Beweis:

Sei $\pi = (u_0, \sigma_0) s_1 (u_1, \sigma_1) \ldots s_m (u_m, \sigma_m)$

Gelte: $\sigma_0 \models A$.

Wir müssen zeigen: $\sigma_m \models B$.

Idee:

Induktion nach der Länge m der Programmausführung :-)

$m = 0$:

Der Endpunkt der Ausführung ist gleich dem Startpunkt

$\Longrightarrow$ $\sigma_0 = \sigma_m$ und $A \equiv B$

$\Longrightarrow$ Behauptung gilt :-)

$m > 0$**:**

Annahme: Die Behauptung gilt bereits für $m - 1$.

Sei B' die Zusicherung am Punkt u_{m-1}.

$\Longrightarrow \quad \sigma_{m-1} \models B'$

$m > 0$:

Annahme: Die Behauptung gilt bereits für $m - 1$.

Sei B' die Zusicherung am Punkt u_{m-1}.

$\Longrightarrow \quad \sigma_{m-1} \models B'$

Behandeln wir erst einmal Tests $s_m \equiv b$.

Dann ist insbesondere $\sigma_{m-1} = \sigma_m$:-)

$m > 0$:

Annahme: Die Behauptung gilt bereits für $m-1$.

Sei B' die Zusicherung am Punkt u_{m-1}.

$\Longrightarrow \quad \sigma_{m-1} \models B'$

Behandeln wir erst einmal Tests $s_m \equiv b$.

Dann ist insbesondere $\sigma_{m-1} = \sigma_m$:-)

1. Möglichkeit: $\sigma_m \models b$

$\Longrightarrow \quad B' \Rightarrow \mathbf{WP}[\![b]\!]\,(C, B)$ wobei

$\mathbf{WP}[\![b]\!]\,(C, B) \equiv (\neg b \Rightarrow C) \wedge (b \Rightarrow B)$

$\Longrightarrow \quad \sigma_m \models b \wedge (b \Rightarrow B)$

$\Longrightarrow \quad \sigma_m \models B$:-)

$m > 0$**:**

Annahme: Die Behauptung gilt bereits für $m - 1$.

Sei B' die Zusicherung am Punkt u_{m-1}.

$\Longrightarrow \quad \sigma_{m-1} \models B'$

Behandeln wir erst einmal Tests $s_m \equiv b$.

Dann ist insbesondere $\sigma_{m-1} = \sigma_m$:-)

2. Möglichkeit: $\sigma_m \models \neg b$

$\Longrightarrow \quad B' \Rightarrow \mathbf{WP}[\![b]\!]\,(B, C)$ wobei

$\mathbf{WP}[\![b]\!]\,(B, C) \equiv (\neg b \Rightarrow B) \wedge (b \Rightarrow C)$

$\Longrightarrow \quad \sigma_m \models \neg b \wedge (\neg b \Rightarrow B)$

$\Longrightarrow \quad \sigma_m \models B$:-)

$m > 0$**:**

Annahme: Die Behauptung gilt bereits für $m - 1$.

Sei B' die Zusicherung am Punkt u_{m-1}.

$\Longrightarrow \quad \sigma_{m-1} \models B'$

Nun behandeln wir Anweisungen.

1. Möglichkeit: $s_m \equiv \;;$

Dann gilt:
- $\sigma_{m-1} = \sigma_m$
- $\mathbf{WP}[\![\,;\,]\!]\,(B) \equiv B$

$\Longrightarrow \quad B' \Rightarrow B$

$\Longrightarrow \quad \sigma_{m-1} = \sigma_m \models B$:-)

$m > 0$:

Annahme: Die Behauptung gilt bereits für $m-1$.

Sei B' die Zusicherung am Punkt u_{m-1}.

$\Longrightarrow \quad \sigma_{m-1} \models B'$

Nun behandeln wir Anweisungen.

2. Möglichkeit: $s_m \equiv$ `write(e);`

Dann gilt:

- $\sigma_{m-1} = \sigma_m$
- $\mathbf{WP}[\![\texttt{;write(e)}]\!]\,(B) \equiv B$

$\Longrightarrow \quad B' \Rightarrow B$

$\Longrightarrow \quad \sigma_{m-1} = \sigma_m \models B \quad$:-)

$m > 0$:

Annahme: Die Behauptung gilt bereits für $m - 1$.

Sei B' die Zusicherung am Punkt u_{m-1}.

$\Longrightarrow \quad \sigma_{m-1} \models B'$

Nun behandeln wir Anweisungen.

3. Möglichkeit: $s_m \equiv$ `x = read();`

Dann gilt:

- $\sigma_m = \sigma_{m-1} \oplus \{x \mapsto c\}$ für ein $c \in \mathbb{Z}$
- $\mathbf{WP}[\![\texttt{x = read();}]\!] (B) \equiv \forall\, x.\, B$

$\Longrightarrow \quad B' \Rightarrow \forall\, x.\, B \Rightarrow B[c/x]$

$\Longrightarrow \quad \sigma_m \models B$:-)

$m > 0$**:**

Annahme: Die Behauptung gilt bereits für $m - 1$.

Sei B' die Zusicherung am Punkt u_{m-1}.

$\Longrightarrow \quad \sigma_{m-1} \models B'$

Nun behandeln wir Anweisungen.

4. Möglichkeit: $s_m \equiv$ `x = e;` Dann gilt:

- $\sigma_m = \sigma_{m-1} \oplus \{x \mapsto c\}$ für c = Wert von `e` in σ_{m-1}
- $\mathbf{WP}[\![$`x = e;`$]\!]\,(B) \equiv B[e/x]$

$\Longrightarrow \quad B' \Rightarrow B[e/x] \qquad \Longrightarrow \quad \sigma_{m-1} \models B[c/x]$

$\Longrightarrow \quad \sigma_m \models B$:-))

Fazit:

- Das Verfahren nach Floyd ermöglicht uns zu beweisen, dass eine Zusicherung B bei Erreichen eines Programmpunkts stets (bzw. unter geeigneten Zusatzannahmen :-) gilt ...
- Zur Durchführung benötigen wir:
  - Zusicherungen an jedem Programmpunkt :-(
  - Nachweis, dass die Zusicherungen lokal konsistent sind

  $\Longrightarrow$ Logik, automatisches Beweisen

3.3 Optimierung

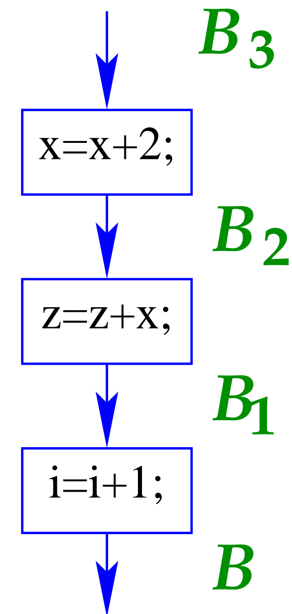
Ziel: Verringerung der benötigten Zusicherungen

Beobachtung:

Hat das Programm keine Schleifen, können wir für jeden Programmpunkt eine hinreichende Vorbedingung ausrechnen !!!

Beispiel:

```
x = x+2;
z = z+x;
i = i+1;
```



Beispiel (Fort.):

Sei $\quad B \equiv z = i^2 \;\wedge\; x = 2i - 1$

Dann rechnen wir:

$$
\begin{aligned}
B_1 \quad \equiv \quad \mathbf{WP}[\![\texttt{i = i+1;}]\!](B) \quad & \equiv \quad z = (i+1)^2 \wedge x = 2(i+1) - 1 \\
& \equiv \quad z = (i+1)^2 \wedge x = 2i + 1
\end{aligned}
$$

Beispiel (Fort.):

Sei $\quad B \equiv z = i^2 \ \wedge \ x = 2i - 1$

Dann rechnen wir:

$$
\begin{array}{lllll}
B_1 & \equiv & \mathbf{WP}[\![\texttt{i = i+1;}]\!](B) & \equiv & z = (i+1)^2 \wedge x = 2(i+1) - 1 \\
 & & & \equiv & z = (i+1)^2 \wedge x = 2i + 1 \\
B_2 & \equiv & \mathbf{WP}[\![\texttt{z = z+x;}]\!](B_1) & \equiv & z + x = (i+1)^2 \wedge x = 2i + 1 \\
 & & & \equiv & z = i^2 \wedge x = 2i + 1
\end{array}
$$

Beispiel (Fort.):

Sei $\quad B \equiv z = i^2 \;\wedge\; x = 2i - 1$

Dann rechnen wir:

$$
\begin{array}{lcllcl}
B_1 & \equiv & \mathbf{WP}[\![\texttt{i = i+1;}]\!](B) & \equiv & z = (i+1)^2 \wedge x = 2(i+1) - 1 \\
& & & \equiv & z = (i+1)^2 \wedge x = 2i + 1 \\
B_2 & \equiv & \mathbf{WP}[\![\texttt{z = z+x;}]\!](B_1) & \equiv & z + x = (i+1)^2 \wedge x = 2i + 1 \\
& & & \equiv & z = i^2 \wedge x = 2i + 1 \\
B_3 & \equiv & \mathbf{WP}[\![\texttt{x = x+2;}]\!](B_2) & \equiv & z = i^2 \wedge x + 2 = 2i + 1 \\
& & & \equiv & z = i^2 \wedge x = 2i - 1 \\
& & & \equiv & B \qquad \text{;-)}
\end{array}
$$

Idee:

- Für jede Schleife wähle einen Programmpunkt aus.

  Sinnvolle Auswahlen:

  $\longrightarrow$ Vor der Bedingung;

  $\longrightarrow$ Am Beginn des Rumpfs;

  $\longrightarrow$ Am Ende des Rumpfs ...

- Stelle für jeden gewählten Punkt eine Zusicherung bereit

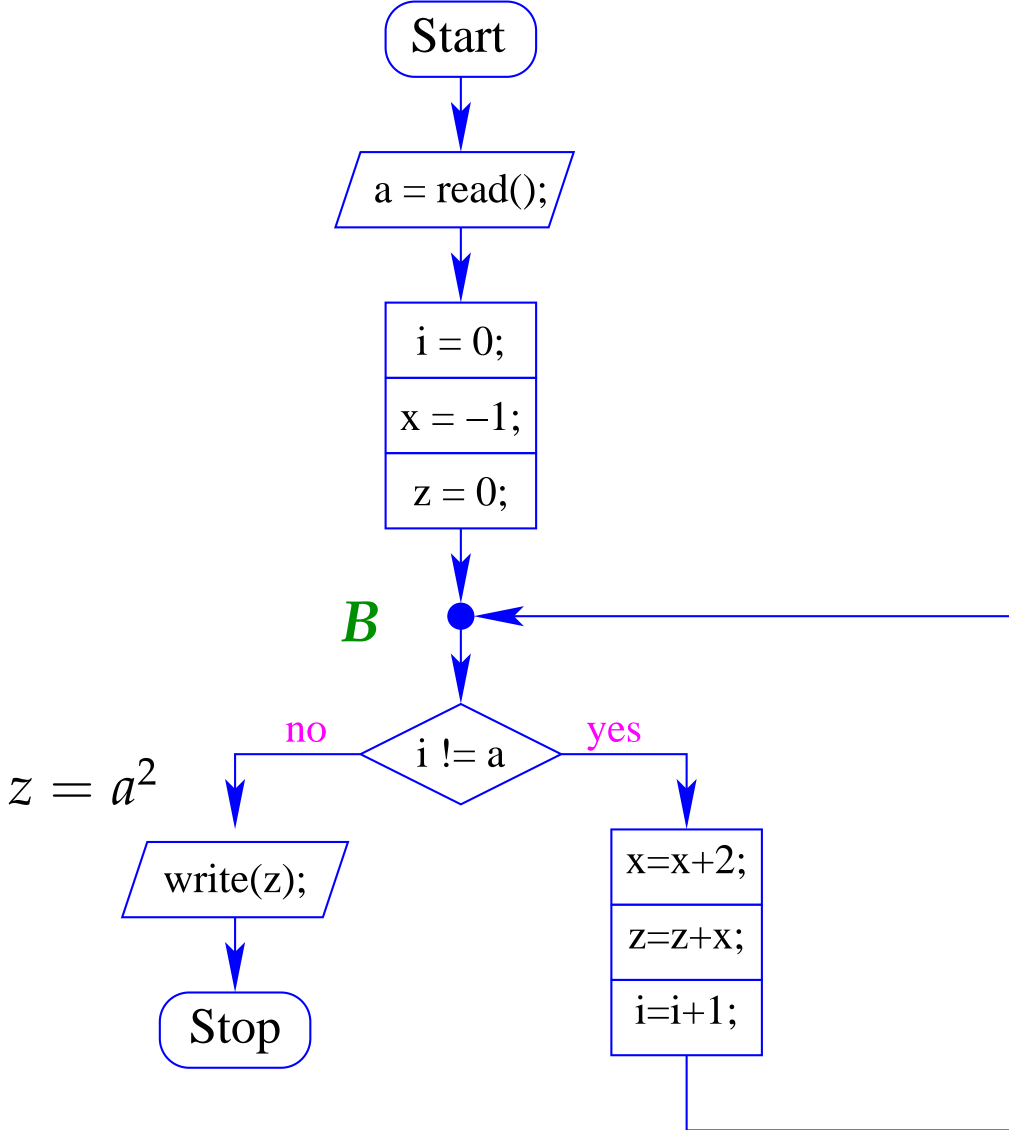
  $\Longrightarrow$ Schleifen-Invariante

- Für alle übrigen Programmpunkte bestimmen wir Zusicherungen mithilfe $\mathbf{WP}[\![\ldots]\!]()$:-)

Beispiel:

```
int a, i, x, z;
a = read();
i = 0;
x = -1;
z = 0;
while (i != a) {
    x = x+2;
    z = z+x;
    i = i+1;
}
assert(z==a*a);
write(z);
```

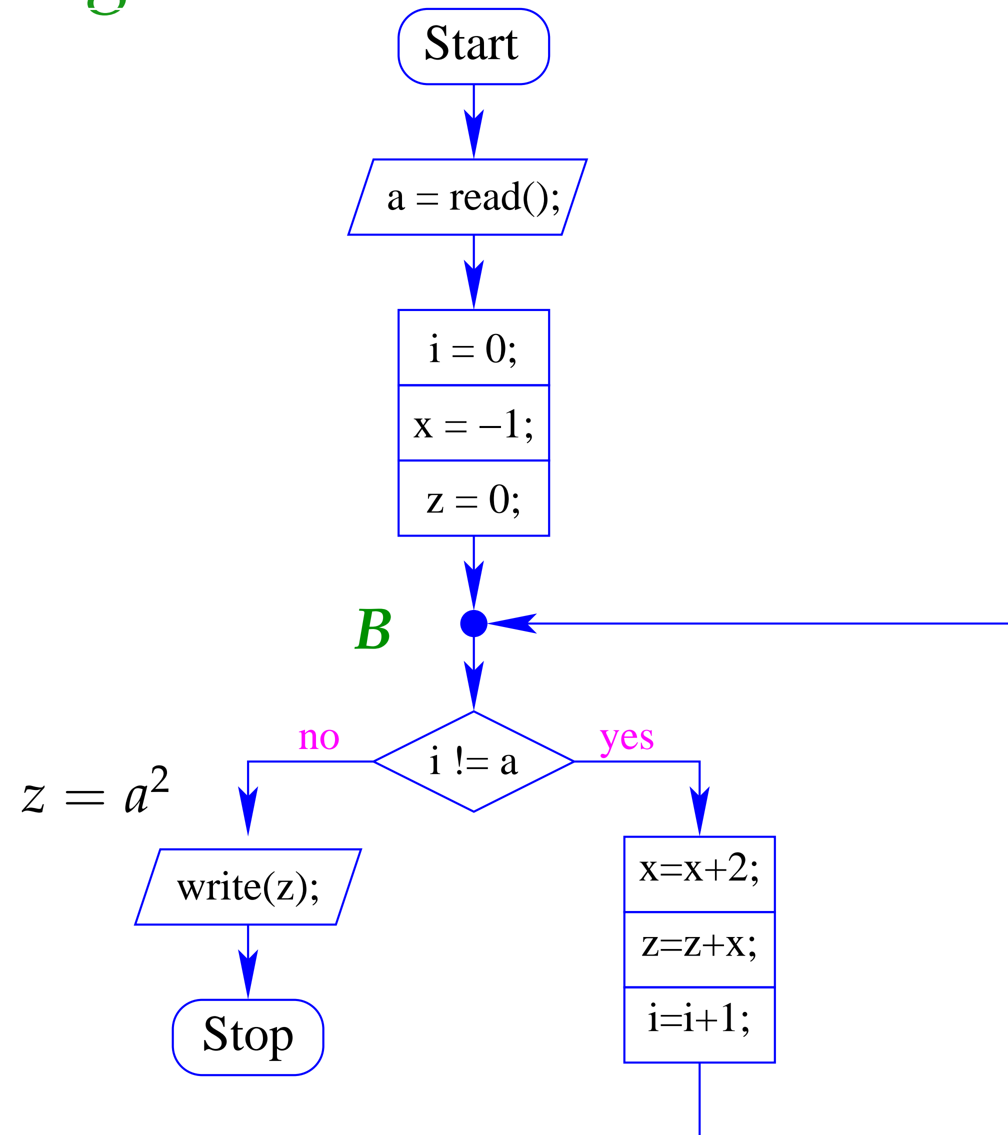
Beispiel:



Wir überprüfen:

$$
\begin{aligned}
\mathbf{WP}[\![\texttt{i != a}]\!](z = a^2, B) \\
&\equiv \quad (i = a \wedge z = a^2) \ \vee \ (i \neq a \wedge B) \ \vee \ (z = a^2 \wedge B) \\
&\equiv \quad (i = a \wedge z = a^2) \ \vee \ (i \neq a \wedge z = i^2 \wedge x = 2i - 1) \ \vee \\
&\qquad\qquad\qquad (z = a^2 \wedge z = i^2 \wedge x = 2i - 1) \\
&\Leftarrow \quad (i \neq a \wedge z = i^2 \wedge x = 2i - 1) \ \vee \ (i = a \wedge z = i^2 \wedge x = 2i - 1) \\
&\equiv \quad z = i^2 \wedge x = 2i - 1 \quad \equiv \quad B \qquad\qquad :\!-)
\end{aligned}
$$

Orientierung:



Wir überprüfen:

$$\begin{array}{lll}
\textbf{WP}[\![\texttt{z = 0;}]\!](B) & \equiv & 0 = i^2 \wedge x = 2i - 1 \\
 & \equiv & i = 0 \wedge x = -1 \\
\textbf{WP}[\![\texttt{x = -1;}]\!](i = 0 \wedge x = -1) & \equiv & i = 0 \\
\textbf{WP}[\![\texttt{i = 0;}]\!](i = 0) & \equiv & \textbf{true} \\
\textbf{WP}[\![\texttt{a = read();}]\!](\textbf{true}) & \equiv & \textbf{true} \qquad \text{:-))}
\end{array}$$

3.4 Terminierung

Problem:

- Mit unserer Beweistechnik können wir nur beweisen, dass eine Eigenschaft gilt wann immer wir einen Programmpunkt erreichen !!!
- Wie können wir aber garantieren, dass das Programm immer terminiert ?
- Wie können wir eine Bedingung finden, unter der das Programm immer terminiert ??

Beispiele:

- Das ggT-Programm terminiert nur für Eingaben a, b mit: $a > 0$ und $b > 0$.
- Das Quadrier-Programm terminiert nur für Eingaben $a \geq 0$.
- `while (true) ;` terminiert nie.
- Programme ohne Schleifen terminieren immer :-)

Beispiele:

- Das ggT-Programm terminiert nur für Eingaben a, b mit: $a > 0$ und $b > 0$.
- Das Quadrier-Programm terminiert nur für Eingaben $a \geq 0$.
- `while (true) ;` terminiert nie.
- Programme ohne Schleifen terminieren immer :-)

Lässt sich dieses Beispiel verallgemeinern ??

Beispiel:

```
int i, j, t;
t = 0;
i = read();
while (i>0) {
   j = read();
   while (j>0) { t = t+1; j = j-1; }
   i = i-1;
}
write(t);
```

- Die gelesene Zahl `i` (falls positiv) gibt an, wie oft eine Zahl j eingelesen wird.
- Die Gesamtlaufzeit ist (im wesentlichen :-) die Summe der positiven für j gelesenen Werte

Beispiel:

```
int i, j, t;
t = 0;
i = read();
while (i>0) {
    j = read();
    while (j>0) { t = t+1; j = j-1; }
    i = i-1;
}
write(t);
```

- Die gelesene Zahl `i` (falls positiv) gibt an, wie oft eine Zahl j eingelesen wird.
- Die Gesamtlaufzeit ist (im wesentlichen :-) die Summe der positiven für j gelesenen Werte

$\Longrightarrow$ das Programm terminiert immer !!!

Programme nur mit `for`-Schleifen der Form:

```
for (i=n; i>0; i--) {...}
```

// im Rumpf wird `i` nicht modifiziert

... terminieren ebenfalls immer :-))

Programme nur mit `for`-Schleifen der Form:

```
for (i=n; i>0; i--) {...}
```

// im Rumpf wird `i` nicht modifiziert

... terminieren ebenfalls immer :-))

Frage:

Wie können wir aus dieser Beobachtung eine Methode machen, die auf beliebige Schleifen anwendbar ist ?

Idee:

- Weise nach, dass jede Scheife nur endlich oft durchlaufen wird
  ...
- Finde für jede Schleife eine Kenngröße r, die zwei Eigenschaften hat:
  - (1) Wenn immer der Rumpf betreten wird, ist $r > 0$;
  - (2) Bei jedem Schleifen-Durchlauf wird r kleiner :-)
- Transformiere das Programm so, dass es neben der normalen Programmausführung zusätzlich die Kenngrößen r mitberechnet.
- Verifiziere, dass (1) und (2) gelten :-)

Beispiel: Sicheres ggT-Programm

```
int a, b, x, y;
a = read(); b = read();
if (a < 0) x = -a; else x = a;
if (b < 0) y = -b; else y = b;
if (x == 0) write(y);
else if (y == 0) write(x);
     else {
        while (x != y)
           if (y > x) y = y-x;
           else       x = x-y;
        write(x);
}
```

Wir wählen: $r = x + y$

Transformation:

```
int a, b, x, y, r;
a = read(); b = read();
if (a < 0) x = -a; else x = a;
if (b < 0) y = -b; else y = b;
if (x == 0) write(y);
else if (y == 0) write(x);
     else { r = x+y;
        while (x != y) {
           if (y > x) y = y-x;
           else       x = x-y;
           r = x+y; }
        write(x);
}
```