Korrekte Software: Grundlagen und Methoden
Vorlesung 9 vom 5/12.06.24
Verifikationsbedingungen

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2024

Fahrplan

Idee

- Hier ist ein einfaches Programm:

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

- Wir sehen:
  1. Die Verifikation erfolgt **rückwärts** (von hinten nach vorne).
  2. Die Vorbedingung kann **berechnet** werden.
- Geht das immer?
- Was bringt uns das?

Rückwärtsanwendung der Regeln

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

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

- Was ist mit den anderen Regeln?

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

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

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

Arbeitsblatt 9.1: Eine Kleine Fallunterscheidung

Berechnet die Vorbedingungen an den Stellen (A), (B), (?) für folgendes Programm:

```
// (?)
if (y == 7) {
  // (A)
  x= 3;
  //
  }
else {
  // (B)
  y= 0;
  //
  x= 10;
  //
  }
// {x + y == 10}
```


Rückwärtsanwendung: if

```
// ?
if (b) {
  // {P1}
  ...
  // {Q}
  }
else {
  // {P2}
  ...
  // {Q}
  }
// {Q}
```

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

Regel in der Form nicht geeignet. Besser:

$$A \stackrel{def}{=} (P_1 \wedge b) \vee (P_2 \wedge \neg b)$$
$$A \wedge b \Longrightarrow P_1 \quad (1)$$
$$A \wedge \neg b \Longrightarrow P_2 \quad (2)$$

Kombiniert mit Weakening ergibt neue Regel:

$$\frac{\dfrac{A \wedge b \Longrightarrow P_1 \quad \vdash \{P_1\}\ c_0\ \{B\}}{\vdash \{A \wedge b\}\ c_0\ \{B\}} \quad \dfrac{A \wedge \neg b \Longrightarrow P_2 \quad \vdash \{P_2\}\ c_1\ \{B\}}{\vdash \{A \wedge \neg b\}\ c_1\ \{B\}}}{\vdash \underbrace{\{(P_1 \wedge b) \vee (P_2 \wedge \neg b)\}}_{A}\ \textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1\ \{B\}}$$

Arbeitsblatt 9.2: Etwas Logik

Beweist Lemma (1) und (2) von der vorherigen Folie:

$$A = (P_1 \wedge b) \vee (P_2 \wedge \neg b)$$
$$A \wedge b \Longrightarrow P_1 \quad (1)$$
$$A \wedge \neg b \Longrightarrow P_2 \quad (2)$$

Hinweis:

- Nutzt logische Umformungen: Distributivität, Idempotenz, ...

Neue Regeln

- Wir können aus dem Hoare-Kalkül **neue Regeln** ableiten, in dem wir
  1. Existierende Regeln **instantiieren**, oder
  2. existierende Regeln **verknüpfen**.
- Wir benötigen das hier, um die Regeln des Hoare-Kalkül in eine Form zu bringen, welche die Rückwärtsrechnung ermöglicht.

Das Hinzufügen abgeleiteter Regeln ist eine **konservative Erweiterung** — es lassen sich damit nicht mehr oder weniger Hoare-Tripel $\vdash \{P\}\ c\ \{Q\}$ herleiten.

Regeln für die Rückwärtsrechnung

1. **Nachbedingung** der **Konklusion** ist von der Form $\{Q\}$ (**offene** Meta-Variable)
2. Alle **Vorbedingungen** der **Prämissen** ist von der Form $\{P_i\}$ (**unterschiedliche** P_i)
3. Alle Variablen in den Vorbedingungen der Konklusion, den Weakenings und Nachbedingungen der Prämisse sind **determiniert**[1].

Welche Regeln passen noch nicht? **while**-Regel passt noch nicht ...

[1] **Entweder** in der Nachbedingung oder dem Programmausdruck der Konklusion, **oder** den Vorbedingungen der Prämisse enthalten.

Regeln für die Rückwärtsrechnung: while

- **while**-Regel (3) wird mit Weakening zu (4):

$$\frac{\vdash \{I \wedge b\}\, c\, \{I\}}{\vdash \{I\}\ \textbf{while}\ (b)\ c\, \{I \wedge \neg b\}} \quad (3)$$

$$\frac{I \wedge b \Longrightarrow R \qquad \vdash \{R\}\, c\, \{I\} \qquad I \wedge \neg b \Longrightarrow Q}{\vdash \{I\}\ \textbf{while}\ (b)\ c\, \{Q\}} \quad (4)$$

- Implikationen $I \wedge b \Longrightarrow R$, $I \wedge \neg b \Longrightarrow Q$ werden zu **Beweisverpflichtungen**
- Bedingung I (**Invariante**) muss **vorgegeben** werden.
  - Durch **Annotation**: **while** (b) /** **inv** I */ c

Übersicht: Regeln für den Hoare-Kalkül Rückwärts

$$\frac{}{\vdash \{P[e/l]\}\ l = e\, \{P\}} \qquad \frac{}{\vdash \{A\}\, \{\,\}\, \{A\}} \qquad \frac{\vdash \{A\}\, c_1\, \{B\} \vdash \{B\}\, c_2\, \{C\}}{\vdash \{A\}\, c_1; c_2\, \{C\}}$$

$$\frac{\vdash \{A_0\}\, c_0\, \{B\} \quad \vdash \{A_1\}\, c_1\, \{B\}}{\vdash \{(A_0 \wedge b) \vee (A_1 \wedge \neg b)\}\ \textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1\, \{B\}}$$

$$\frac{I \wedge b \Longrightarrow B \quad \vdash \{B\}\, c\, \{I\} \quad I \wedge \neg b \Longrightarrow C}{\vdash \{I\}\ \textbf{while}\ (b)\ /{*}{*}\ \textbf{inv}\ I\ {*}/\ c\, \{C\}}$$

Der Hoare-Kalkül Rückwärts

- Mit diesen Regeln können wir für ein gegebenes Programm c und eine Nachbedingung Q eine Vorbedingung P_{pre} **berechnen**.
- Was bringt uns das?
  - Um ein **Hoare-Tripel** $\vdash \{P\}\, c\, \{Q\}$ zu **beweisen**, berechnen wir wie oben die **Vorbedingung** P_{pre}.
  - Jetzt müssen wir nur noch $P \Longrightarrow P_{\text{pre}}$ und alle Weakenings aus der Berechnung von P_{pre} zeigen.
  - Damit **reduzieren** wir die **Korrektheit** auf eine Menge von (zustandsfreien) **Beweisverpflichtungen**.

Berechnung von Vorbedingungen

- Die Rückwärtsrechnung von einer gegebenen Nachbedingung entspricht der Berechnung einer Vorbedingung.
- Gegeben C0-Programm c, Prädikat Q, dann ist
  - $\text{wp}(c, Q)$ 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 $Q \in \textbf{Assn}$ und Programm $c \in \textbf{Stmt}$, dann

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

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

Annotierte Programme

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

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

Überblick: Approximative schwächste Vorbedingung

$$\begin{aligned}
\text{awp}(\{\,\}, P) &\stackrel{\text{def}}{=} P \\
\text{awp}(l = e, P) &\stackrel{\text{def}}{=} P[e/l] \qquad (\textbf{Genauer}: \text{Folie 24 letzte VL}) \\
\text{awp}(c_1; c_2, P) &\stackrel{\text{def}}{=} \text{awp}(c_1, \text{awp}(c_2, P)) \\
\text{awp}(\textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1, P) &\stackrel{\text{def}}{=} (b \wedge \text{awp}(c_0, P)) \vee (\neg b \wedge \text{awp}(c_1, P)) \\
\text{awp}(\textbf{while}\ (b)\ /{*}{*}\ \textbf{inv}\ i\ {*}/\ c, P) &\stackrel{\text{def}}{=} i \\
\\
\text{wvc}(\{\,\}, P) &\stackrel{\text{def}}{=} \emptyset \\
\text{wvc}(l = e, P) &\stackrel{\text{def}}{=} \emptyset \\
\text{wvc}(c_1; c_2, P) &\stackrel{\text{def}}{=} \text{wvc}(c_1, \text{awp}(c_2, P)) \cup \text{wvc}(c_2, P) \\
\text{wvc}(\textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1, P) &\stackrel{\text{def}}{=} \text{wvc}(c_0, P) \cup \text{wvc}(c_1, P) \\
\text{wvc}(\textbf{while}\ (b)\ /{*}{*}\ \textbf{inv}\ i\ {*}/\ c, P) &\stackrel{\text{def}}{=} \text{wvc}(c, i) \cup \{i \wedge b \longrightarrow \text{awp}(c, i)\} \cup \{i \wedge \neg b \longrightarrow P\} \\
\text{wvc}(\{P\}\, c\, \{Q\}) &\stackrel{\text{def}}{=} \{P \longrightarrow \text{awp}(c, Q)\} \cup \text{wvc}(c, Q)
\end{aligned}$$

Berechnung der Verifikationsbedingungen

Programmkorrektheit

- Gegeben: Annotiertes Programm c mit Vorbedingung P und Nachbedingung Q.
- Gesucht: $\text{wvc}(\{P\}\, c\, \{Q\})$

1. Rekursiv von der Nachbedigung ausgehend berechnen wir für jede Zeile des Programmes die gültige approximative Vorbedingung $\text{awp}(c, -)$.
2. Dabei notieren wir alle auftretenden Verifikationsbedingungen $\text{wvc}(c, -)$
3. Dabei werden **keine** Vereinfachungen vorgenommen.

Beispiel: das Fakultätsprogramm

- 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 ∧ 0 < c} */
5  { p = p * c;
6    c = c + 1;
7    }
8  // {p = n!}
```

- Berechnung der Verifikationsbedingungen zur Nachbedingung wvc$(F, p = n!)$

Notation für Verifikationsbedingungen

AWP wird am Programm annotiert:

```
1  // {0 ≤ n}
2  // {1 = (1 − 1)! ∧ 1 − 1 ≤ n ∧ 0 < 1}
3  p= 1;
4  // {p = (1 − 1)! ∧ 1 − 1 ≤ n ∧ 0 < 1}
5  c= 1;
6  // {p = (c − 1)! ∧ c − 1 ≤ n ∧ 0 < c}
7  while (c <= n)
8    /** inv p= (c−1)! ∧ c−1 ≤ n ∧ 0<c*/ {
9    // {p·c = ((c+1)−1)! ∧ (c+1)−1 ≤ n ∧ 0 < c+1}
10   p = p * c;
11   // {p = ((c+1)−1)! ∧ (c+1)−1 ≤ n ∧ 0 < c+1}
12   c = c + 1;
13   // {p = (c − 1)! ∧ c − 1 ≤ n ∧ 0 < c}
14   }
15 // {p = n!}
```

WVC wird daneben notiert:

1 | $p = (c-1)! \land c - 1 \leq n \land 0 < c \land \neg(c \leq n) \longrightarrow p = n!$

2 | $p = (c-1)! \land c - 1 \leq n \land 0 < c \land c \leq n \longrightarrow p \cdot c = ((c+1)-1)! \land (c+1) - 1 \leq n \land 0 < c+1$

3 | $0 \leq n \longrightarrow 1 = (1-1)! \land (1-1) \leq n \land 0 < 1$

Vereinfachung von Verifikationsbedingungen

Wir nehmen folgende **strukturelle Vereinfachungen** vor:

1. Konjunktionen in der Konklusion werden zu einzelnen Verifikationsbedingungen
   - Bsp: $A_1 \land A_2 \land A_3 \longrightarrow P \land Q \rightsquigarrow A_1 \land A_2 \land A_3 \longrightarrow P, A_1 \land A_2 \land A_3 \longrightarrow Q$
2. Auswertung konstanter arithmetischer Ausdrücke, einfache arithmetische Gesetze
   - Bsp. $(x+1) - 1 \rightsquigarrow x$, $1 - 1 \rightsquigarrow 0$
3. Normalisierung der Relationen (zu $<, \leq, =, \neq$) und Vereinfachung
   - Bsp: $\neg(x \leq y) \rightsquigarrow x > y \rightsquigarrow y < x$, $x \leq x \rightsquigarrow \mathit{true}$, $4 \leq 5 \rightsquigarrow \mathit{true}$
4. Alle Bedingungen mit einer Prämisse *false* oder einer Konklusion *true* sind trivial erfüllt.

Vereinfachung am Beispiel

- 1: $p = (c-1)! \land c - 1 \leq n \land 0 < c \land \neg(c \leq n) \longrightarrow p = n!$
  $\rightsquigarrow p = (c-1)! \land c - 1 \leq n \land 0 < c \land n < c \longrightarrow p = n!$
- 2: $p = (c-1)! \land c - 1 \leq n \land 0 < c \land c \leq n \longrightarrow p \cdot c = ((c+1)-1)! \land (c+1) - 1 \leq n \land 0 < c+1$
  $\rightsquigarrow p = (c-1)! \land c - 1 \leq n \land 0 < c \land c \leq n \longrightarrow p \cdot c = c!$
  $p = (c-1)! \land c - 1 \leq n \land 0 < c \land c \leq n \longrightarrow c \leq n \rightsquigarrow \mathit{true}$
  $p = (c-1)! \land c - 1 \leq n \land 0 < c \land c \leq n \longrightarrow 0 < c + 1$
- 3: $0 \leq n \longrightarrow 1 = (1-1)! \land (1-1) \leq n$
  $\rightsquigarrow 0 \leq n \longrightarrow 1 = 0!$
  $0 \leq n \longrightarrow 0 \leq n \rightsquigarrow \mathit{true}$

Es bleibt zu zeigen

- 1: $p = (c-1)! \land c - 1 \leq n \land 0 < c \land n < c \longrightarrow p = n!$
  Aus $n < c$ folgt $n \leq c - 1$, also $c - 1 = n$, und mit $p = (c-1)!$ folgt die Behauptung.
- 2: $p = (c-1)! \land c - 1 \leq n \land 0 < c \land c \leq n \longrightarrow p \cdot c = c!$
  Aus $p = (c-1)!$ folgt $p \cdot c = c \cdot (c-1)!$, und mit $0 < c$ und $c \cdot (c-1)! = c!$ folgt die Behauptung.
- 3: $p = (c-1)! \land c - 1 \leq n \land 0 < c \land c \leq n \longrightarrow 0 < c + 1$
  Aus $0 < c$ folgt $0 < c + 1$
- 4: $1 = 0!$ folgt direkt aus der Definition der Fakultät.

Arbeitsblatt 9.3: Da summt was...

```
1  // {0 ≤ n ∧ n = N}
2  p= 0;
3  while (n>0) //** inv p= sum(n+1,N); */
4  { p = p + n;
5    n = n − 1;
6    }
7  // {p = sum(1, N)}
```

1. Berechnet zuerst die **unvereinfachten** VCs (dafür sind die AWPs nötig)
2. Danach vereinfacht die VCs **schematisch** wie oben beschrieben.
3. Welche VCs sind beweisbar?

Dabei gilt: $\mathit{sum}(i,j) = \begin{cases} 0 & i > j \\ i + \mathit{sum}(i+1, j) & i \leq j \end{cases}$

Weiteres Beispiel: Maximales Element

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


Maximales Element (Schleifenrumpf)

```
1  while (i< n)
                                  φ(i,r)
2    /** inv {(∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n ∧ 0 ≤ i ≤ n}
3  */
4  {
5    // {a[r] < a[i] ∧ φ(i+1, i) ∨ (¬(a[r] < a[i]) ∧ φ(i+1, r))}
6    if (a[r] < a[i]) {
7      // {φ(i+1, i)}
8      r= i;
9      // {φ(i+1, r)}
10     }
11   else {
12     // {φ(i+1, r)}
13   }
14   // {φ(i+1, r)}
15   i= i+1;
16   // {φ(i, r)}
17   }
18 // {(∀j. 0 ≤ j < n ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

VC:

1 | $\varphi(i,r) \land \neg(i < n) \longrightarrow (\forall j.\, 0 \leq j < n \longrightarrow a[j] \leq a[r]) \land 0 \leq r < n$

2 | $\varphi(i,r) \land i < n \longrightarrow a[r] < a[i] \land \varphi(i+1, i) \lor \neg(a[r] < a[i]) \land \varphi(i+1, r)$

Maximales Element (Initialisierung)

```
1 // {0 < n}
2 // {φ(0,0)}
3 i= 0;
4 // {φ(i,0)}
5 r= 0;
6 // {φ(i,r)}
7 while (i< n)
8   /** inv {(∀j.0 ≤ j < i ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n ∧ 0 ≤ i < n}  φ(i,r)
    */
```

VC:

1 $\varphi(i,r) \wedge \neg(i < n) \longrightarrow$
$(\forall j.\, 0 \le j <\longrightarrow a[j] \le a[r])$
$\wedge$
$0 \le r < n$

2 $\varphi(i,r) \wedge i < n \longrightarrow$
$a[r] < a[i] \wedge \varphi(i+1, i)$
$\vee$
$\neg(a[r] < a[i]) \wedge \varphi(i+1, r)$

3 $0 \le n \longrightarrow \varphi(0,0)$

Maximales Element (Verifikationsbedingungen)

Unvereinfacht:

1 $(\forall j.0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i \le n \wedge \neg(i < n) \longrightarrow$
$(\forall j.\, 0 \le j <\longrightarrow a[j] \le a[r]) \wedge 0 \le r < n$

2 $(\forall j.0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i \le n \wedge i < n \longrightarrow$
$((a[r] < a[i] \wedge (\forall j.0 \le j < i+1 \longrightarrow a[j] \le a[i]) \wedge 0 \le i < n \wedge 0 \le i+1 \le n) \vee$
$(\neg(a[r] < a[i]) \wedge (\forall j.0 \le j < i+1 \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i+1 \le n))$

3 $0 \le n \longrightarrow (\forall j.0 \le j < 0 \longrightarrow a[j] \le a[0]) \wedge 0 \le 0 < n \wedge 0 \le 0 \le n$

Vereinfacht:

1.1 $(\forall j.0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i \le n \wedge n \le i \longrightarrow$
$\forall j.0 \le j < n \longrightarrow a[j] \le a[r]$

1.2 $(\forall j.0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i \le n \wedge n \le i \longrightarrow 0 \le r \le n$

2 $(\forall j.0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i \le n \wedge i < n \longrightarrow$
$((a[r] < a[i] \wedge (\forall j.0 \le j < i+1 \longrightarrow a[j] \le a[i]) \wedge 0 \le i < n \wedge 0 \le i+1 \le n) \vee$
$(\neg(a[i] \le a[r]) \wedge (\forall j.0 \le j < i+1 \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i+1 \le n))$

3.1 $0 \le n \longrightarrow \forall j.0 \le j < 0 \longrightarrow a[j] \le a[0]$

3.2 $0 \le n \longrightarrow 0 \le 0 < 0$

3.2 $0 \le n \longrightarrow 0 \le 0 \le 0$

- Sehr **lange** Verifikationsbedingungen (u.a. wegen Fallunterscheidung)
  - Insbesondere schwer zu **vereinfachen**
- Wie können wir das **beheben**?

Explizite Vorbedingungen

Lange Vorbedingung:

```
// {(P1 ∧ b) ∨ (P2 ∧ ¬b)}
if (b) {
  // {P1}
  ...
  // {Q}
  } else {
  // {P2}
  ...
  // {Q}
  }
```

Kurze Vorbedingung:

```
// {A}
if (b) {
  // {A ∧ b}
  ...
  // {Q}
  } else {
  // {A ∧ ¬b}
  ...
  // {Q}
  }
```

Dazu VCs:

$$A \wedge b \longrightarrow P_1$$
$$A \wedge \neg b \longrightarrow P_2$$

Spracherweiterung: Explizite Spezifikationen

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

Assn $a ::= \ldots$ — Zusicherungen

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

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

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

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

$$\begin{aligned}
\text{awp}(\{\}, P) &\stackrel{def}{=} P \\
\text{awp}(l = e, P) &\stackrel{def}{=} P[e/l] \qquad \text{(Genauer: Folie 24 letzte VL)} \\
\text{awp}(c_1; c_2, P) &\stackrel{def}{=} \text{awp}(c_1, \text{awp}(c_2, P)) \\
\text{awp}(\textbf{if } (b)\ c_0\ \textbf{else}\ c_1, P) &\stackrel{def}{=} Q \quad \text{wenn } \text{awp}(c_0, P) = b \wedge Q, \text{awp}(c_1, P) = \neg b \wedge Q \\
\text{awp}(\textbf{if } (b)\ c_0\ \textbf{else}\ c_1, P) &\stackrel{def}{=} (b \wedge \text{awp}(c_0, P)) \vee (\neg b \wedge \text{awp}(c_1, P)) \\
\text{awp}(\text{/** } \{q\} \text{ */}, P) &\stackrel{def}{=} q \\
\text{awp}(\textbf{while } (b)\ \text{/** } \textbf{inv } i \text{ */ } c, P) &\stackrel{def}{=} i \\
\text{wvc}(\{\}, P) &\stackrel{def}{=} \emptyset \\
\text{wvc}(l = e, P) &\stackrel{def}{=} \emptyset \\
\text{wvc}(c_1; c_2, P) &\stackrel{def}{=} \text{wvc}(c_1, \text{awp}(c_2, P)) \cup \text{wvc}(c_2, P) \\
\text{wvc}(\textbf{if } (b)\ c_0\ \textbf{else}\ c_1, P) &\stackrel{def}{=} \text{wvc}(c_0, P) \cup \text{wvc}(c_1, P) \\
\text{wvc}(\text{/** } \{q\} \text{ */}, P) &\stackrel{def}{=} \{q \longrightarrow P\} \\
\text{wvc}(\textbf{while } (b)\ \text{/** } \textbf{inv } i \text{ */ } c, P) &\stackrel{def}{=} \text{wvc}(c, i) \cup \{i \wedge b \longrightarrow \text{awp}(c, i)\} \cup \{i \wedge \neg b \longrightarrow P\}
\end{aligned}$$

Maximales Element mit Zusicherung

```
1  // {0 < n}
2  // {(∀j.0 ≤ j < 0 ⟶ a[j] ≤ a[0]) ∧ 0 ≤ 0 < 0 ∧ 0 ≤ n}
3  i= 0;
4  r= 0;
5  // {(∀j.0 ≤ j < i ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < i ∧ i ≤ n}
6  while (i< n) /** inv {(∀j.0 ≤ j < i ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n ∧ 0 ≤ i ≤ n} */
7  {
8    // {∀j.0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < i ∧ i < n}
9    if (a[r] < a[i]) {
10     // {(∀j.0 ≤ j < i ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n ∧ 0 ≤ i < n ∧ a[r] < a[i]}
11     // {(∀j.0 ≤ j < i+1 ⟶ a[j] ≤ a[i]) ∧ 0 ≤ i < n ∧ 0 ≤ i+1 ≤ n}
12     r= i;
13     // {(∀j.0 ≤ j < i+1 ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n ∧ 0 ≤ i+1 ≤ n}
14     }
15   else {
16     // {(∀j.0 ≤ j < i ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n ∧ 0 ≤ i < n ∧ ¬(a[r] < a[i])}
17     // {(∀j.0 ≤ j < i+1 ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n ∧ 0 ≤ i+1 ≤ n}
18     }
19   // {(∀j.0 ≤ j < i+1 ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n ∧ 0 ≤ i+1 ≤ n}
20   i= i+1;
21   // {(∀j.0 ≤ j < i ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n ∧ 0 ≤ i ≤ n}
22   }
23 // {(∀j.0 ≤ j < n ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```


Maximales Element mit Zusicherung: Beweisverpflichtungen

Unvereinfacht:

(1) $(\forall j.\, 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i \le n \wedge \neg(i < n)$
$\longrightarrow (\forall j.\, 0 \le j < n \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n$

(2) $(\forall j.\, 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i < n \wedge \neg(a[r] < a[i])$
$\longrightarrow (\forall j.\, 0 \le j < i+1 \longrightarrow a[j] \le a[r]) \wedge 0 \le r < i+1 \wedge 0 \le i+1 \le n$

(3) $(\forall j.\, 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i < n \wedge a[r] < a[i]$
$\longrightarrow (\forall j.\, 0 \le j < i+1 \longrightarrow a[j] \le a[i]) \wedge 0 \le i < n \wedge 0 \le i+1 \le n$

(4) $(\forall j.\, 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i \le n \longrightarrow$
$(\forall j.\, 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i < n$

(5) $0 < n \longrightarrow (\forall j.\, 0 \le j < 0 \longrightarrow a[j] \le a[0]) \wedge 0 \le 0 < n \wedge 0 \le 0 \le n$

Maximales Element mit Zusicherung: Beweisverpflichtungen

Vereinfacht (Teil 1):

(1.1) $(\forall j.\, 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i \le n \wedge n \le i$
$\longrightarrow (\forall j.\, 0 \le j < n \longrightarrow a[j] \le a[r])$

(1.2) $(\forall j.\, 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i \le n \wedge n \le i$
$\longrightarrow 0 \le r < n$

(2.1) $(\forall j.\, 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i < n \wedge a[i] \le a[r]$
$\longrightarrow (\forall j.\, 0 \le j < i+1 \longrightarrow a[j] \le a[r])$

(2.2) $(\forall j.\, 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i < n \wedge a[i] \le a[r]$
$\longrightarrow 0 \le r < n$

(2.3) $(\forall j.\, 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i < n \wedge a[i] \le a[r]$
$\longrightarrow 0 \le i+1 \le n$

Maximales Element mit Zusicherung: Beweisverpflichtungen

Vereinfacht (Teil 2):

(3.1) $(\forall j.\, 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i < n \wedge a[r] < a[i] \longrightarrow (\forall j.\, 0 \le j < i+1 \longrightarrow a[j] \le a[i])$

(3.2) $(\forall j.\, 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i < n \wedge a[r] < a[i] \longrightarrow 0 \le i < n$

(3.3) $(\forall j.\, 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i < n \wedge a[r] < a[i] \longrightarrow 0 \le i+1 \le n$

(4.1) $(\forall j.\, 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i \le n \longrightarrow (\forall j.\, 0 \le j < i \longrightarrow a[j] \le a[r])$

(4.2) $(\forall j.\, 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i \le n \longrightarrow 0 \le r < n$

(4.3) $(\forall j.\, 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n \wedge 0 \le i \le n \longrightarrow 0 \le i < n$

(5.1) $0 < n \longrightarrow (\forall j.\, 0 \le j < 0 \longrightarrow a[j] \le a[0])$

(5.2) $0 < n \longrightarrow 0 \le 0 < n$

(5.3) $0 < n \longrightarrow 0 \le 0 \le n$

Beweismethoden

- Um $P_1 \wedge \ldots \wedge P_n \longrightarrow Q$ zu zeigen, nehmen wir $P_1, \ldots, P_n$ an und zeigen Q.
- Dabei nutzen wir **u.a.** folgende Regeln:

Wenn P, dann P (Trivial)
Wenn P und $I = t$, dann $P[t/I]$ (Substitution)
$x \le x$ (Reflexität)
Wenn $x \le y$ und $y \le z$, dann $x \le z$ (Transitivität)
Wenn $x \le y$ und $y \le x$, dann $x = y$ (Antisymmetrie)
Wenn $x < y$, dann $x \le y + 1$ oder $x + 1 \le y$ (Inc)
Wenn $\forall x.\, P$, dann $P[t/x]$ (Instantiierung)
Wenn *false*, dann P (Ex falso)
Wenn $a \le b$ und $x \le y$, dann $a + x \le b + y$ und Variation mit $x = 0$ etc.
Umformungen mit $(0, +)$ und $(1, \cdot)$
Domänenspezifische Regeln

Arbeitsblatt 9.4: Beweisverpflichtungen Beweisen

Betrachtet die vereinfachten Verifikationsbedingungen. Wie würdet ihr sie beweisen? Was für Methoden verwendet ihr?

- Welches sind **triviale** Beweise?
- Welches sind **einfache** Beweise?
- Welche erfordern längere Argumentation?

Arbeitsblatt 9.5: Kopien

Dieses Programm kopiert ein Array:

```
i= 0;
while (i< m)
  /** inv ??? */ {
  b[m-1-i]= a[i];
  i= i+1;
  }
```

1. Spezifiziert die Funktionalität.
2. Findet die Invariante.
3. Berechnet die Verifikationsbedingungen (VCs) und schwächste Vorbedingung.
4. Beweist die VCs.

Noch ein Beispiel: kleinstes Null-Element

- Folgendes Programm sucht in `a` nach dem **ersten** Null-Element:

```
void find_zero()
{
i= 0;
r= -1;
while (i< n)
  if (r== -1 && a[i] == 0) {
    r= i;
    }
  else {
    }
  }
}
```

- Wie **spezifizieren** wir das?
- Welche **Beweisverpflichtungen** entstehen?
- Wie **beweisen** wir diese?

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.