

Korrekte Software: Grundlagen und Methoden
Vorlesung 6 vom 09.05.16: Vorwärts und Rückwärts mit Floyd und
Hoare.

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2016

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 ∧ Y = y}
```

```
z = y;
```

```
y = x;
```

```
// {X = y ∧ Y = z}
```

```
x = z;
```

```
// {X = y ∧ Y = x}
```

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}
```

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.

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.

- ▶ Muss das so sein? Ist das immer so?

Vorwärts!

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

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

Vorwärts!

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

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

- ▶ Alternative Zuweisungsregel (nach Floyd):

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

- ▶ $FV(P)$ sind die **freien** Variablen in P .
- ▶ Jetzt ist die Vorbedingung offen— Regel kann vorwärts angewandt werden.

Vorwärtsverkettung

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

- ▶ Ein einfaches Beispiel (nach Mike Gordon):

// {x = 1}

x = x + 1;

// { $\exists V. x = \llbracket x + 1 \rrbracket[V/x] \wedge (x = 1)[V/x]$ }

- ▶ Vereinfachung der Nachbedingung:

$$\exists V. x = \llbracket x + 1 \rrbracket[V/x] \wedge (x = 1)[V/x]$$

$$\longleftrightarrow \exists V. x = (x + 1)[V/x] \wedge (x = 1)[V/x]$$

$$\longleftrightarrow \exists V. x = (V + 1) \wedge (V = 1)$$

$$\longleftrightarrow x = 1 + 1$$

$$\longleftrightarrow x = 2$$

Vorwärtsverkettung

- ▶ Vorwärtsaxiom äquivalent zum Rückwärtsaxiom.
- ▶ In der Anwendung **umständlicher**.
- ▶ Vereinfachung benötigt Lemma:

$$\exists x. P(x) \wedge x = t \longleftrightarrow P(t)$$

- ▶ Vorteile?
 - ▶ Wir wollten doch sowieso die Anwendung automatisieren...

Schwächste Vorbedingung, stärkste Nachbedingung

- ▶ Prädikat P **schwächer** als Q wenn $Q \longrightarrow P$ (**stärker** wenn $P \longrightarrow Q$).
- ▶ Gegeben C0-Programm c , Prädikat P , dann ist
 - ▶ $\text{wp}(c, P)$ die **schwächste Vorbedingung** Q so dass $\models \{Q\} c \{P\}$;
 - ▶ $\text{sp}(P, c)$ die **stärkste Nachbedingung** Q so dass $\models \{P\} c \{Q\}$.

Schwächste Vorbedingung, stärkste Nachbedingung

- ▶ Prädikat P **schwächer** als Q wenn $Q \longrightarrow P$ (**stärker** wenn $P \longrightarrow Q$).
- ▶ Gegeben C0-Programm c , Prädikat P , dann ist
 - ▶ $\text{wp}(c, P)$ die **schwächste Vorbedingung** Q so dass $\models \{Q\} c \{P\}$;
 - ▶ $\text{sp}(P, c)$ die **stärkste Nachbedingung** Q so dass $\models \{P\} c \{Q\}$.
- ▶ Semantische Charakterisierung:

$$\begin{aligned}\models \{P\} c \{Q\} &\iff P \longrightarrow \text{wp}(c, Q) \\ \models \{P\} c \{Q\} &\iff \text{sp}(P, c) \longrightarrow Q\end{aligned}$$

Berechnung von $wp(c, Q)$

- ▶ Einfach für Programme ohne Schleifen:

$$wp(\{\}, P) \stackrel{def}{=} P$$

$$wp(x = e, P) \stackrel{def}{=} P[\llbracket e \rrbracket / x]$$

$$wp(\{c \ c_s\}, P) \stackrel{def}{=} wp(c, wp(\{c_s\}, P))$$

$$wp(\mathbf{if} (b) \ c_0 \ \mathbf{else} \ c_1, P) \stackrel{def}{=} (\llbracket b \rrbracket \wedge wp(c_0, P)) \vee (\neg \llbracket b \rrbracket \wedge wp(c_1, P))$$

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

$$wp(\mathbf{while} (b) \ \{c\}, P) \stackrel{def}{=} (\neg \llbracket b \rrbracket \wedge P) \vee (\llbracket b \rrbracket \wedge wp(c, wp(\mathbf{while} (b) \ \{c\}, P)))$$

- ▶ 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 $\text{awp}(c, Q)$ zusammen mit einer Menge von **Verifikationsbedingungen** $\text{wvc}(c, Q)$
 - ▶ oder die **approximative** stärkste Nachbedingung $\text{asp}(P, c)$ zusammen mit einer Menge von **Verifikationsbedingungen** $\text{svc}(P, c)$
- ▶ Es gilt:

$$\begin{aligned} \bigwedge \text{wvc}(c, Q) &\longrightarrow \models \{\text{awp}(c, Q)\} c \{Q\} \\ \bigwedge \text{svc}(P, c) &\longrightarrow \models \{P\} c \{\text{asp}(P, c)\} \end{aligned}$$

Approximative schwächste Vorbedingung

$$\text{awp}(\{\}, P) \stackrel{\text{def}}{=} P$$

$$\text{awp}(x = e, P) \stackrel{\text{def}}{=} P[\llbracket e \rrbracket / x]$$

$$\text{awp}(\{c \ c_s\}, P) \stackrel{\text{def}}{=} \text{awp}(c, \text{awp}(\{c_s\}, P))$$

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

$$\text{awp}(/** \{q\} */ , P) \stackrel{\text{def}}{=} \llbracket q \rrbracket$$

$$\text{awp}(\mathbf{while} (b) /** \mathbf{inv} \ i */ c, P) \stackrel{\text{def}}{=} \llbracket i \rrbracket$$

$$\text{wvc}(\{\}, P) \stackrel{\text{def}}{=} \emptyset$$

$$\text{wvc}(x = e, P) \stackrel{\text{def}}{=} \emptyset$$

$$\text{wvc}(\{c \ c_s\}, P) \stackrel{\text{def}}{=} \text{wvc}(c, \text{awp}(\{c_s\}, P)) \cup \text{wvc}(\{c_s\}, P)$$

$$\text{wvc}(\mathbf{if} (b) \ c_0 \ \mathbf{else} \ c_1, P) \stackrel{\text{def}}{=} \text{wvc}(c_0, P) \cup \text{wvc}(c_1, P)$$

$$\text{wvc}(/** \{q\} */ , P) \stackrel{\text{def}}{=} \{\llbracket q \rrbracket \longrightarrow P\}$$

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

Approximative stärkste Nachbedingung

$\text{asp}(P, \{ \})$	$\stackrel{\text{def}}{=} P$
$\text{asp}(P, x = e)$	$\stackrel{\text{def}}{=} \exists V. x = \llbracket e \rrbracket[V/x] \wedge P[V/x]$
$\text{asp}(P, \{c \ c_s\})$	$\stackrel{\text{def}}{=} \text{asp}(\text{asp}(P, c), c_s)$
$\text{asp}(P, \mathbf{if} (b) \ c_0 \ \mathbf{else} \ c_1, P)$	$\stackrel{\text{def}}{=} \text{asp}(\llbracket b \rrbracket \wedge P, c_0) \vee \text{asp}(\neg \llbracket b \rrbracket \wedge P, c_1)$
$\text{asp}(P, /** \{q\} */)$	$\stackrel{\text{def}}{=} \llbracket q \rrbracket$
$\text{asp}(P, \mathbf{while} (b) /** \mathbf{inv} \ i */ c)$	$\stackrel{\text{def}}{=} \llbracket i \rrbracket \wedge \neg \llbracket b \rrbracket$
$\text{svc}(P, \{ \})$	$\stackrel{\text{def}}{=} \emptyset$
$\text{svc}(P, x = e)$	$\stackrel{\text{def}}{=} \emptyset$
$\text{svc}(P, \{c \ c_s\})$	$\stackrel{\text{def}}{=} \text{svc}(P, c) \cup \text{svc}(\text{asp}(P, c), \{c_s\})$
$\text{svc}(P, \mathbf{if} (b) \ c_0 \ \mathbf{else} \ c_1)$	$\stackrel{\text{def}}{=} \text{svc}(\llbracket b \rrbracket \wedge P, c_0) \cup \text{svc}(\neg \llbracket b \rrbracket \wedge P, c_1)$
$\text{svc}(P, /** \{q\} */)$	$\stackrel{\text{def}}{=} \{P \longrightarrow \llbracket q \rrbracket\}$
$\text{svc}(P, \mathbf{while} \ b /** \mathbf{inv} \ i */ c)$	$\stackrel{\text{def}}{=} \{P \longrightarrow \llbracket i \rrbracket\}$ $\cup \{\text{asp}(\llbracket b \rrbracket \wedge \llbracket i \rrbracket, c) \longrightarrow \llbracket i \rrbracket\}$ $\cup \text{svc}(\llbracket b \rrbracket \wedge \llbracket i \rrbracket, c)$

Beispiel: das Fakultätsprogramm

- ▶ In der Praxis sind Vor- und Nachbedingung gegeben, und nur die Verifikationsbedingungen relevant:

Sei F das annotierte Fakultätsprogramm:

```
// {0 <= N}
p= 1;
c= 1;
while (c <= n) /** inv p == fac(c-1) && c-1 <= N; */ {
    p = p * c;
    c = c + 1;
}
// {p == fac(N)}
```

- ▶ Berechnung vorwärts: $\text{svc}(F, \top)$
- ▶ Berechnung rückwärts: $\text{wvc}(\top, F)$

Zusammenfassung

- ▶ Die Zuweisungsregel gibt es “rückwärts” und “vorwärts”.
- ▶ Bis auf die Invarianten an Schleifen können wir Korrektheit automatisch prüfen.

Rückwärtsberechnung:

- ▶ Einfacher zu berechnen
- ▶ Führt zu großen Formeln
- ▶ Keine Möglichkeit, Zwischenzustände zu vereinfachen

Vorwärtsberechnung:

- ▶ Entspricht **symbolischer Ausführung**
 - ▶ Umständlichere Berechnung der Verifikationsbedingungen
 - ▶ Erlaubt **zustandsbasierte Vereinfachung** (z.B. Entfernen unerreichbarer Fälle)
- ▶ Die Generierung von Verifikationsbedingungen korrespondiert zur **relativen Vollständigkeit** der Floyd-Hoare-Logik — nächste Woche.