Korrekte Software: Grundlagen und Methoden
Vorlesung 10 vom 12.06.17: Verifikationsbedingungen Revisited

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2017

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

Heute

- Der Hoare-Kalkül ist viel Schreibarbeit
- Deshalb haben wir Verifikationsbedingungen berechnet:
  - Approximative schwächste Vorbedingung
  - Approximative stärkste Nachbedingung
- Mit Zeigern ist rückwärts nicht das beste ...

Formal: Konversion in Zustandsprädikate

$$
\begin{aligned}
(-)^\dagger &: \mathbf{Lexp} \rightarrow \mathbf{Lexp} \\
v^\dagger &= v \quad (v \text{ Variable}) \\
l.id^\dagger &= l^\dagger.id \\
l[e]^\dagger &= l^\dagger[e^\#] \\
*l^\dagger &= l^\#
\end{aligned}
$$

$$
\begin{aligned}
(-)^\# &: \mathbf{Aexp} \rightarrow \mathbf{Aexp} \\
e^\# &= read(\sigma, e^\dagger) \quad (e \in \mathbf{Lexp}) \\
n^\# &= n \\
v^\# &= v \quad (v \text{ logische Variable}) \\
\&e^\# &= e^\dagger \\
e_1 + e_2{}^\# &= e_1{}^\# + e_2{}^\# \\
\backslash\mathbf{result}^\# &= \backslash\mathbf{result} \\
\backslash\mathbf{old}(e)^\# &= \backslash\mathbf{old}(e)
\end{aligned}
$$

$$
\frac{}{\Gamma \vdash \{Q[upd(\sigma, x^\dagger, e^\#)/\sigma]\}\, x = e\, \{Q|R\}}
$$

Approximative schwächste Vorbedingung

- Für die Berechnung der approximativen schwächsten Vorbedingung (AWP) und der Verifikationsbedingungen (WVC) müssen zwei Anpassungen vorgenommen werden:
  - Sowohl AWP als auch WVC berechnen symbolische Zustandsprädikate.
  - Die Zuweisungsregel muss angepasst werden.
- Berechnung von awp und wvc:

$$\begin{aligned}
&\mathrm{awp}(\Gamma, f(x_1, \ldots, x_n) /\!** \text{ pre } P \text{ post } Q *\!/ \; \{ds\ blk\}) \\
&\qquad \stackrel{def}{=} \mathrm{awp}(\Gamma', blk, Q^{\#}, Q^{\#}) \\
&\mathrm{wvc}(\Gamma, f(x_1, \ldots, x_n) /\!** \text{ pre } P \text{ post } Q *\!/ \; \{ds\ blk\}) \\
&\qquad \stackrel{def}{=} \{P^{\#} \Longrightarrow \mathrm{awp}(\Gamma', blk, Q^{\#}, Q^{\#})[e_j{}^{\#} / \backslash\mathbf{old}(e_j)]\} \\
&\qquad\quad \cup\, \mathrm{wvc}(\Gamma', blk, Q^{\#}, Q^{\#}) \\
&\Gamma' \stackrel{def}{=} \Gamma[f \mapsto \forall x_1, \ldots, x_n.\, (P, Q)]
\end{aligned}$$

Approximative schwächste Vorbedingung (Revisited)

$$\mathrm{awp}(\Gamma, \{\,\}, Q, Q_R) \stackrel{def}{=} Q$$

$$\mathrm{awp}(\Gamma, l = f(e_1, \ldots, e_n), Q, Q_R) \stackrel{def}{=} P[e_i/x_i]^\# \quad \text{mit } \Gamma(f) = \forall x_1, \ldots, x_n.\,(P, R)$$

$$\mathrm{awp}(\Gamma, f(e_1, \ldots, e_n), Q, Q_R) \stackrel{def}{=} P[e_i/x_i]^\# \quad \text{mit } \Gamma(f) = \forall x_1, \ldots, x_n.\,(P, R)$$

$$\mathrm{awp}(\Gamma, l = e, Q, Q_R) \stackrel{def}{=} Q[upd(\sigma, l^\dagger, e^\#)/\sigma]$$

$$\mathrm{awp}(\Gamma, \{c\ c_s\}, Q, Q_R) \stackrel{def}{=} \mathrm{awp}(\Gamma, c, \mathrm{awp}(\{c_s\}, Q, Q_R), Q_R)$$

$$\mathrm{awp}(\Gamma, \textbf{if}\ (b)\ \{c_0\}\ \textbf{else}\ \{c_1\}, Q, Q_R) \stackrel{def}{=} (b^\#\ \&\&\ \mathrm{awp}(\Gamma, c_0, Q, Q_R)) \;||\; (\,!\,b^\#\ \&\&\ \mathrm{awp}(\Gamma, c_1, Q, Q_R))$$

$$\mathrm{awp}(\Gamma, /**\ \{q\}\ */, Q, Q_R) \stackrel{def}{=} q$$

$$\mathrm{awp}\left(\Gamma, \begin{bmatrix} \textbf{while}\ (b) \\ /**\ \textbf{inv}\ \ i\ */ \\ c \end{bmatrix}, Q, Q_R\right) \stackrel{def}{=} i$$

$$\mathrm{awp}(\Gamma, \textbf{return}\ \ e, Q, Q_R) \stackrel{def}{=} Q_R[e^\# / \backslash\textbf{result}]$$

$$\mathrm{awp}(\Gamma, \textbf{return}, Q, Q_R) \stackrel{def}{=} Q_R$$

Approximative Verifikationsbedingungen (Revisited)

$$\begin{aligned}
\mathrm{wvc}(\Gamma, \{\,\}, Q, Q_R) &\stackrel{def}{=} \emptyset \\
\mathrm{wvc}(\Gamma, x = e, Q, Q_R) &\stackrel{def}{=} \emptyset \\
\mathrm{wvc}(\Gamma, x = f(e_1, \ldots, e_n), Q, Q_R) &\stackrel{def}{=} \{ (R[e_i/x_i][x/\backslash\mathbf{result}])^{\#} \Longrightarrow Q\} \\
&\quad \text{mit } \Gamma(f) = \forall x_1, \ldots, x_n. (P, R) \\
\mathrm{wvc}(\Gamma, f(e_1, \ldots, e_n), Q, Q_R) &\stackrel{def}{=} \{ (R[e_i/x_i])^{\#} \Longrightarrow Q\} \\
&\quad \text{mit } \Gamma(f) = \forall x_1, \ldots, x_n. (P, R) \\
\mathrm{wvc}(\Gamma, \{c\ \ c_s\}, Q, Q_R) &\stackrel{def}{=} \mathrm{wvc}(\Gamma, c, \mathrm{awp}(\Gamma, \{c_s\}, Q, Q_R), Q_R) \\
&\quad \cup\ \mathrm{wvc}(\Gamma, \{c_s\}, Q, Q_R) \\
\mathrm{wvc}(\Gamma, \mathbf{if}\ (b)\ c_0\ \ \mathbf{else}\ \ c_1, Q, Q_R) &\stackrel{def}{=} \mathrm{wvc}(\Gamma, c_0, Q, Q_R) \\
&\quad \cup\ \mathrm{wvc}(\Gamma, c_1, Q, Q_R) \\
\mathrm{wvc}(\Gamma, /{**}\ \{q\}\ {*}/, Q, Q_R) &\stackrel{def}{=} \{q \Longrightarrow Q\} \\
\mathrm{wvc}(\Gamma, \mathbf{while}\ (b)\ /{**}\ \mathbf{inv}\ \ i\ {*}/\ \ c, Q, Q_R) &\stackrel{def}{=} \mathrm{wvc}(\Gamma, c, i) \\
&\quad \cup \{i \wedge b^{\#} \Longrightarrow \mathrm{awp}(\Gamma, c, i, Q_R)\} \\
&\quad \cup \{i \wedge\ !\, b^{\#} \Longrightarrow Q\} \\
\mathrm{wvc}(\Gamma, \mathbf{return}\ \ e, Q, Q_R) &\stackrel{def}{=} \emptyset
\end{aligned}$$

Beispiel: swap

```
void swap (int *x, int *y)
/** pre  \valid(*x);
    pre  \valid(*x); */
/** post \old(*x) == *y
        && \old(*y) == *x; */
{
 int z;

 z= *x;
 *x= *y;
 *y= z;
}
```

swap I

```
void swap (int *x, int *y)
/** pre  \valid(*x);
    pre  \valid(*x);  */
/** post \old(*x) == *y
         && \old(*y) == x;  */
{
 int z;

 z= *x;
 *x= *y;
 *y= z;
}

G = [swap |--> \forall x,y. (P = \valid(*x) && \valid(*y),
                             Q = \old(*x) == *y && \old(*y) == x)]

Q# = {\old(*x) == read(s, read(s, y)) && \old(*y) == read(s, read(s, x))}

*****************************************************************************
(A) awp(G, z = *x; *x = *y; *y = z, Q#, Q#)
= awp(G, z = *x, awp(G, *x = *y, awp(G, *y = z, Q#, Q#), Q#), Q#)
// Siehe Einzelberechnungen unten
= { \old(*x) == read(s, read(s, x)) && \old(*y) == read(s, read(s, y)) }


(A.1) awp(G, *y = z, Q#, Q#)
= { let s3=Upd(s2, read(s2, y), read(s2, z));
    in \old(*x) == read(s3, read(s3, y)) && \old(*y) == read(s3, read(s3, x)) }
Da: read(s3, y) = read(Upd(s2, read(s2, y), read(s2, z)), y)
                = read(s3, y) // da y != read(s2, y)
Also: read(s3, read(s3, y)) = read(s3, read(s2, y)) = read(s2, z)
```

swap II

```
= { let s3=Upd(s2, read(s2, y), read(s2, z));
    in \old(*x) == read(s2, z) && \old(*y) == read(s3, read(s3, x)) }
= Q1

(A.2) awp(G, *x = *y, Q1, Q#)
= { let s2=Upd(s1, read(s1, x), read(s1, read(s1, y));
    let s3=Upd(s2, read(s2, y), read(s2, z));
    in \old(*x) == read(s2, z) && \old(*y) == read(s3, read(s3, x)) }
Da: read(s3, x) = read(s2, x) // x != read(s2, y)
    read(s2, x) = read(s1, x) // x != read(s1, x)
    read(s2, z) = read(s1, z) // z != read(s1, x)

= { let s2=Upd(s1, read(s1, x), read(s1, read(s1, y));
    let s3=Upd(s2, read(s2, y), read(s2, z));
    in \old(*x) == read(s1, z) && \old(*y) == read(s3, read(s1, x)) }
= Q2

(A.3) awp(G, z = *x, Q2, Q#)
= { let s1=Upd(s, z, read(s, read(s, x)));
    let s2=Upd(s1, read(s1, x), read(s1, read(s1, y));
    let s3=Upd(s2, read(s2, y), read(s2, z));
    in \old(*x) == read(s1, z) && \old(*y) == read(s3, read(s1, x)) }
Da: read(s1, z) = read(s, read(s, x))
= { let s1=Upd(s, z, read(s, read(s, x)));
    let s2=Upd(s1, read(s1, x), read(s1, read(s1, y));
    let s3=Upd(s2, read(s2, y), read(s2, z));
    in \old(*x) == read(s, read(s, x)) && \old(*y) == read(s3, read(s1, x)) }
Es gilt: read(s2, read(s1, x)) = read(s1, read(s1, x))
```

swap III

```
** Fallunterscheidung **

1) read(s1, y) != read(s1, x):
Dann: read(s3, read(s1, x)) = read(s2, read(s1, x))
Also: read(s2, read(s1, x)) = read(s1, read(s1, y)
Folgt:
  = { let s1=Upd(s, z, read(s, read(s, x)));
      let s2=Upd(s1, read(s1, x), read(s1, read(s1, y));
      let s3=Upd(s2, read(s2, y), read(s2, z));
      in \old(*x) == read(s, read(s, x)) && \old(*y) == read(s1, read(s1, y)) }
Da ausserdem: read(s1, y) != (lokale Variable sind von aussen nicht sichtbar)
Folgt:
  = { let s1=Upd(s, z, read(s, read(s, x)));
      let s2=Upd(s1, read(s1, x), read(s1, read(s1, y));
      let s3=Upd(s2, read(s2, y), read(s2, z));
      in \old(*x) == read(s, read(s, x)) && \old(*y) == read(s, read(s, y)) }

2) read(s1, y) == read(s1, x):
Dann war auch: read(s, y) == read(s, x):
Dann: read(s2, read(s2, y)) = read(s1, read(s1, y)
Dann: read(s3, read(s1, x)) = read(s2, z)
                             = read(s, read(s, x))
                             = read(s, read(s, y))
Folgt (auch wie in 1)
  = { let s1=Upd(s, z, read(s, read(s, x)));
      let s2=Upd(s1, read(s1, x), read(s1, read(s1, y)));
      let s3=Upd(s2, read(s2, y), read(s2, z));
      in \old(*x) == read(s, read(s, x)) && \old(*y) == read(s, read(s, y)) }

******************************************************************************
(B) wvc(G, swap) =
```

swap IV

```
    { P# ==> awp(G, z = *x; *x = *y; *y = z, Q#, Q#)[e_i/\old(e_i)] }
  U wvc(G,  z = *x; *x = *y; *y = z, Q#, Q#)
=   { P# ==> (\old(*x) == read(s, read(s, x)) && \old(*y) == read(s, read(s, y)))
             [read(s, read(s, x))/\old(*x), read(s, read(s, y))/\old(*y)] }
  U wvc(G,  z = *x; *x = *y; *y = z, Q#, Q#)
=   { P# ==> (read(s, read(s, x)) == read(s, read(s, x)) &&
              read(s, read(s, y)) == read(s, read(s, y))) }
  U wvc(G,  z = *x; *x = *y; *y = z, Q#, Q#)
= { True } U wvc(G,  z = *x; *x = *y; *y = z, Q#, Q#)
(Aus B.2 folgt)
= { True }

(B.1) P# = (\valid(*x) && \valid(*y))#
        = \valid(read(s, read(s, x))) && \valid(read(s, read(s, y)))

(B.2) wvc(G,  z = *x; *x = *y; *y = z, Q#, Q#)
=   wvc(G, z = *x, awp(G, *x = *y; *y = z,  Q#, Q#))
  U wvc(G, *x = *y, awp(G, *y = z,  Q#, Q#))
  U wvc(G, *y = z, awp(G, {}, Q#, Q#))
  U wvc(G, {}, Q#, Q#)
=   wvc(G, z = *x, awp(G, *x = *y; *y = z,  Q#, Q#)) [A.2]
  U wvc(G, *x = *y, awp(G, *y = z,  Q#, Q#)) [A.1]
  U wvc(G, *y = z, Q#)
  U {}
Durch (A.1), (A.2)
=   wvc(G, z = *x, Q2)
  U wvc(G, *x = *y, Q1)
  U wvc(G, *y = z, Q#)
= {}
```

Beispiel: `findmax` revisited

```
#include <limits.h>

int findmax(int a[], int a_len)
  /** pre  \array(a, a_len); */
  /** post \forall int i; 0 <= i && i < a_len
                                     --> a[i] <= \result; */
{
  int x; int j;

  x= INT_MIN; j= 0;
  while (j< a_len)
    /* /\**  */ inv \forall int i; 0 <= i && i < j--> a[i]<
    {
    if (a[j]> x) x= a[j];
    j= j+1;
    }
  return x;
}
```

Fazit

- Der Hoare-Kalkül ist viel Schreibarbeitet
- Deshalb haben wir Verifikationsbedingungen berechnet:
  - Approximative schwächste Vorbedingung
- Als nächstes: Approximative stärkste Nachbedingung