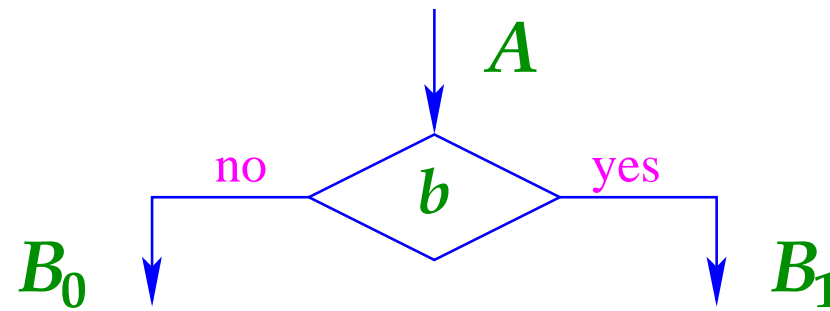


Teilproblem 2: Verzweigungen



Es sollte gelten:

- $A \wedge \neg b \Rightarrow B_0$ und
- $A \wedge b \Rightarrow B_1$.

Das ist der Fall, falls A die schwächste Vorbedingung der Verzweigung:

$$\mathbf{WP}[[b]] (B_0, B_1) \equiv ((\neg b) \Rightarrow B_0) \wedge (b \Rightarrow B_1)$$

impliziert :-)

Das ist der Fall, falls A die **schwächste Vorbedingung** der Verzweigung:

$$\mathbf{WP}[[b]] (B_0, B_1) \equiv ((\neg b) \Rightarrow B_0) \wedge (b \Rightarrow B_1)$$

impliziert :-)

Die schwächste Vorbedingung können wir umschreiben in:

$$\begin{aligned} \mathbf{WP}[[b]] (B_0, B_1) &\equiv (b \vee B_0) \wedge (\neg b \vee B_1) \\ &\equiv (\neg b \wedge B_0) \vee (b \wedge B_1) \vee (B_0 \wedge B_1) \\ &\equiv (\neg b \wedge B_0) \vee (b \wedge B_1) \end{aligned}$$

Beispiel:

$$B_0 \equiv x > y \wedge y > 0$$

$$B_1 \equiv x > 0 \wedge y > x$$

Sei b die Bedingung $y > x$.

Dann ist die schwächste Vorbedingung:

Beispiel:

$$B_0 \equiv x > y \wedge y > 0$$

$$B_1 \equiv x > 0 \wedge y > x$$

Sei b die Bedingung $y > x$.

Dann ist die schwächste Vorbedingung:

$$\begin{aligned} & (x > y \wedge y > 0) \vee (x > 0 \wedge y > x) \\ \equiv & x > 0 \wedge y > 0 \wedge x \neq y \end{aligned}$$

... im GGT-Beispiel:

$$b \equiv y > x$$

$$\neg b \wedge A \equiv x \geq y \wedge \text{ggT}(a, b) = \text{ggT}(x, y)$$

$$b \wedge A \equiv y > x \wedge \text{ggT}(a, b) = \text{ggT}(x, y)$$

... im GGT-Beispiel:

$$b \equiv y > x$$

$$\neg b \wedge A \equiv x \geq y \wedge \text{ggT}(a, b) = \text{ggT}(x, y)$$

$$b \wedge A \equiv y > x \wedge \text{ggT}(a, b) = \text{ggT}(x, y)$$

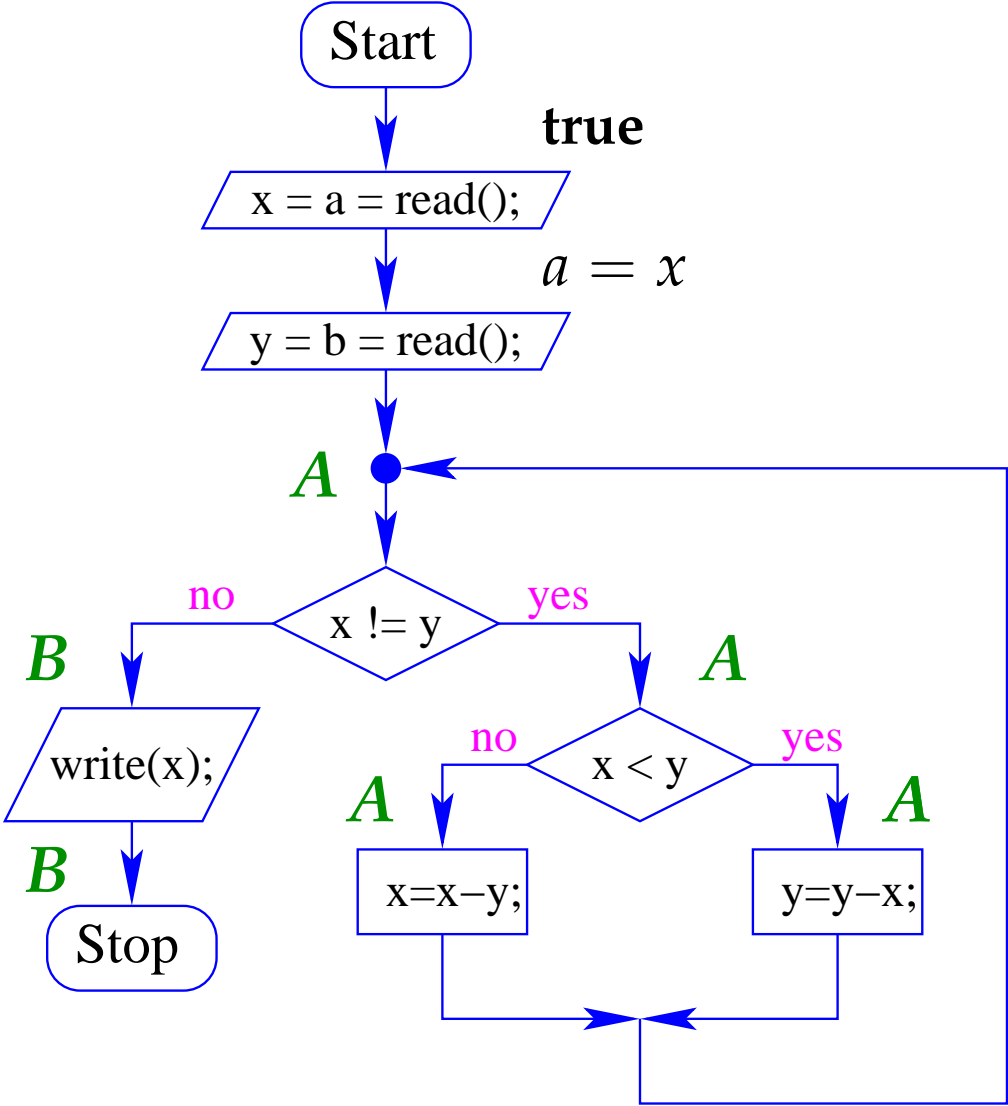


Die schwächste Vorbedingung ist:

$$\text{ggT}(a, b) = \text{ggT}(x, y)$$

... also genau A :-)

Orientierung:



Analog argumentieren wir für die Zusicherung vor der Schleife:

$$b \equiv y \neq x$$

$$\neg b \wedge B \equiv B$$

$$b \wedge A \equiv A \wedge x \neq y$$

\implies $A \equiv (A \wedge x = y) \vee (A \wedge x \neq y)$ ist die schwächste Vorbedingung für die Verzweigung $:-)$

Zusammenfassung der Methode:

- Annotiere jeden Programmpunkt mit einer Zusicherung.
- Überprüfe für jede Anweisung s zwischen zwei Zusicherungen A und B , dass A die schwächste Vorbedingung von s für B impliziert, d.h.:

$$A \Rightarrow \mathbf{WP}[[s]](B)$$

- Überprüfe entsprechend für jede Verzweigung mit Bedingung b , ob die Zusicherung A vor der Verzweigung die schwächste Vorbedingung für die Nachbedingungen B_0 und B_1 der Verzweigung impliziert, d.h.

$$A \Rightarrow \mathbf{WP}[[b]](B_0, B_1)$$

Solche Annotierungen nennen wir **lokal konsistent**.

1.2 Korrektheit

Fragen:

- Welche Programm-Eigenschaften können wir mithilfe lokal konsistenter Annotierungen garantieren ?
- Wie können wir nachweisen, dass unser Verfahren **keine falschen Ergebnisse** liefert ??

Erinnerung (1):

- In **MiniJava** können wir ein Zustand σ aus einer **Variablen-Belegung**, d.h. einer Abbildung der Programm-Variablen auf ganze Zahlen (ihren Werten), z.B.:

$$\sigma = \{x \mapsto 5, y \mapsto -42\}$$

Erinnerung (1):

- In **MiniJava** können wir ein Zustand σ aus einer **Variablen-Belegung**, d.h. einer Abbildung der Programm-Variablen auf ganze Zahlen (ihren Werten), z.B.:

$$\sigma = \{x \mapsto 5, y \mapsto -42\}$$

- Ein Zustand σ **erfüllt** eine Zusicherung A , falls

$$A[\sigma(x)/x]_{x \in A}$$

// wir substituieren jede Variable in A durch ihren Wert in σ
eine **wahre** Aussage ist, d.h. äquivalent **true**.

Wir schreiben: $\sigma \models A$.

Beispiel:

$$\begin{aligned}\sigma &= \{x \mapsto 5, y \mapsto 2\} \\ A &\equiv (x > y) \\ A[5/x, 2/y] &\equiv (5 > 2) \\ &\equiv \mathbf{true}\end{aligned}$$

Beispiel:

$$\sigma = \{x \mapsto 5, y \mapsto 2\}$$

$$A \equiv (x > y)$$

$$A[5/x, 2/y] \equiv (5 > 2)$$

$$\equiv \mathbf{true}$$

$$\sigma = \{x \mapsto 5, y \mapsto 12\}$$

$$A \equiv (x > y)$$

$$A[5/x, 12/y] \equiv (5 > 12)$$

$$\equiv \mathbf{false}$$

Triviale Eigenschaften:

$\sigma \models \mathbf{true}$ für jedes σ

$\sigma \models \mathbf{false}$ für kein σ

$\sigma \models A_1$ und $\sigma \models A_2$ ist äquivalent zu

$\sigma \models A_1 \wedge A_2$

$\sigma \models A_1$ oder $\sigma \models A_2$ ist äquivalent zu

$\sigma \models A_1 \vee A_2$

Erinnerung (2):

- Eine Programmausführung π durchläuft einen **Pfad** im Kontrollfluss-Graphen :-)
- Sie beginnt in einem Programmpunkt u_0 in einem Anfangszustand σ_0 . und führt in einen Programmpunkt u_m und einen Endzustand σ_m .
- Jeder Schritt der Programm-Ausführung führt eine Aktion aus und ändert Programmpunkt und Zustand :-)

Erinnerung (2):

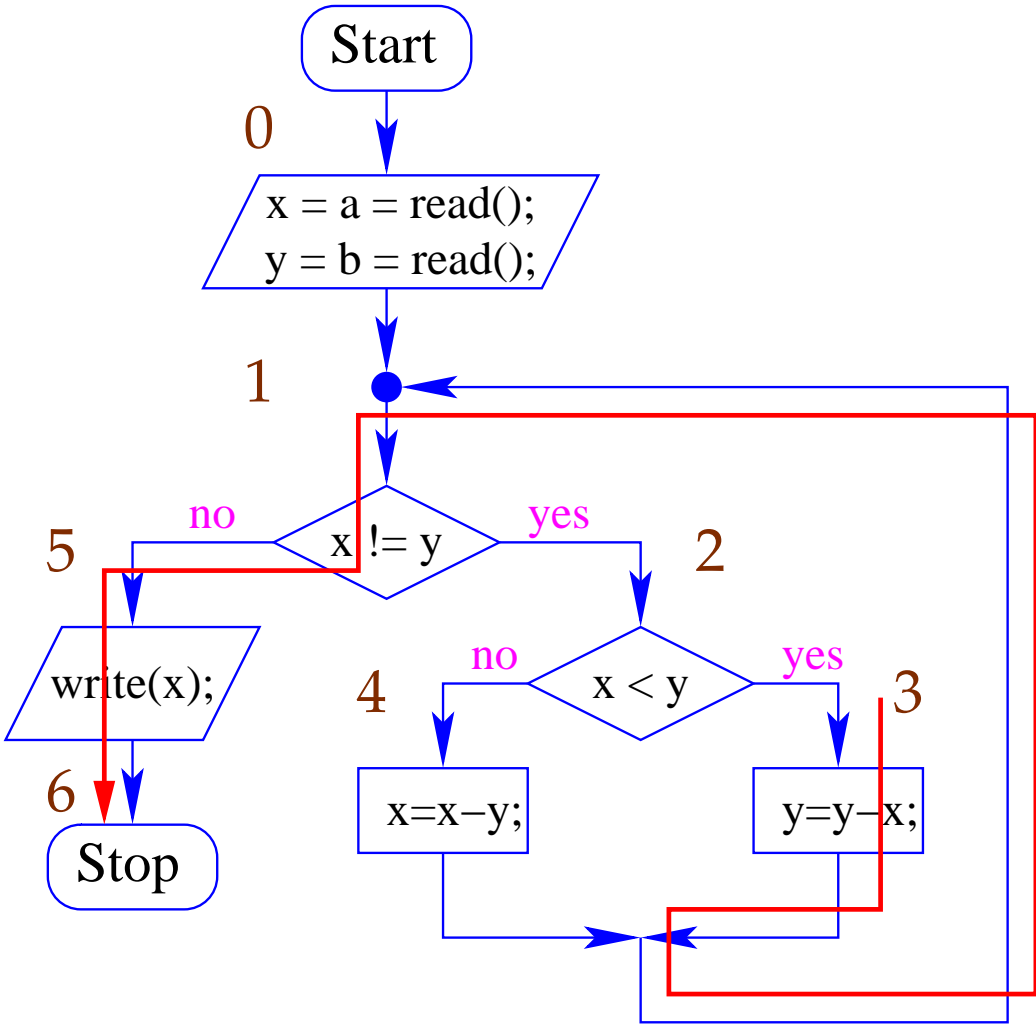
- Eine Programmausführung π durchläuft einen **Pfad** im Kontrollfluss-Graphen :-)
- Sie beginnt in einem Programmpunkt u_0 in einem Anfangszustand σ_0 und führt in einen Programmpunkt u_m und einen Endzustand σ_m .
- Jeder Schritt der Programm-Ausführung führt eine Aktion aus und ändert Programmpunkt und Zustand :-)

\implies Wir können π als Folge darstellen:

$$(u_0, \sigma_0) s_1 (u_1, \sigma_1) \dots s_m (u_m, \sigma_m)$$

wobei die s_i Elemente des Kontrollfluss-Graphen sind, d.h. Anweisungen oder Bedingungen ...

Beispiel:



Nehmen wir an, wir starten in Punkt **3** mit $\{x \mapsto 6, y \mapsto 12\}$.

Dann ergibt sich die **Programmausführung**:

$$\begin{aligned} \pi = & \quad (3, \{x \mapsto 6, y \mapsto 12\}) \quad y = y - x; \\ & \quad (1, \{x \mapsto 6, y \mapsto 6\}) \quad !(x \neq y) \\ & \quad (5, \{x \mapsto 6, y \mapsto 6\}) \quad \text{write}(x); \\ & \quad (6, \{x \mapsto 6, y \mapsto 6\}) \end{aligned}$$

Satz:

Sei p ein MiniJava-Programm, Sei π eine Programmausführung, die im Programmpunkt u startet und zum Programmpunkt v führt.

Annahmen:

- Die Programmpunkte von p seien lokal konsistent mit Zusicherungen annotiert.
- Der Programmpunkt u sei mit A annotiert.
- Der Programmpunkt v sei mit B annotiert.

Satz:

Sei p ein MiniJava-Programm, Sei π eine Programmausführung, die im Programmpunkt u startet und zum Programmpunkt v führt.

Annahmen:

- Die Programmpunkte von p seien lokal konsistent mit Zusicherungen annotiert.
- Der Programmpunkt u sei mit A annotiert.
- Der Programmpunkt v sei mit B annotiert.

Dann gilt:

Erfüllt der Anfangszustand von π die Zusicherung A , dann erfüllt der Endzustand die Zusicherung B .

Bemerkungen:

- Ist der Startpunkt des Programms mit **true** annotiert, dann erfüllt **jede** Programmausführung, die den Programmpunkt v erreicht, die Zusicherung an v :-)
- Zum Nachweis, dass eine Zusicherung A an einem Programmpunkt v gilt, benötigen wir eine lokal konsistente Annotierung mit zwei Eigenschaften:
 - (1) der Startpunkt ist mit **true** annotiert;
 - (2) Die Zusicherung an v **impliziert** A :-)

Bemerkungen:

- Ist der Startpunkt des Programms mit **true** annotiert, dann erfüllt **jede** Programmausführung, die den Programmpunkt v erreicht, die Zusicherung an v **:-)**
- Zum Nachweis, dass eine Zusicherung A an einem Programmpunkt v gilt, benötigen wir eine lokal konsistente Annotierung mit zwei Eigenschaften:
 - (1) der Startpunkt ist mit **true** annotiert;
 - (2) Die Zusicherung an v **impliziert** A **:-)**
- Unser Verfahren gibt (vorerst) keine Garantie, dass v überhaupt erreicht wird **!!!**
- Falls ein Programmpunkt v mit der Zusicherung **false** annotiert werden kann, kann v **nie** erreicht werden **:-))**

Beweis:

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

Gelte: $\sigma_0 \models A$.

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

Idee:

Induktion nach der Länge m der Programmausführung :-)

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:
 - Zusicherung **true** am Startpunkt.
 - Zusicherungen an jedem weiteren Programmpunkt $:-)$
 - Nachweis, dass die Zusicherungen lokal konsistent sind
 \implies Logik, automatisches Beweisen

1.3 Optimierung

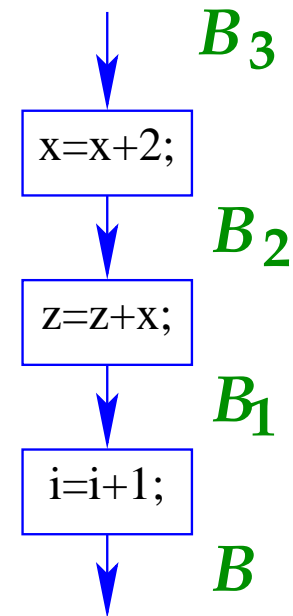
Ziel: Verringerung der benötigten Zusicherungen

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

Dann rechnen wir:

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

Beispiel (Fort.):

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

Dann rechnen wir:

$$B_1 \equiv \mathbf{WP}[[i = i+1;]](B) \equiv z = (i+1)^2 \wedge x = 2(i+1) - 1$$

$$\equiv z = (i+1)^2 \wedge x = 2i + 1$$

$$B_2 \equiv \mathbf{WP}[[z = z+x;]](B_1) \equiv z + x = (i+1)^2 \wedge x = 2i + 1$$

$$\equiv z = i^2 \wedge x = 2i + 1$$

Beispiel (Fort.):

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

Dann rechnen wir:

$$\begin{aligned} B_1 &\equiv \mathbf{WP}[[i = i+1;]](B) &&\equiv z = (i+1)^2 \wedge x = 2(i+1) - 1 \\ &&&\equiv z = (i+1)^2 \wedge x = 2i+1 \\ B_2 &\equiv \mathbf{WP}[[z = z+x;]](B_1) &&\equiv z + x = (i+1)^2 \wedge x = 2i+1 \\ &&&\equiv z = i^2 \wedge x = 2i+1 \\ B_3 &\equiv \mathbf{WP}[[x = x+2;]](B_2) &&\equiv z = i^2 \wedge x + 2 = 2i+1 \\ &&&\equiv z = i^2 \wedge x = 2i-1 \\ &&&\equiv B \quad ;-)) \end{aligned}$$

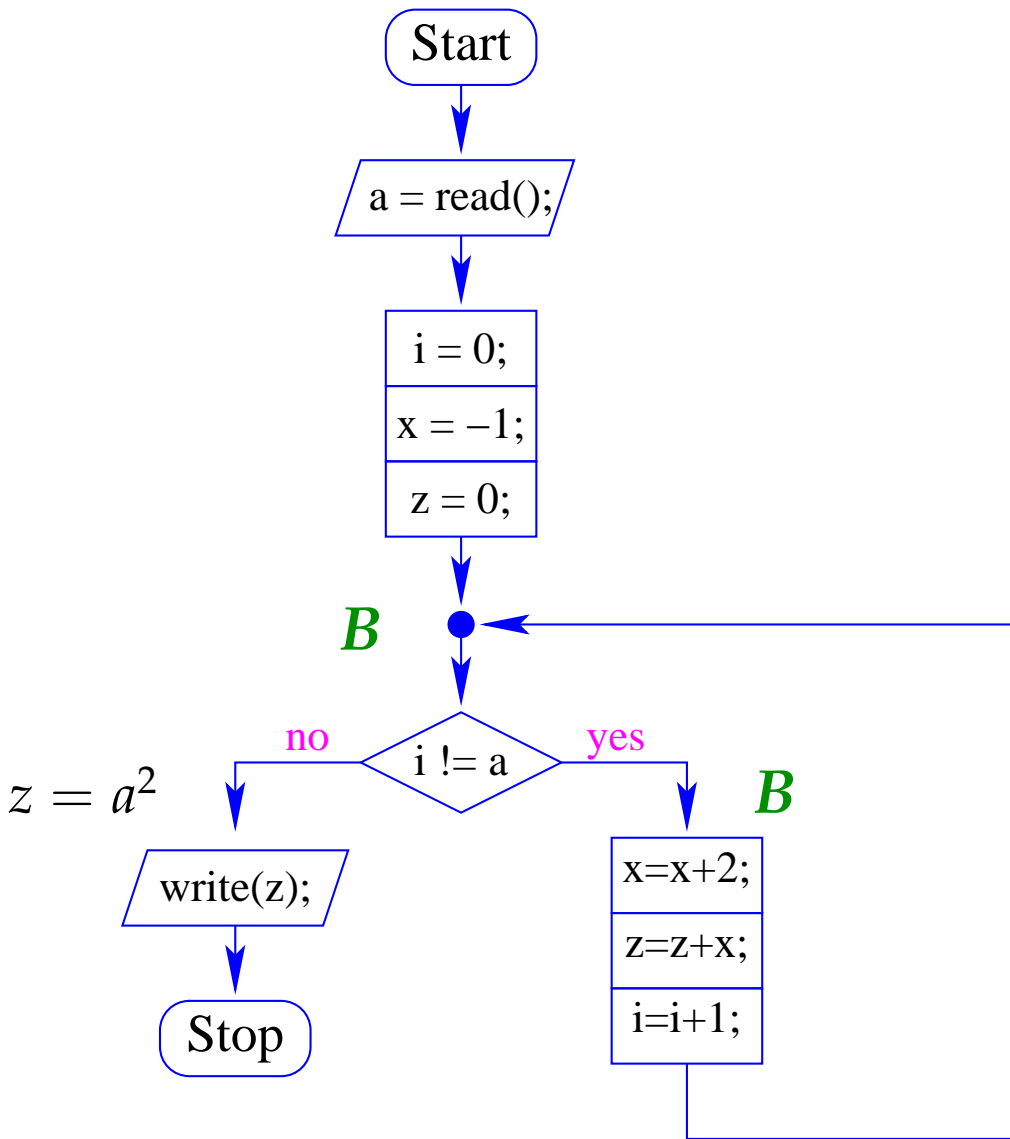
Idee:

- Für jede Schleife wähle **einen** Programmpunkt aus.
Sinnvolle Auswahlen:
 - Vor der Bedingung;
 - Am Beginn des Rumpfs;
 - Am Ende des Rumpfs ...
- Stelle für jeden gewählten Punkt eine Zusicherung bereit
 - ⇒ **Schleifen-Invariante**
- Für alle übrigen Programmpunkte bestimmen wir Zusicherungen mithilfe $\mathbf{WP}[\dots](\) \text{ :-}$

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:

$$\mathbf{WP}[[i \neq a]](z = a^2, B)$$

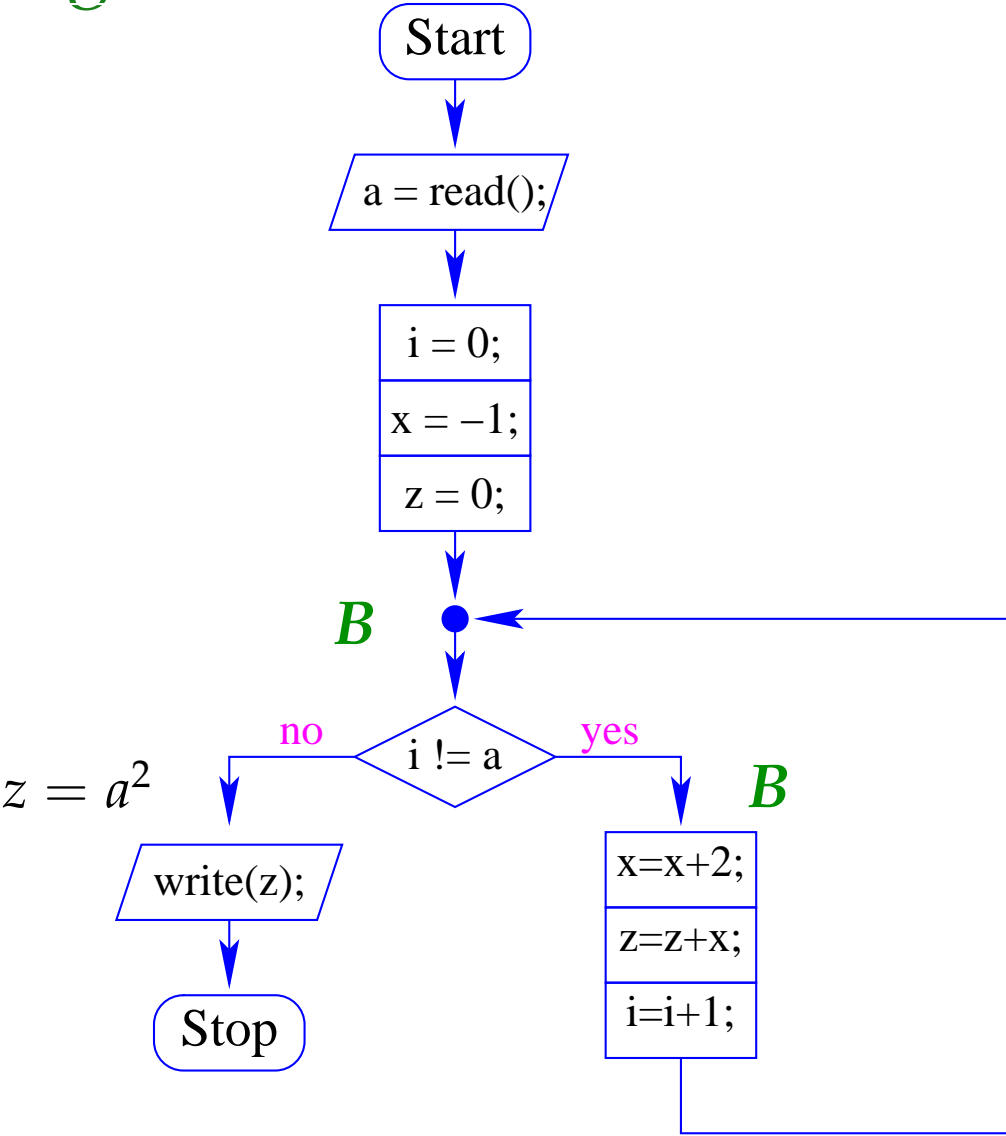
$$\equiv (i = a \wedge z = a^2) \vee (i \neq a \wedge B)$$

$$\equiv (i = a \wedge z = a^2) \vee (i \neq a \wedge z = i^2 \wedge x = 2i - 1)$$

$$\Leftarrow (i \neq a \wedge z = i^2 \wedge x = 2i - 1) \vee (i = a \wedge z = i^2 \wedge x = 2i - 1)$$

$$\equiv z = i^2 \wedge x = 2i - 1 \quad \equiv B \quad \text{: -)}$$

Orientierung:



Wir überprüfen:

$$\begin{aligned}\mathbf{WP}[\mathbf{z} = 0;](B) &\equiv 0 = i^2 \wedge x = 2i - 1 \\ &\equiv i = 0 \wedge x = -1 \\ \mathbf{WP}[\mathbf{x} = -1;](i = 0 \wedge x = -1) &\equiv i = 0 \\ \mathbf{WP}[\mathbf{i} = 0;](i = 0) &\equiv \mathbf{true} \\ \mathbf{WP}[\mathbf{a} = \mathbf{read}();](\mathbf{true}) &\equiv \mathbf{true} \quad \text{: -))}\end{aligned}$$

1.4 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 ??