedge Q \rightsquigarrow A_1 \wedge A_2 \wedge A_3 \longrightarrow P, A_1 \wedge A_2 \wedge A_3 \longrightarrow Q$
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**

$$\begin{array}{lll} \textbf{Assn} & a ::= \ldots & \text{— Zusicherungen} \\ \textbf{Stmt} & c ::= & l = e \mid c_1; c_2 \mid \{\,\} \mid \textbf{if}\ (b)\ c_1\ \textbf{else}\ c_2 \\ & \mid & \textbf{while}\ (b)\ /{*}{*}\,\textbf{inv}\ a\ {*}/\ c \\ & \mid & /{*}{*}\{a\}\ {*}/ \end{array}$$

- Zusicherungen haben **keine Semantik** (Kommentar!), sondern erzwingen eine neue Vorbedingung.
- Dazu vereinfachte Regel für Fallunterscheidung:

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

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

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

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**?