# Korrekte Software: Grundlagen und Methoden
# Vorlesung 11 vom 19.06.17: Vorwärtsrechnung 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

# Es geht Vorwärts.

- Verifikation nach dem Hoare-Kalkül mit Zeigern:
  - Viel Schreibarbeit.
- Berechnung von Verifikationsbedingungen:
  - Besser, aber:
  - Es entstehen viele “unbestimmte” Zwischenzustände, die nicht vereinfacht werden können.
- Daher heute Vorwärtsrechnung:
  - Die Vorwärtsregel nach Floyd (für explizite Zustandsprädikate)
  - Berechnung der stärksten Nachbedingung

# Vorwärts?

- Wie kann eine Vorwärtsregel aussehen?
- Alt:

$$\frac{V \notin FV(P)}{\vdash \{P\}\, x = e\, \{\exists V.P[V/x] \,\&\&\, x = e[V/x]\}}$$

- Jetzt: Explizite Zustandsprädikate
- Nachbedingung: $\exists S.\, P[S/\sigma] \,\&\&\, \sigma = upd(S, x^{\dagger}, e^{\#})$
  - $S$ ist der Vorzustand
  - Aber: $x$ und $e$ müssen im Vorzustand $S$ ausgewertet werden!
- Daher nötig: Zustand als zusätzlicher Parameter für $-^{\dagger}$ und $-^{\#}$

# Formal: Konversion in Zustandsprädikate

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

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

$$x^{\dagger} = x^{\dagger}_{\sigma}$$

$$e^{\#} = e^{\#}_{\sigma}$$

- $\sigma$ aktueller Zustand
- $\rho$ initialer Zustand der gerade analysierten Funktion.

# Vorwärts!

- Alternative Zuweisungsregel (nach Floyd):

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

  - $FV(P)$ sind die freien Variablen in $P$.

- Jetzt ist die Vorbedingung offen — Regel kann vorwärts angewandt werden

- Gilt auch für die anderen Regeln

# Das übliche Beispiel I

```
void foo(){
 int x, y, *z;
 /** { True }  */
 z= &x;
 /** \exists S. s= upd(s, z+_S, &x#_S) */
 /** \exists S. s= upd(S, z, x) */
 x= 0;
 /** \exists S'. (\exists S. s= upd(S, z, x))[S'/s] && \sigma= upd(S', x+_S', 0#_S') */
 /** \exists S'. (\exists S. S'= upd(S, z, x)) && \sigma= upd(S', x, 0) */
 /* With \exists x. x= t && P(x) <==> P(t) we get: */
 /** \exists S. \sigma= upd(upd(S, z, x), x, 0) */
 *z= 5;
 /** \exists S'. (\exists S. S'= upd(upd(S, z, x), x, 0)) && \sigma= upd(S', (*z)+_S', 5#_S
 /** \exists S. \sigma= upd(upd(upd(S, z, x), x, 0),
                           read(upd(upd(S, z, x), x, 0), z),    /* this rewrites to x *
                           5) */
 /** \exists S. \sigma= upd(upd(upd(S, z, x), x, 0), x, 5) */
 /** \exists S. \sigma= upd(upd(S, z, x), x, 5) */
 y= x;
 /** \exists S'. (\exists S. S'= upd(upd(S, z, x), x, 5)) && \sigma= upd(S', y+_S', x#_S') *
 /** \exists S. \sigma= upd(upd(upd(S, z, x), x, 5),
                           y,
                           read(upd(upd(S, z, x), x, 5), x)) /* this rewrites to 5 */
 /** \exists S. \sigma= upd(upd(upd(S, z, x), x, 5), y, 5) */
 /** \exists S. \sigma= upd(upd(upd(S, z, x), x, 5), y, 5) && read(\sigma, y) == 5 */
}
```

# Berechnung der stärksten Nachbedingung

- Analog zur schwächsten Vorbedingung berechnen wir die approximative stärkste Nachbedingung $\text{asp}(\Gamma, P, c)$ zusammen mit einer Menge von Verifikationsbedingungen $\text{svc}(\Gamma, P, c)$
- Es gilt:

$$\bigwedge_{p_i \in \text{svc}(\Gamma,P,c)} \forall \sigma.\, p_i \Longrightarrow \models \{P\}\, c\, \{\text{asp}(\Gamma, P, c) \mid \text{asp}(\Gamma, P, c)\}$$

- Zu beachten:
  - **return** beendet den Kontrollfluss
  - Bei Funktionsaufruf: Auswertung der Argumente im Vorzustand

# Berechnung von *awp* und *wvc*

- Ausgehend von Spezifikation mit Vor- und Nachbedingung:

  $$\mathrm{asp}(\Gamma, f(x_1, \ldots, x_n)/\text{** pre } P \text{ post } Q \text{ */ } \{ds\ blk\}) \stackrel{def}{=} \mathrm{asp}(\Gamma', P^{\#}, blk, Q^{\#})$$

  $$\mathrm{svc}(\Gamma, f(x_1, \ldots, x_n)/\text{** pre } P \text{ post } Q \text{ */ } \{ds\ blk\}) \stackrel{def}{=} \mathrm{svc}(\Gamma', P^{\#}, blk, Q^{\#})$$

  $$\Gamma' \stackrel{def}{=} \Gamma[f \mapsto \forall x_1, \ldots, x_n. (P, Q)]$$

- Für diese Form muss jede Funktion mit einem **return** enden.

- Die Verifikationsbedingungen sind implizit über $\sigma$ und $\rho$ allquantifiziert

# Approximative stärkste Nachbedingung

$$
\begin{array}{rcl}
\mathrm{asp}(\Gamma, P, \{\,\}, Q) & \stackrel{def}{=} & P \\
\mathrm{asp}(\Gamma, P, x = e, Q) & \stackrel{def}{=} & \exists S. P[S/\sigma] \;\&\&\; \sigma == upd(S, x^{\dagger}_{S}, e^{\#}_{S}) \\
\mathrm{asp}(\Gamma, P, \{\mathbf{return}\ \ e;\ \ c_s\}, Q) & \stackrel{def}{=} & P \\
\mathrm{asp}(\Gamma, P, \{\mathbf{return};\ \ c_s\}, Q) & \stackrel{def}{=} & P \\
\mathrm{asp}(\Gamma, P, \{c;\ \ c_s\}, Q) & \stackrel{def}{=} & \mathrm{asp}(\Gamma, \mathrm{asp}(\Gamma, P, c, Q), c_s, Q) \\
\mathrm{asp}(\Gamma, P, \mathbf{if}\ (b)\ c_0\ \ \mathbf{else}\ \ c_1, P, Q) & \stackrel{def}{=} & \mathrm{asp}(\Gamma, b^{\#} \;\&\&\; P, c_0, Q) \\
 & & ||\ \mathrm{asp}(\Gamma, !(b^{\#}) \;\&\&\; P, c_1, Q) \\
\mathrm{asp}(\Gamma, P, /{*}{*}\ \{q\}\ {*}/, Q) & \stackrel{def}{=} & q^{\#} \\
\mathrm{asp}(\Gamma, P, \mathbf{while}\ (b)\ /{*}{*}\ \mathbf{inv}\ \ i\ {*}/\ \ c, Q) & \stackrel{def}{=} & i^{\#} \;\&\&\; !\, b^{\#} \\
\mathrm{asp}(\Gamma, P, f(e_1, \ldots, e_n), Q) & \stackrel{def}{=} & \exists S.\, P[S/\sigma] \;\&\&\; R_2[(e_i)^{\#}_{S}/x_i] \\
\mathrm{asp}(\Gamma, P, l = f(e_1, \ldots, e_n), Q) & \stackrel{def}{=} & \exists S.\ \ P[S/\sigma]\&\& \\
 & & \qquad R_2[(e_i)^{\#}_{S}/x_i][l^{\dagger}_{S}/\ \backslash\mathbf{result}] \\
 & & \text{mit } \Gamma(f) = \forall x_1, \ldots, x_n.\, (R_1, R_2)
\end{array}
$$

# Verifikationsbedingungen

$$
\begin{array}{rcl}
\mathrm{svc}(\Gamma, P, \{\,\}, Q) & \stackrel{def}{=} & \emptyset \\
\mathrm{svc}(\Gamma, P, x = e, Q) & \stackrel{def}{=} & \emptyset \\
\mathrm{svc}(\Gamma, P, \{\textbf{return}\ \ e;\ \ c_s\}, Q) & \stackrel{def}{=} & \{P \Longrightarrow Q[e^{\#} / \backslash\textbf{result}]\} \\
\mathrm{svc}(\Gamma, P, \{\textbf{return};\ \ c_s\}, Q) & \stackrel{def}{=} & \{P \Longrightarrow Q\} \\
\mathrm{svc}(\Gamma, P, \{c\ \ c_s\}, Q) & \stackrel{def}{=} & \mathrm{svc}(\Gamma, P, c, Q) \cup \\
 & & \mathrm{svc}(\Gamma, \mathrm{asp}(\Gamma, P, c, Q), \{c_s\}, Q) \\
\mathrm{svc}(\Gamma, P, \textbf{if}\ (b)\ c_0\ \ \textbf{else}\ \ c_1, Q) & \stackrel{def}{=} & \mathrm{svc}(\Gamma, b^{\#}\ \&\&\ P, c_0, Q) \\
 & & \cup\, \mathrm{svc}(!(b^{\#})\ \&\&\ P, c_1, Q) \\
\mathrm{svc}(\Gamma, P, /{*}{*}\ \{q\}\ {*}/, Q) & \stackrel{def}{=} & \{P \Longrightarrow q^{\#}\} \\
\mathrm{svc}(P, \textbf{while}\ \ b\ /{*}{*}\ \ \textbf{inv}\ \ i\ {*}/\ c, Q) & \stackrel{def}{=} & \{P \Longrightarrow i^{\#}\} \\
 & & \cup\, \{\mathrm{asp}(\Gamma, b^{\#}\ \&\&\ i^{\#}, c, Q) \Longrightarrow i^{\#}\} \\
 & & \cup\ \mathrm{svc}(\Gamma, b^{\#}\ \&\&\ i^{\#}, c, Q) \\
\mathrm{svc}(\Gamma, P, f(e_1, \ldots, e_n), Q) & \stackrel{def}{=} & \{P \Longrightarrow R_1[{e_i}^{\#} / x_i]\} \\
\mathrm{svc}(\Gamma, P, l = f(e_1, \ldots, e_n), Q) & \stackrel{def}{=} & \{P \Longrightarrow R_1[{e_i}^{\#} / x_i][l^{\dagger} / \backslash\textbf{result}]\} \\
 & & \hspace{-3em}\text{mit } \Gamma(f) = \forall x_1, \ldots, x_n.\,(R_1, R_2)
\end{array}
$$

# 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;
}
```