

Korrekte Software: Grundlagen und Methoden
Vorlesung 8 vom 28.05.19
Verifikationsbedingungen

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2019

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
- ▶ Funktionsaufrufe und das Framing-Problem
- ▶ 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.

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

Rückwärtsanwendung der Regeln

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

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

- Was ist mit den anderen Regeln?

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

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

$$\frac{A' \implies A \quad \vdash \{A\} c \{B\} \quad B \implies B'}{\vdash \{A'\} c \{B'\}}$$

Rückwärtsanwendung der Regeln

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

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

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

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

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

$$\frac{A' \implies A \quad \vdash \{A\} c \{B\} \quad B \implies B'}{\vdash \{A'\} c \{B'\}}$$

Berechnung von Vorbedingungen

- ▶ Die Rückwärtsrechnung von einer gegebenen Nachbedingung entspricht der Berechnung einer Vorbedingung.
- ▶ Gegeben C0-Programm c , Prädikat Q , dann ist
 - ▶ $\text{wp}(c, Q)$ die **schwächste Vorbedingung** P so dass $\models \{P\} c \{Q\}$;
 - ▶ Prädikat P **schwächer** als P' wenn $P' \implies P$
- ▶ Semantische Charakterisierung:

Schwächste Vorbedingung

Gegeben Zusicherung $Q \in \mathbf{Assn}$ und Programm $c \in \mathbf{Stmt}$, dann

$$\models \{P\} c \{Q\} \iff P \implies \text{wp}(c, Q)$$

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

Berechnung von $\text{wp}(c, Q)$

- Einfach für Programme ohne Schleifen:

$$\begin{aligned}\text{wp}(\{ \}, P) &\stackrel{\text{def}}{=} P \\ \text{wp}(x = e, P) &\stackrel{\text{def}}{=} P[e/x] \\ \text{wp}(c_1; c_2, P) &\stackrel{\text{def}}{=} \text{wp}(c_1, \text{wp}(c_2, P)) \\ \text{wp}(\text{if } (b) \text{ } c_0 \text{ else } c_1, P) &\stackrel{\text{def}}{=} (b \wedge \text{wp}(c_0, P)) \vee (\neg b \wedge \text{wp}(c_1, P))\end{aligned}$$

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

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

- 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** $\text{wvc}(c, Q)$
- ▶ Die Verifikationsbedingungen treten dort auf, wo die Weakening-Regel angewandt wird.
- ▶ Es gilt:

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

Approximative schwächste Vorbedingung

- Für die **while**-Schleife:

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

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

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

$$\frac{\vdash \{A \wedge b\} c \{A\}}{\vdash \{A\} \text{ while } (b) c \{A \wedge \neg b\}} \quad (1)$$

$$\frac{A \wedge b \implies C \quad \vdash \{C\} c \{A\} \quad A \wedge \neg b \implies B}{\vdash \{A\} \text{ while } (b) c \{B\}} \quad (2)$$

Überblick: Approximative schwächste Vorbedingung

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

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

$$\text{awp}(c_1; c_2, P) \stackrel{\text{def}}{=} \text{awp}(c_1, \text{awp}(c_2, P))$$

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

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

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

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

$$\text{wvc}(c_1; c_2, P) \stackrel{\text{def}}{=} \text{wvc}(c_1, \text{awp}(c_2, P)) \cup \text{wvc}(c_2, P)$$

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

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

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

Beispiel: das Fakultätsprogramm

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

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

- ▶ Berechnung der Verifikationsbedingungen zur Nachbedingung.

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 |

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

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

6	$p = ((c + 1) - 1)! \wedge ((c - 1) + 1) \leq n$
5	$p \cdot c = ((c + 1) - 1)! \wedge ((c - 1) + 1) \leq n$
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

6	$p = ((c + 1) - 1)! \wedge ((c - 1) + 1) \leq n$
5	$p \cdot c = ((c + 1) - 1)! \wedge ((c - 1) + 1) \leq n$
4	$p = (c - 1)! \wedge c - 1 \leq n$
3	

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	$p \cdot c = ((c + 1) - 1)! \wedge ((c - 1) + 1) \leq n$
	4	$p = (c - 1)! \wedge c - 1 \leq n$
	3	$p = (1 - 1)! \wedge (1 - 1) \leq n$
	2	

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	$p \cdot c = ((c + 1) - 1)! \wedge ((c - 1) + 1) \leq n$
4	$p = (c - 1)! \wedge c - 1 \leq n$
3	$p = (1 - 1)! \wedge (1 - 1) \leq n$
2	$1 = (1 - 1)! \wedge (1 - 1) \leq n$

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!}
```

VC 6, 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!}
```

VC 6, 5 | $\emptyset$
 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!}
```

VC	6, 5		$\emptyset$
	4		$(p = (c - 1)! \wedge c - 1 \leq n \wedge c \leq n \longrightarrow$ $p = (c - 1)! \wedge c - 1 \leq n \wedge c \leq n)$ $\wedge (p = (c - 1)! \wedge c - 1 \leq n \wedge \neg(c \leq n) \longrightarrow$ $p = n!)$
	3, 2		

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!}
```

VC	6, 5		$\emptyset$
	4		$(p = (c - 1)! \wedge c - 1 \leq n \wedge c \leq n \longrightarrow$ $p = (c - 1)! \wedge c - 1 \leq n \wedge c \leq n)$ $\wedge (p = (c - 1)! \wedge c - 1 \leq n \wedge \neg(c \leq n) \longrightarrow$ $p = n!)$
	3, 2		$\emptyset$
	1		

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!}
```

VC	6, 5		$\emptyset$
	4		$(p = (c - 1)! \wedge c - 1 \leq n \wedge c \leq n \longrightarrow$ $p = (c - 1)! \wedge c - 1 \leq n \wedge c \leq n)$ $\wedge (p = (c - 1)! \wedge c - 1 \leq n \wedge \neg(c \leq n) \longrightarrow$ $p = n!)$
	3, 2		$\emptyset$
	1		$0 \leq n \longrightarrow 1 = (1 - 1)! \wedge (1 - 1) \leq n$

Vereinfachung von Verifikationsbedingungen

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

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

Weiteres Beispiel: Maximales Element

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

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

AWP 8 |

Weiteres Beispiel: Maximales Element

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

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

AWP $\begin{array}{c|c} 8 & \varphi(i+1, r) \\ 7 & \end{array}$

Weiteres Beispiel: Maximales Element

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

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

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

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;

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

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

Weiteres Beispiel: Maximales Element

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

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

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

Weiteres Beispiel: Maximales Element

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

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

AWP	8	$\varphi(i+1, r)$
	7	$\varphi(i+1, r)$
	6	$\varphi(i+1, i)$
	5	$(a[r] < a[i] \wedge \varphi(i+1, i) \vee (\neg(a[r] < a[i]) \wedge \varphi(i+1, r)))$
	4	$\varphi(i, r)$
	3	$\varphi(i, 0)$
	2	

Weiteres Beispiel: Maximales Element

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

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

AWP	8	$\varphi(i+1, r)$
	7	$\varphi(i+1, r)$
	6	$\varphi(i+1, i)$
	5	$(a[r] < a[i] \wedge \varphi(i+1, i) \vee (\neg(a[r] < a[i]) \wedge \varphi(i+1, r)))$
	4	$\varphi(i, r)$
	3	$\varphi(i, 0)$
	2	$\varphi(0, 0)$

Weiteres Beispiel: Maximales Element

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

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

VC

8, 7, 6, 5 |

Weiteres Beispiel: Maximales Element

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

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

VC

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

Weiteres Beispiel: Maximales Element

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

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

VC

8, 7, 6, 5		$\emptyset$
4		$(\varphi(i, r) \wedge i \neq n) \rightarrow$ $((a[r] < a[i] \wedge \varphi(i + 1, i)) \vee (\neg(a[r] < a[i]) \wedge \varphi(i + 1, r)))$

Weiteres Beispiel: Maximales Element

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

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

VC

8, 7, 6, 5		$\emptyset$
4		$(\varphi(i, r) \wedge i \neq n) \rightarrow$ $((a[r] < a[i] \wedge \varphi(i + 1, i)) \vee (\neg(a[r] < a[i]) \wedge \varphi(i + 1, r)))$ $(\varphi(i, r) \wedge \neg(i \neq n)) \rightarrow \varphi(n, r)$
3, 2		

Weiteres Beispiel: Maximales Element

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

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

VC

8, 7, 6, 5		$\emptyset$
4		$(\varphi(i, r) \wedge i \neq n) \rightarrow$ $((a[r] < a[i] \wedge \varphi(i + 1, i)) \vee (\neg(a[r] < a[i]) \wedge \varphi(i + 1, r)))$ $(\varphi(i, r) \wedge \neg(i \neq n)) \rightarrow \varphi(n, r)$
3, 2		$\emptyset$

Weiteres Beispiel: Maximales Element

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

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

- ▶ 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**

Assn $a ::= \dots$ — Zusicherungen

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

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

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

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

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

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

$$\text{awp}(c_1; c_2, P) \stackrel{\text{def}}{=} \text{awp}(c_1, \text{awp}(c_2, P))$$

$$\text{awp}(\text{if } (b) \ c_0 \ \text{else} \ c_1, P) \stackrel{\text{def}}{=} \begin{array}{l} Q \quad \text{wenn} \quad \text{awp}(c_0, P) = b \wedge Q, \\ \quad \text{awp}(c_1, P) = \neg b \wedge Q \end{array}$$

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

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

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

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

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

$$\text{wvc}(c_1; c_2, P) \stackrel{\text{def}}{=} \text{wvc}(c_1, \text{awp}(c_2, P)) \cup \text{wvc}(c_2, P)$$

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

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

$$\text{wvc}(\text{while } (b) \ \text{/** } \text{inv } i \text{ */ } c, P) \stackrel{\text{def}}{=} \begin{array}{l} \text{wvc}(c, i) \cup \{i \wedge b \longrightarrow \text{awp}(c, i)\} \\ \cup \{i \wedge \neg b \longrightarrow P\} \end{array}$$

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 {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n} */ {
5      if (a[r] < a[i]) {
6          // {\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge a[r] < a[i]} */
7          r = i; }
8      else {
9          // {\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge \neg(a[r] < a[i])} */
10         }
11     i = i + 1; }
12 // {(\forall j. 0 \le j < n \longrightarrow a[j] \le a[r]) \wedge 0 \le 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 {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n} */ {
5      if (a[r] < a[i]) {
6          // {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge a[r] < a[i])} */
7          r = i; }
8      else {
9          // {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge \neg(a[r] < a[i]))} */
10         }
11     i = i + 1; }
12 // {(\forall j. 0 \le j < n \longrightarrow a[j] \le a[r]) \wedge 0 \le 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 {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n} */ {
5      if (a[r] < a[i]) {
6          // {\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge a[r] < a[i]} */
7          r = i; }
8      else {
9          // {\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge \neg(a[r] < a[i])} */
10         }
11     i = i + 1; }
12 // {(\forall j. 0 \le j < n \longrightarrow a[j] \le a[r]) \wedge 0 \le 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 {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n} */ {
5      if (a[r] < a[i]) {
6          // {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge a[r] < a[i])} */
7          r = i; }
8      else {
9          // {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge \neg(a[r] < a[i]))} */
10         }
11     i = i + 1; }
12 // {(\forall j. 0 \le j < n \longrightarrow a[j] \le a[r]) \wedge 0 \le 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 {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n} */ {
5     if (a[r] < a[i]) {
6         /** {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge a[r] < a[i])} */
7         r = i; }
8     else {
9         /** {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge \neg(a[r] < a[i]))} */
10        }
11    i = i + 1; }
12 /** {(\forall j. 0 \le j < n \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n}
```

AWP	11	$\varphi(i+1, r)$	5	$\varphi(i, r)$
	9	$\varphi(i, r) \wedge \neg(a[r] < a[i])$	4	
	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 {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n} */ {
5      if (a[r] < a[i]) {
6          // {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge a[r] < a[i])} */
7          r = i; }
8      else {
9          // {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge \neg(a[r] < a[i]))} */
10         }
11     i = i + 1; }
12 // {(\forall j. 0 \le j < n \longrightarrow a[j] \le a[r]) \wedge 0 \le 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 {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n} */ {
5      if (a[r] < a[i]) {
6          // {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge a[r] < a[i])} */
7          r = i; }
8      else {
9          // {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge \neg(a[r] < a[i]))} */
10         }
11     i = i + 1; }
12 // {(\forall j. 0 \le j < n \longrightarrow a[j] \le a[r]) \wedge 0 \le 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 {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n} */ {
5      if (a[r] < a[i]) {
6          // {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge a[r] < a[i])} */
7          r = i; }
8      else {
9          // {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge \neg(a[r] < a[i]))} */
10         }
11     i = i + 1; }
12 // {(\forall j. 0 \le j < n \longrightarrow a[j] \le a[r]) \wedge 0 \le 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

```
1  // {0 < n}
2  i = 0;
3  r = 0;
4  while (i != n) /** inv {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n} */ {
5      if (a[r] < a[i]) {
6          // {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge a[r] < a[i])} */
7          r = i; }
8      else {
9          // {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge \neg(a[r] < a[i]))} */
10         }
11     i = i + 1; }
12 // {(\forall j. 0 \le j < n \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n}
```

VC 11 |

Maximales Element mit Zusicherung

```
1  // {0 < n}
2  i = 0;
3  r = 0;
4  while (i != n) /** inv {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n} */ {
5      if (a[r] < a[i]) {
6          // {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge a[r] < a[i])} */
7          r = i; }
8      else {
9          // {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge \neg(a[r] < a[i]))} */
10         }
11     i = i + 1; }
12 // {(\forall j. 0 \le j < n \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n}

VC 11 | \emptyset
    9 |
```

Maximales Element mit Zusicherung

```
1  // {0 < n}
2  i = 0;
3  r = 0;
4  while (i != n) /** inv {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n} */ {
5      if (a[r] < a[i]) {
6          // {\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge a[r] < a[i]} */
7          r = i; }
8      else {
9          // {\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge \neg(a[r] < a[i])} */
10         }
11     i = i + 1; }
12 // {(\forall j. 0 \le j < n \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n}
```

VC 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 {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n} */ {
5      if (a[r] < a[i]) {
6          // {\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge a[r] < a[i]} */
7          r = i; }
8      else {
9          // {\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge \neg(a[r] < a[i])} */
10         }
11     i = i + 1; }
12 // {(\forall j. 0 \le j < n \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n}
```

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

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

Maximales Element mit Zusicherung

```
1  // {0 < n}
2  i = 0;
3  r = 0;
4  while (i != n) /** inv {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n} */ {
5      if (a[r] < a[i]) {
6          // {\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge a[r] < a[i]} */
7          r = i; }
8      else {
9          // {\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge \neg(a[r] < a[i])} */
10         }
11     i = i + 1; }
12 // {(\forall j. 0 \le j < n \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n}
```

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

Maximales Element mit Zusicherung

```
1  // {0 < n}
2  i = 0;
3  r = 0;
4  while (i != n) /** inv {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n} */ {
5      if (a[r] < a[i]) {
6          // {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge a[r] < a[i])} */
7          r = i; }
8      else {
9          // {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge \neg(a[r] < a[i]))} */
10         }
11     i = i + 1; }
12 // {(\forall j. 0 \le j < n \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n}
```

VC 4 |

Maximales Element mit Zusicherung

```
1  // {0 < n}
2  i = 0;
3  r = 0;
4  while (i != n) /** inv {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n} */ {
5      if (a[r] < a[i]) {
6          // {\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge a[r] < a[i]} */
7          r = i; }
8      else {
9          // {\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge \neg(a[r] < a[i])} */
10         }
11     i = i + 1; }
12 // {(\forall j. 0 \le j < n \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n}
```

VC	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 {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n} */ {
5      if (a[r] < a[i]) {
6          // {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge a[r] < a[i])} */
7          r = i; }
8      else {
9          // {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge \neg(a[r] < a[i]))} */
10         }
11     i = i + 1; }
12 // {(\forall j. 0 \le j < n \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n}
```

VC	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		$\emptyset$

Maximales Element mit Zusicherung

```
1  // {0 < n}
2  i = 0;
3  r = 0;
4  while (i != n) /** inv {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r]) \wedge 0 \le r < n} */
   {
5      if (a[r] < a[i]) {
6          // {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge a[r] < a[i])} */
7          r = i; }
8      else {
9          // {(\forall j. 0 \le j < i \longrightarrow a[j] \le a[r] \wedge 0 \le r < n \wedge \neg(a[r] < a[i]))} */
10         }
11     i = i + 1; }
12 // {(\forall j. 0 \le j < n \longrightarrow a[j] \le a[r]) \wedge 0 \le 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.
- ▶ Nächste Woche: warum eigentlich immer **rückwärts**?