Korrekte Software: Grundlagen und Methoden

Vorlesung 8 vom 11.6.20

Verifikationsbedingungen

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2020

Fahrplan

- Einführung
- Operationale Semantik
- Denotationale Semantik
- Äquivalenz der Operationalen und Denotationalen Semantik
- Der Floyd-Hoare-Kalkül
- Invarianten und die Korrektheit des Floyd-Hoare-Kalküls
- Strukturierte Datentypen
- Verifikationsbedingungen
- Vorwärts mit Floyd und Hoare
- Modellierung
- Spezifikation von Funktionen
- Referenzen und Speichermodelle
- 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 ∧ 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.

- Geht das immer?

Rückwärtsanwendung der Regeln

- Zuweisungsregel kann **rückwärts** angewandt werden, weil die Nachbedingung eine offene Variable ist — P passt auf jede beliebige Nachbedingung (siehe “Definition” Folie 24 der letzten Vorlesung)

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

Rückwärtsanwendung der Regeln

- Zuweisungsregel kann **rückwärts** angewandt werden, weil die Nachbedingung eine offene Variable ist — P passt auf jede beliebige Nachbedingung (siehe “Definition” Folie 24 der letzten Vorlesung)

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

- Was ist mit den anderen Regeln?

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

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

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

Rückwärtsanwendung der Regeln

- Zuweisungsregel kann **rückwärts** angewandt werden, weil die Nachbedingung eine offene Variable ist — P passt auf jede beliebige Nachbedingung (siehe “Definition” Folie 24 der letzten Vorlesung)

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

- Was ist mit den anderen Regeln? Nur **while** macht Probleme!

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

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

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

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:

$$
\begin{aligned}
\mathrm{wp}(\{\,\}, P) &\stackrel{def}{=} P \\
\mathrm{wp}(I = e, P) &\stackrel{def}{=} P[e/I] \quad \text{(Genauer: Folie 24 letzte VL)} \\
\mathrm{wp}(c_1; c_2, P) &\stackrel{def}{=} \mathrm{wp}(c_1, \mathrm{wp}(c_2, P)) \\
\mathrm{wp}(\textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1, P) &\stackrel{def}{=} (b \land \mathrm{wp}(c_0, P)) \lor (\neg b \land \mathrm{wp}(c_1, P))
\end{aligned}
$$

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

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

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

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

Approximative schwächste Vorbedingung

- Für die **while**-Schleife:

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

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

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

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

Überblick: Approximative schwächste Vorbedingung

$$
\begin{aligned}
\mathrm{awp}(\{\,\}, P) &\stackrel{def}{=} P \\
\mathrm{awp}(l = e, P) &\stackrel{def}{=} P[l/x] \quad \text{(Genauer: Folie 24 letzte VL)} \\
\mathrm{awp}(c_1; c_2, P) &\stackrel{def}{=} \mathrm{awp}(c_1, \mathrm{awp}(c_2, P)) \\
\mathrm{awp}(\textbf{if }(b)\ c_0\ \textbf{ else }\ c_1, P) &\stackrel{def}{=} (b \land \mathrm{awp}(c_0, P)) \lor (\neg b \land \mathrm{awp}(c_1, P)) \\
\mathrm{awp}(\textbf{while }(b)\ /\!/{*}{*}\ \textbf{inv}\ i\ {*}/\ c, P) &\stackrel{def}{=} i
\end{aligned}
$$

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

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

Beispiel: das Fakultätsprogramm

- In der Praxis sind Vorbedingung gegeben, und nur die Verifikationsbedingungen relevant.

- Sei F das annotierte Fakultätsprogramm:

```
// {0 ≤ n}
p= 1;
c= 1;
while (c <= n) //** inv {p = (c − 1)! ∧ c − 1 ≤ n} */
{ p = p * c;
  c = c + 1;
  }
// {p = n!}
```

- Berechnung der Verifikationsbedingungen zur Nachbedingung.

Notation für Verifikationsbedingungen

```
// {0 ≤ n}
p= 1;
c= 1;
while (c <= n) //** inv {p = (c − 1)! ∧ c − 1 ≤ n}; */
{ p = p * c;
  c = c + 1;
  }
// {p = n!}
```

AWP 6

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	

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	$p \times c = ((c+1)-1)! \wedge ((c-1)+1) \leq n$
	4	

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

$$
\begin{array}{c|l}
6 & p = ((c+1)-1)! \land ((c+1)-1) \leq n \\
5 & p \times c = ((c+1)-1)! \land ((c-1)+1) \leq n \\
4 & p = (c-1)! \land c-1 \leq n \\
3 &
\end{array}
$$

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

$$
\begin{array}{r|l}
6 & p = ((c+1)-1)! \land ((c+1)-1) \leq n \\
5 & p \times c = ((c+1)-1)! \land ((c-1)+1) \leq n \\
4 & p = (c-1)! \land c-1 \leq n \\
3 & p = (1-1)! \land (1-1) \leq n \\
2 &
\end{array}
$$

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

$$
\begin{array}{r|l}
6 & p = ((c+1)-1)! \wedge ((c+1)-1) \leq n \\
5 & p \times c = ((c+1)-1)! \wedge ((c-1)+1) \leq n \\
4 & p = (c-1)! \wedge c-1 \leq n \\
3 & p = (1-1)! \wedge (1-1) \leq n \\
2 & 1 = (1-1)! \wedge (1-1) \leq n
\end{array}
$$

Notation für Verifikationsbedingungen

```
// {0 ≤ n}
p= 1;
c= 1;
while (c <= n) //** inv {p = (c − 1)! ∧ c − 1 ≤ n}; */
{ p = p * c;
  c = c + 1;
  }
// {p = n!}
```

WVC $6, 5 \,\big|$

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!}
```

WVC $\begin{array}{r|l} 6,5 & \emptyset \\ 4 & \end{array}$

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!}
```

WVC

$$
\begin{array}{r|l}
6,5 & \emptyset \\
4 & (p = (c-1)! \land c - 1 \le n \land c \le n \longrightarrow \\
 & \qquad\qquad p \times n = (c-1)! \land c - 1 \le n \land c \le n) \\
 & \land (p = (c-1)! \land c - 1 \le n \land \neg(c \le n) \longrightarrow \\
 & \qquad\qquad p = n! \\
3,2 &
\end{array}
$$

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!}
```

WVC

$$
\begin{array}{r|l}
6,5 & \emptyset \\
4 & (p = (c-1)! \wedge c - 1 \leq n \wedge c \leq n \longrightarrow \\
 & \qquad\qquad p \times n = (c-1)! \wedge c - 1 \leq n \wedge c \leq n) \\
 & \wedge (p = (c-1)! \wedge c - 1 \leq n \wedge \neg (c \leq n) \longrightarrow \\
 & \qquad\qquad p = n! \\
3,2 & \emptyset \\
1 &
\end{array}
$$

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!}
```

WVC

$$
\begin{array}{r|l}
6,5 & \emptyset \\
4 & (p = (c-1)! \land c-1 \leq n \land c \leq n \longrightarrow \\
 & \qquad\qquad p \times n = (c-1)! \land c-1 \leq n \land c \leq n) \\
 & \land(p = (c-1)! \land c-1 \leq n \land \neg(c \leq n) \longrightarrow \\
 & \qquad\qquad p = n! \\
3,2 & \emptyset \\
1 & 0 \leq n \longrightarrow 1 = (1-1)! \land (1-1) \leq n
\end{array}
$$

Vereinfachung von Verifikationsbedingungen

Wir nehmen folgende **strukturellen Vereinfachungen** an den generierten Verifikationsbedingungen vor:

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

Arbeitsblatt 8.1: Jetzt seid ihr dran!

```
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)}
```

- Wobei gilt: $\mathrm{sum}(i, j) = \begin{cases} 0 & \text{falls } i > j \\ i + \mathrm{sum}(i+1, j) & \text{sonst} \end{cases}$
- Berechnet die **AWP** für die Zeilen 5,4,3,2
- Berechnet die **WVC** für die Zeilen 5,4,3,2,1
- Sei c obiges Programm: Berechnet

$$WVC(\{0 \leq n \wedge n = N\}\, c\, \{p = \mathrm{sum}(1, N)\})$$

Jetzt seid ihr dran!

```
// {0 ≤ n ∧ n = N}
p= 0;
while (n>0) //** inv {p = sum(n+1, N); }*/
{ p = p + n;
  n = n - 1;
  }
// {p = sum(1, N)}
```

AWP 5 |

Jetzt seid ihr dran!

```
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)}
```

AWP	5	$p = \mathrm{sum}((n-1)+1, N)$
	4	

Jetzt seid ihr dran!

```
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)}
```

AWP		
	5	$p = \text{sum}((n-1)+1, N)$
	4	$p + n = \text{sum}((n-1)+1, N)$
	3	

Jetzt seid ihr dran!

```
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)}
```

AWP

$$
\begin{array}{r|l}
5 & p = \mathrm{sum}((n-1)+1, N) \\
4 & p + n = \mathrm{sum}((n-1)+1, N) \\
3 & p = \mathrm{sum}(n+1, N) \\
2 &
\end{array}
$$

Jetzt seid ihr dran!

```
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)}
```

$$\textbf{AWP}\quad \begin{array}{r|l} 5 & p = \mathrm{sum}((n-1)+1, N) \\ 4 & p + n = \mathrm{sum}((n-1)+1, N) \\ 3 & p = \mathrm{sum}(n+1, N) \\ 2 & 0 = \mathrm{sum}(n+1, N) \end{array}$$

Jetzt seid ihr dran!

```
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)}
```

$$
\begin{array}{ll|l}
\textbf{AWP} & 5 & p = \mathrm{sum}((n-1)+1, N) \\
 & 4 & p + n = \mathrm{sum}((n-1)+1, N) \\
 & 3 & p = \mathrm{sum}(n+1, N) \\
 & 2 & 0 = \mathrm{sum}(n+1, N) \\
\textbf{WVC} & 5 &
\end{array}
$$

Jetzt seid ihr dran!

```
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)}
```

AWP	5	$p = \mathrm{sum}((n-1)+1, N)$
	4	$p + n = \mathrm{sum}((n-1)+1, N)$
	3	$p = \mathrm{sum}(n+1, N)$
	2	$0 = \mathrm{sum}(n+1, N)$
WVC	5	$\emptyset$
	4	

Jetzt seid ihr dran!

```
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)}
```

AWP	5	$p = \mathrm{sum}((n-1)+1, N)$
	4	$p + n = \mathrm{sum}((n-1)+1, N)$
	3	$p = \mathrm{sum}(n+1, N)$
	2	$0 = \mathrm{sum}(n+1, N)$
WVC	5	$\emptyset$
	4	$\emptyset$
	3	

Jetzt seid ihr dran!

```
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)}
```

AWP	5	$p = \mathrm{sum}((n-1)+1, N)$
	4	$p + n = \mathrm{sum}((n-1)+1, N)$
	3	$p = \mathrm{sum}(n+1, N)$
	2	$0 = \mathrm{sum}(n+1, N)$
WVC	5	$\emptyset$
	4	$\emptyset$
	3	$\{(p = \mathrm{sum}(n+1, N) \wedge n > 0) \longrightarrow p + n = \mathrm{sum}((n-1)+1, N),$ $(p = \mathrm{sum}(n+1, N) \wedge \neg(n > 0)) \longrightarrow p = \mathrm{sum}(1, N)\}$
	2	

Jetzt seid ihr dran!

```
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)}
```

AWP	5	$p = \mathrm{sum}((n-1)+1, N)$
	4	$p + n = \mathrm{sum}((n-1)+1, N)$
	3	$p = \mathrm{sum}(n+1, N)$
	2	$0 = \mathrm{sum}(n+1, N)$
WVC	5	$\emptyset$
	4	$\emptyset$
	3	$\{(p = \mathrm{sum}(n+1, N) \wedge n > 0) \longrightarrow p + n = \mathrm{sum}((n-1)+1, N),$ $(p = \mathrm{sum}(n+1, N) \wedge \neg(n > 0)) \longrightarrow p = \mathrm{sum}(1, N)\}$
	2	$\emptyset \cup (3)$

Jetzt seid ihr dran!

```
// {0 ≤ n ∧ n = N}
p= 0;
while (n>0) //** inv {p = sum(n+1, N); }*/
{ p = p + n;
  n = n - 1;
  }
// {p = sum(1, N)}
```

$$WVC(\{0 \leq n \wedge n = N\}\, c\, \{p = \mathrm{sum}(1, N)\})$$

$$= \{(0 \leq n \wedge n = N) \longrightarrow 0 = \mathrm{sum}(n+1, N)\} \cup (3)$$

$$\begin{aligned} = \{&(0 \leq n \wedge n = N) \longrightarrow 0 = \mathrm{sum}(n+1, N), \\ &(p = \mathrm{sum}(n+1, N) \wedge n > 0) \longrightarrow p + n = \mathrm{sum}((n-1)+1, N), \\ &(p = \mathrm{sum}(n+1, N) \wedge \neg(n > 0)) \longrightarrow p = \mathrm{sum}(1, N)\} \end{aligned}$$

Jetzt seid ihr dran!

```
// {0 ≤ n ∧ n = N}
p= 0;
while (n>0) //** inv {0 ≤ n ∧ n ≤ N ∧ p = sum(n + 1, N); }*/ {
   p = p + n;
   n = n - 1;
   }
// {p = sum(1, N)}
```

AWP 5 |

Jetzt seid ihr dran!

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

AWP	5	$0 \leq (n-1) \wedge (n-1) \leq N \wedge p = \mathrm{sum}((n-1)+1, N)$
	4	

Jetzt seid ihr dran!

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

AWP		
	5	$0 \leq (n-1) \land (n-1) \leq N \land p = \text{sum}((n-1)+1, N)$
	4	$0 \leq (n-1) \land (n-1) \leq N \land p + n = \text{sum}((n-1)+1, N)$
	3	

Jetzt seid ihr dran!

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

$$
\begin{array}{ll|l}
\textbf{AWP} & 5 & 0 \leq (n-1) \land (n-1) \leq N \land p = \mathrm{sum}((n-1)+1, N) \\
 & 4 & 0 \leq (n-1) \land (n-1) \leq N \land p + n = \mathrm{sum}((n-1)+1, N) \\
 & 3 & 0 \leq n \land n \leq N \land p = \mathrm{sum}(n+1, N) \\
 & 2 &
\end{array}
$$

Jetzt seid ihr dran!

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

AWP	5	$0 \leq (n-1) \wedge (n-1) \leq N \wedge p = \mathrm{sum}((n-1)+1, N)$
	4	$0 \leq (n-1) \wedge (n-1) \leq N \wedge p + n = \mathrm{sum}((n-1)+1, N)$
	3	$0 \leq n \wedge n \leq N \wedge p = \mathrm{sum}(n+1, N)$
	2	$0 \leq n \wedge n \leq N \wedge 0 = \mathrm{sum}(n+1, N)$

Jetzt seid ihr dran!

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

$$
\begin{array}{rr|l}
\textbf{AWP} & 5 & 0 \leq (n-1) \wedge (n-1) \leq N \wedge p = \mathrm{sum}((n-1)+1, N) \\
 & 4 & 0 \leq (n-1) \wedge (n-1) \leq N \wedge p + n = \mathrm{sum}((n-1)+1, N) \\
 & 3 & 0 \leq n \wedge n \leq N \wedge p = \mathrm{sum}(n+1, N) \\
 & 2 & 0 \leq n \wedge n \leq N \wedge 0 = \mathrm{sum}(n+1, N) \\
\textbf{WVC} & 5,4 &
\end{array}
$$

Jetzt seid ihr dran!

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

AWP	5	$0 \leq (n-1) \land (n-1) \leq N \land p = \text{sum}((n-1)+1, N)$
	4	$0 \leq (n-1) \land (n-1) \leq N \land p + n = \text{sum}((n-1)+1, N)$
	3	$0 \leq n \land n \leq N \land p = \text{sum}(n+1, N)$
	2	$0 \leq n \land n \leq N \land 0 = \text{sum}(n+1, N)$
WVC	5, 4	$\emptyset$
	3	

Jetzt seid ihr dran!

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

AWP	5	$0 \leq (n-1) \wedge (n-1) \leq N \wedge p = \mathrm{sum}((n-1)+1, N)$
	4	$0 \leq (n-1) \wedge (n-1) \leq N \wedge p + n = \mathrm{sum}((n-1)+1, N)$
	3	$0 \leq n \wedge n \leq N \wedge p = \mathrm{sum}(n+1, N)$
	2	$0 \leq n \wedge n \leq N \wedge 0 = \mathrm{sum}(n+1, N)$
WVC	5, 4	$\emptyset$
	3	$\{(0 \leq n \wedge n \leq N \wedge p = \mathrm{sum}(n+1, N) \wedge n > 0) \longrightarrow (0 \leq (n-1) \wedge (n-1) \leq N \wedge p + n = \mathrm{sum}((n-1)+1, N)),$ $(n \geq 0 \wedge n \leq N \wedge p = \mathrm{sum}(n+1, N) \wedge \neg(n > 0)) \longrightarrow p = \mathrm{sum}(1, N)\}$
	2	

Jetzt seid ihr dran!

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

$$
\begin{array}{ll|l}
\textbf{AWP} & 5 & 0 \le (n-1) \wedge (n-1) \le N \wedge p = \mathrm{sum}((n-1)+1, N) \\
 & 4 & 0 \le (n-1) \wedge (n-1) \le N \wedge p + n = \mathrm{sum}((n-1)+1, N) \\
 & 3 & 0 \le n \wedge n \le N \wedge p = \mathrm{sum}(n+1, N) \\
 & 2 & 0 \le n \wedge n \le N \wedge 0 = \mathrm{sum}(n+1, N) \\
\textbf{WVC} & 5,4 & \emptyset \\
 & 3 & \{(0 \le n \wedge n \le N \wedge p = \mathrm{sum}(n+1, N) \wedge n > 0) \\
 & & \quad \longrightarrow (0 \le (n-1) \wedge (n-1) \le N \wedge p + n = \mathrm{sum}((n-1)+1, N)), \\
 & & \ (n \ge 0 \wedge n \le N \wedge p = \mathrm{sum}(n+1, N) \wedge \neg(n > 0)) \longrightarrow p = \mathrm{sum}(1, N)\} \\
 & 2 & \emptyset \cup (3)
\end{array}
$$

Jetzt seid ihr dran!

```
// {0 ≤ n ∧ n = N}
p= 0;
while (n>0) //** inv {0 ≤ n ∧ n ≤ N ∧ p = sum(n + 1, N); }*/ {
   p = p + n;
   n = n - 1;
   }
// {p = sum(1, N)}
```

$$
\begin{aligned}
&WVC(\{0 \leq n \wedge n = N\}\, c\, \{p = \mathrm{sum}(1, N)\})\\
&\qquad = \{\{(0 \leq n \wedge n = N) \longrightarrow (0 \leq n \wedge n \leq N \wedge 0 = \mathrm{sum}(n + 1, N))\} \cup (3)\\
&\qquad = \{(0 \leq n \wedge n = N) \longrightarrow (0 \leq n \wedge n \leq N \wedge 0 = \mathrm{sum}(n + 1, N)),\\
&\qquad\qquad (0 \leq n \wedge n \leq N \wedge p = \mathrm{sum}(n + 1, N) \wedge n > 0)\\
&\qquad\qquad\qquad \longrightarrow (0 \leq (n - 1) \wedge (n - 1) \leq N \wedge p + n = \mathrm{sum}((n - 1) + 1, N)),\\
&\qquad\qquad (n \geq 0 \wedge n \leq N \wedge p = \mathrm{sum}(n + 1, N) \wedge \neg(n > 0)) \longrightarrow p = \mathrm{sum}(1, N)\}
\end{aligned}
$$

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

AWP 8

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 < i} */
5  { if (a[r] < a[i]) {
6        r= i; }
7    else { }
8    i= i+1; }
9  // {(∀j. 0 ≤ j < n ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

AWP

8	$\varphi(i+1, r)$
7	

Weiteres Beispiel: Maximales Element

```
1  // {0 < n}
2  i= 0;
3  r= 0;
```

$$4 \quad \texttt{while (i != n) //** inv } \overbrace{\{(\forall j.\, 0 \leq j < i \longrightarrow a[j] \leq a[r]) \wedge 0 \leq r < i\}}^{\varphi(i,r)} \texttt{ */}$$

```
5  { if (a[r] < a[i]) {
6        r= i; }
7    else { }
8    i= i+1; }
```

$$9 \quad \texttt{// } \underbrace{\{(\forall j.\, 0 \leq j < \boxed{n} \longrightarrow a[j] \leq a[\boxed{r}]) \wedge 0 \leq r < \boxed{n}\}}_{\varphi(\boxed{n},\boxed{r})}$$

AWP		
	8	$\varphi(i+1, r)$
	7	$\varphi(i+1, r)$
	6	

Weiteres Beispiel: Maximales Element

1 `//` $\{0 < n\}$

2 `i= 0;`

3 `r= 0;`

4 `while (i != n)` `//** inv` $\{\overbrace{(\forall j.\, 0 \le j < i \longrightarrow a[j] \le a[r]) \land 0 \le r < i}^{\varphi(i,r)}\}$ `*/`

5 `{ if (a[r] < a[i]) {`

6 `      r= i; }`

7 `   else { }`

8 `   i= i+1; }`

9 `//` $\{\underbrace{(\forall j.\, 0 \le j < n \longrightarrow a[j] \le a[r]) \land 0 \le r < n}_{\varphi(n,r)}\}$

AWP

8	$\varphi(i+1, r)$
7	$\varphi(i+1, r)$
6	$\varphi(i+1, i)$
5	

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

AWP		
	8	$\varphi(i+1, r)$
	7	$\varphi(i+1, r)$
	6	$\varphi(i+1, i)$
	5	$(a[r] < a[i] \land \varphi(i+1, i)) \lor (\neg(a[r] < a[i]) \land \varphi(i+1, r))$
	4	

Weiteres Beispiel: Maximales Element

1 // $\{0 < n\}$

2 i= 0;

3 r= 0;

4 **while** (i != n) //** inv $\{\overbrace{(\forall j.\, 0 \le j < i \longrightarrow a[j] \le a[r]) \land 0 \le r < i}^{\varphi(i,r)}\}$ */

5 { **if** (a[r] < a[i]) {

6 r= i; }

7 **else** { }

8 i= i+1; }

9 // $\{\underbrace{(\forall j.\, 0 \le j < n \longrightarrow a[j] \le a[r]) \land 0 \le r < n}_{\varphi(n,\, r)}\}$

AWP		
	8	$\varphi(i+1, r)$
	7	$\varphi(i+1, r)$
	6	$\varphi(i+1, i)$
	5	$(a[r] < a[i] \land \varphi(i+1, i)) \lor (\neg(a[r] < a[i]) \land \varphi(i+1, r))$
	4	$\varphi(i, r)$
	3	

Weiteres Beispiel: Maximales Element

```
1  // {0 < n}
2  i= 0;
3  r= 0;
```

$\varphi(i,r)$

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

$\varphi(n, r)$

AWP

$$
\begin{array}{r|l}
8 & \varphi(i+1, r) \\
7 & \varphi(i+1, r) \\
6 & \varphi(i+1, i) \\
5 & (a[r] < a[i] \land \varphi(i+1, i)) \lor (\neg(a[r] < a[i]) \land \varphi(i+1, r)) \\
4 & \varphi(i, r) \\
3 & \varphi(i, 0) \\
2 &
\end{array}
$$

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

$$\textbf{AWP}\quad \begin{array}{c|l}
8 & \varphi(i+1, r) \\
7 & \varphi(i+1, r) \\
6 & \varphi(i+1, i) \\
5 & (a[r] < a[i] \wedge \varphi(i+1, i)) \vee (\neg(a[r] < a[i]) \wedge \varphi(i+1, r)) \\
4 & \varphi(i, r) \\
3 & \varphi(i, 0) \\
2 & \varphi(0, 0)
\end{array}$$

Weiteres Beispiel: Maximales Element

1 `//` $\{0 < n\}$
2 `i= 0;`
3 `r= 0;`

4 `while (i != n) //** inv` $\{\overbrace{(\forall j.\, 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < i}^{\varphi(i,r)}\}$ `*/`
5 `{ if (a[r] < a[i]) {`
6 `      r= i; }`
7 `   else { }`
8 `   i= i+1; }`
9 `//` $\{\underbrace{(\forall j.\, 0 \le j < n \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n}_{\varphi(n,r)}\}$

WVC

$8, 7, 6, 5 \mid$

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

WVC

8, 7, 6, 5	$\emptyset$
4	

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

$$\text{// } \{0 < n\} \qquad \overbrace{(\forall j.\, 0 \le j < i \longrightarrow a[j] \le a[r]) \land 0 \le r < i}^{\varphi(i,r)} \qquad \underbrace{(\forall j.\, 0 \le j < n \longrightarrow a[j] \le a[r]) \land 0 \le r < n}_{\varphi(n,r)}$$

WVC

$$\begin{array}{r|l}
8,7,6,5 & \emptyset \\
4 & (\varphi(i,r) \land i \neq n) \longrightarrow \\
 & \qquad ((a[r] < a[i] \land \varphi(i+1, i)) \lor (\neg(a[r] < a[i]) \land \varphi(i+1, r)))
\end{array}$$

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 < i} */
5  { if (a[r] < a[i]) {
6        r= i; }
7     else { }
8     i= i+1; }
9  // {(∀j. 0 ≤ j < n ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

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

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

WVC

$$
\begin{array}{r|l}
8,7,6,5 & \emptyset \\
4 & (\varphi(i,r) \wedge i \neq n) \longrightarrow \\
 & \qquad ((a[r] < a[i] \wedge \varphi(i+1,i)) \vee (\neg(a[r] < a[i]) \wedge \varphi(i+1,r))) \\
 & (\varphi(i,r) \wedge \neg(i \neq n)) \longrightarrow \varphi(n,r) \\
3,2 &
\end{array}
$$

Weiteres Beispiel: Maximales Element

```
1  // {0 < n}
2  i= 0;
3  r= 0;
```

$\varphi(i,r)$

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

$\varphi(n,r)$

WVC

$$
\begin{array}{r|l}
8,7,6,5 & \emptyset \\
4 & (\varphi(i,r) \land i \neq n) \longrightarrow \\
 & \qquad ((a[r] < a[i] \land \varphi(i+1,i)) \lor (\neg(a[r] < a[i]) \land \varphi(i+1,r))) \\
 & (\varphi(i,r) \land \neg(i \neq n)) \longrightarrow \varphi(n,r) \\
3,2 & \emptyset
\end{array}
$$

Weiteres Beispiel: Maximales Element

`//` $\{0 < n\}$
`i= 0;`
`r= 0;`

`while (i!= n) //** inv` $\{\overbrace{(\forall j.\, 0 \leq j < i \longrightarrow a[j] \leq a[r]) \land 0 \leq r < i}^{\varphi(i,r)}\}$ `*/`
`{ if (a[r] < a[i]) {`
`      r= i; }`
`   else { }`
`   i= i+1; }`
`//` $\{\underbrace{(\forall j.\, 0 \leq j < n \longrightarrow a[j] \leq a[r]) \land 0 \leq r < n}_{\varphi(n,r)}\}$

- 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:

$$\mathrm{awp}(\textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1, P) \stackrel{def}{=} (b \land \mathrm{awp}(c_0, P)) \lor (\neg b \land \mathrm{awp}(c_1, P))$$

Wenn $\mathrm{awp}(c_0, P) = b \land P_0$, $\mathrm{awp}(c_1, P) = \neg b \land P_0$, dann gilt

$$(b \land b \land P_0) \lor (\neg b \land \neg b \land P_0) = (b \land P_0) \lor (\neg b \land P_0) = (b \lor \neg b) \land P_0 = P_0$$

Überblick: Approximative schwächste Vorbedingung

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

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  { if (a[r] < a[i]) {
6        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ a[r] < a[i]} */
7        r= i; }
8     else {
9        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])} */
10       }
11    i= i+1; }
12 // {(∀j. 0 ≤ j < n ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

AWP 11 |

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  { if (a[r] < a[i]) {
6        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ a[r] < a[i]} */
7        r= i; }
8     else {
9        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])} */
10       }
11    i= i+1; }
12 // {(∀j. 0 ≤ j < n ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

AWP		
	11	$\varphi(i+1, r)$
	9	

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  { if (a[r] < a[i]) {
6        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ a[r] < a[i]} */
7        r= i; }
8      else {
9        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])} */
10       }
11     i= i+1; }
12 // {(∀j. 0 ≤ j < n ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

AWP		
	11	$\varphi(i+1, r)$
	9	$\varphi(i, r) \wedge \neg(a[r] < a[i])$
	7	

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  { if (a[r] < a[i]) {
6        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ a[r] < a[i]} */
7        r= i; }
8     else {
9        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])} */
10       }
11    i= i+1; }
12 // {(∀j. 0 ≤ j < n ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

AWP	11	$\varphi(i+1, r)$
	9	$\varphi(i, r) \wedge \neg(a[r] < a[i])$
	7	$\varphi(i+1, i)$
	6	

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  { if (a[r] < a[i]) {
6        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ a[r] < a[i]} */
7        r= i; }
8      else {
9        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])} */
10       }
11     i= i+1; }
12 // {(∀j. 0 ≤ j < n ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

AWP				
	11	$\varphi(i+1, r)$	5	
	9	$\varphi(i, r) \wedge \neg(a[r] < a[i])$		
	7	$\varphi(i+1, i)$		
	6	$\varphi(i, r) \wedge a[r] < a[i]$		

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  { if (a[r] < a[i]) {
6        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ a[r] < a[i]} */
7        r= i; }
8     else {
9        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])} */
10       }
11    i= i+1; }
12 // {(∀j. 0 ≤ j < n ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

AWP

11	$\varphi(i+1, r)$	5	$\varphi(i, r)$
9	$\varphi(i, r) \land \neg(a[r] < a[i])$	4	
7	$\varphi(i+1, i)$		
6	$\varphi(i, r) \land a[r] < a[i]$		

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  { if (a[r] < a[i]) {
6        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ a[r] < a[i]} */
7        r= i; }
8     else {
9        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])} */
10       }
11    i= i+1; }
12 // {(∀j. 0 ≤ j < n ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

AWP

11	$\varphi(i+1, r)$	5	$\varphi(i, r)$
9	$\varphi(i, r) \wedge \neg(a[r] < a[i])$	4	$\varphi(i, r)$
7	$\varphi(i+1, i)$	3	
6	$\varphi(i, r) \wedge a[r] < a[i]$		

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   { if (a[r] < a[i]) {
6         //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ a[r] < a[i]} */
7         r= i; }
8     else {
9         //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])} */
10        }
11    i= i+1; }
12  // {(∀j. 0 ≤ j < n ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

AWP	11	$\varphi(i+1, r)$	5	$\varphi(i, r)$
	9	$\varphi(i, r) \wedge \neg(a[r] < a[i])$	4	$\varphi(i, r)$
	7	$\varphi(i+1, i)$	3	$\varphi(i, 0)$
	6	$\varphi(i, r) \wedge a[r] < a[i]$	2	

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  { if (a[r] < a[i]) {
6        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ a[r] < a[i]} */
7        r= i; }
8     else {
9        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])} */
10       }
11    i= i+1; }
12 // {(∀j. 0 ≤ j < n ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

AWP	11	$\varphi(i+1, r)$	5	$\varphi(i, r)$
	9	$\varphi(i, r) \wedge \neg(a[r] < a[i])$	4	$\varphi(i, r)$
	7	$\varphi(i+1, i)$	3	$\varphi(i, 0)$
	6	$\varphi(i, r) \wedge a[r] < a[i]$	2	$\varphi(0, 0)$

Maximales Element mit Zusicherung

```
// {0 < n}
i= 0;
r= 0;
while (i != n) //** inv {(∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n} */
{ if (a[r] < a[i]) {
      //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ a[r] < a[i]} */
      r= i; }
   else {
      //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])} */
      }
   i= i+1; }
// {(∀j. 0 ≤ j < n ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

WVC 11

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  { if (a[r] < a[i]) {
6        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ a[r] < a[i]} */
7        r= i; }
8     else {
9        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])} */
10       }
11    i= i+1; }
12 // {(∀j. 0 ≤ j < n ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

$$\textbf{WVC} \begin{array}{r|l} 11 & \emptyset \\ 9 & \end{array}$$

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  { if (a[r] < a[i]) {
6        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ a[r] < a[i]} */
7        r= i; }
8      else {
9        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])} */
10       }
11     i= i+1; }
12 // {(∀j. 0 ≤ j < n ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

WVC	
11	$\emptyset$
9	$(\varphi(i, r) \wedge \neg(a[r] < a[i])) \longrightarrow \varphi(i+1, r)$
7	

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  { if (a[r] < a[i]) {
6        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ a[r] < a[i]} */
7        r= i; }
8    else {
9        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])} */
10       }
11   i= i+1; }
12 // {(∀j. 0 ≤ j < n ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

WVC		
	11	$\emptyset$
	9	$(\varphi(i, r) \land \neg(a[r] < a[i])) \longrightarrow \varphi(i+1, r)$
	7	$\emptyset$
	6	

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  { if (a[r] < a[i]) {
6        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ a[r] < a[i]} */
7        r= i; }
8      else {
9        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])} */
10       }
11     i= i+1; }
12 // {(∀j. 0 ≤ j < n ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

WVC	11	$\emptyset$	5	
	9	$(\varphi(i,r) \land \neg(a[r] < a[i])) \longrightarrow \varphi(i+1,r)$		
	7	$\emptyset$		
	6	$(\varphi(i,r) \land a[r] < a[i]) \longrightarrow \varphi(i+1,i)$		

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  { if (a[r] < a[i]) {
6        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ a[r] < a[i]} */
7        r= i; }
8    else {
9        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])} */
10       }
11   i= i+1; }
12 // {(∀j. 0 ≤ j < n ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

WVC

11	$\emptyset$
9	$(\varphi(i,r) \wedge \neg(a[r] < a[i])) \longrightarrow \varphi(i+1,r)$
7	$\emptyset$
6	$(\varphi(i,r) \wedge a[r] < a[i]) \longrightarrow \varphi(i+1,i)$
5	$(\varphi(i,r) \wedge \neg(a[r] < a[i])) \longrightarrow \varphi(i+1,r)$ $(\varphi(i,r) \wedge a[r] < a[i]) \longrightarrow \varphi(i+1,i)$

Maximales Element mit Zusicherung

```
// {0 < n}
i= 0;
r= 0;
while (i!= n) //** inv {(∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n} */
{ if (a[r] < a[i]) {
      //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ a[r] < a[i]} */
      r= i; }
   else {
      //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])} */
      }
   i= i+1; }
// {(∀j. 0 ≤ j < n ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

WVC $4 \mid$

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  { if (a[r] < a[i]) {
6        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ a[r] < a[i]} */
7        r= i; }
8     else {
9        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])} */
10       }
11    i= i+1; }
12 // {(∀j. 0 ≤ j < n ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

WVC		
	4	(5)
		$(\varphi(i, r) \wedge i \neq n) \longrightarrow \varphi(i+1, r)$
		$(\varphi(i, r) \wedge \neg(i \neq n)) \longrightarrow \varphi(n, r)$
	3, 2	

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  { if (a[r] < a[i]) {
6        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ a[r] < a[i]} */
7        r= i; }
8     else {
9        //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])} */
10       }
11    i= i+1; }
12 // {(∀j. 0 ≤ j < n ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

WVC

$$
\begin{array}{r|l}
4 & (5) \\
  & (\varphi(i,r) \wedge i \neq n) \longrightarrow \varphi(i+1, r) \\
  & (\varphi(i,r) \wedge \neg(i \neq n)) \longrightarrow \varphi(n, r) \\
3, 2 & \emptyset
\end{array}
$$

Maximales Element mit Zusicherung

```
// {0 < n}
i= 0;
r= 0;
while (i!= n) //** inv {(∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r]) ∧ 0 ≤ r < n} */

{ if (a[r] < a[i]) {
    //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ a[r] < a[i]} */
    r= i; }
  else {
    //{∀j. 0 ≤ j < i ⟶ a[j] ≤ a[r] ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])} */
    }
  i= i+1; }
// {(∀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.

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.
- Warum eigentlich immer **rückwärts**?

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.
- Warum eigentlich immer **rückwärts**?
  Jetzt gleich. . .