... im GGT-Programm (1):

Zuweisung: `x = x-y;`

Nachbedingung: A

schwächste Vorbedingung:

$$\begin{aligned} A[x-y/x] &\equiv ggT(a,b) = ggT(x-y,y) \\ &\equiv ggT(a,b) = ggT(x,y) \\ &\equiv A \end{aligned}$$

... im GGT-Programm (2):

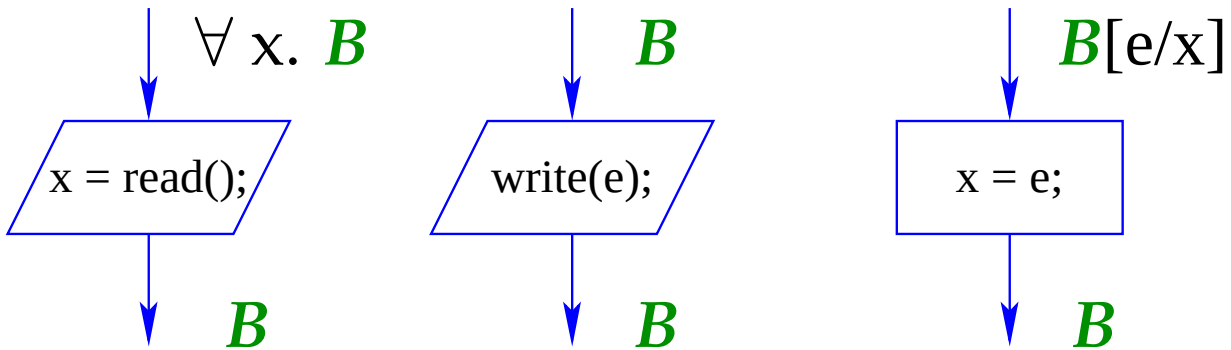
Zuweisung: `y = y-x;`

Nachbedingung: A

schwächste Vorbedingung:

$$\begin{aligned} A[y-x/y] &\equiv ggT(a,b) = ggT(x,y-x) \\ &\equiv ggT(a,b) = ggT(x,y) \\ &\equiv A \end{aligned}$$

Zusammenstellung:



$$\mathbf{WP}[\![\texttt{;}]\!](B) \equiv B$$

$$\mathbf{WP}[\![\texttt{x = e;}]\!](B) \equiv B[e/x]$$

$$\mathbf{WP}[\![\texttt{x = read();}]\!](B) \equiv \forall x.\, B$$

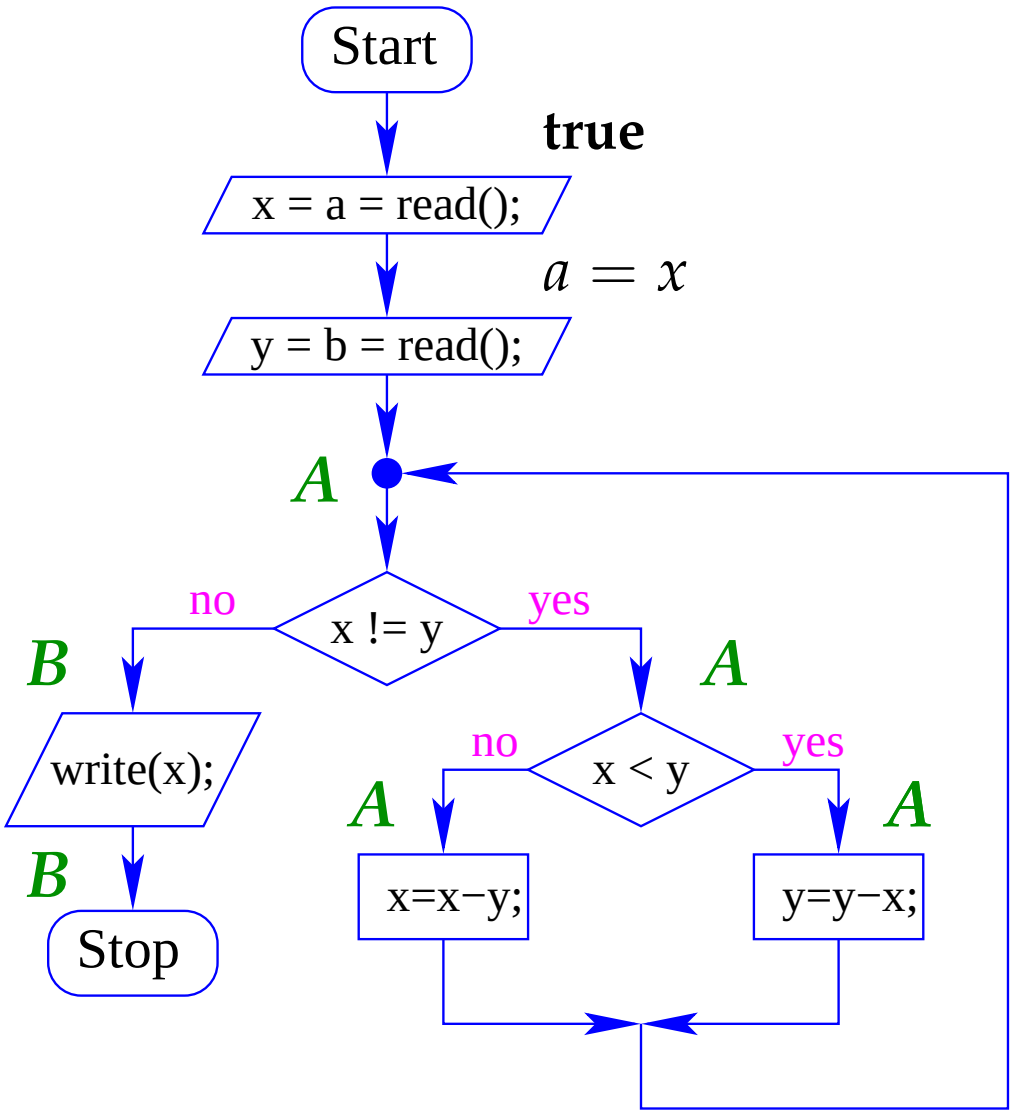
$$\mathbf{WP}[\![\texttt{write(e);}]\!](B) \equiv B$$

Diskussion:

- Die Zusammenstellung liefert für alle Aktionen jeweils die schwächsten Vorbedingungen für eine Nachbedingung B.
- Eine Ausgabe-Anweisung ändert keine Variablen. Deshalb ist da die schwächste Vorbedingung B selbst ;-)
- Eine Eingabe-Anweisung `x=read();` ändert die Variable `x` auf unvorhersehbare Weise.

  Damit nach der Eingabe B gelten kann, muss B vor der Eingabe für jedes mögliche `x` gelten ;-)

Orientierung:



Für die Anweisungen: `b = read(); y = b;` berechnen wir:

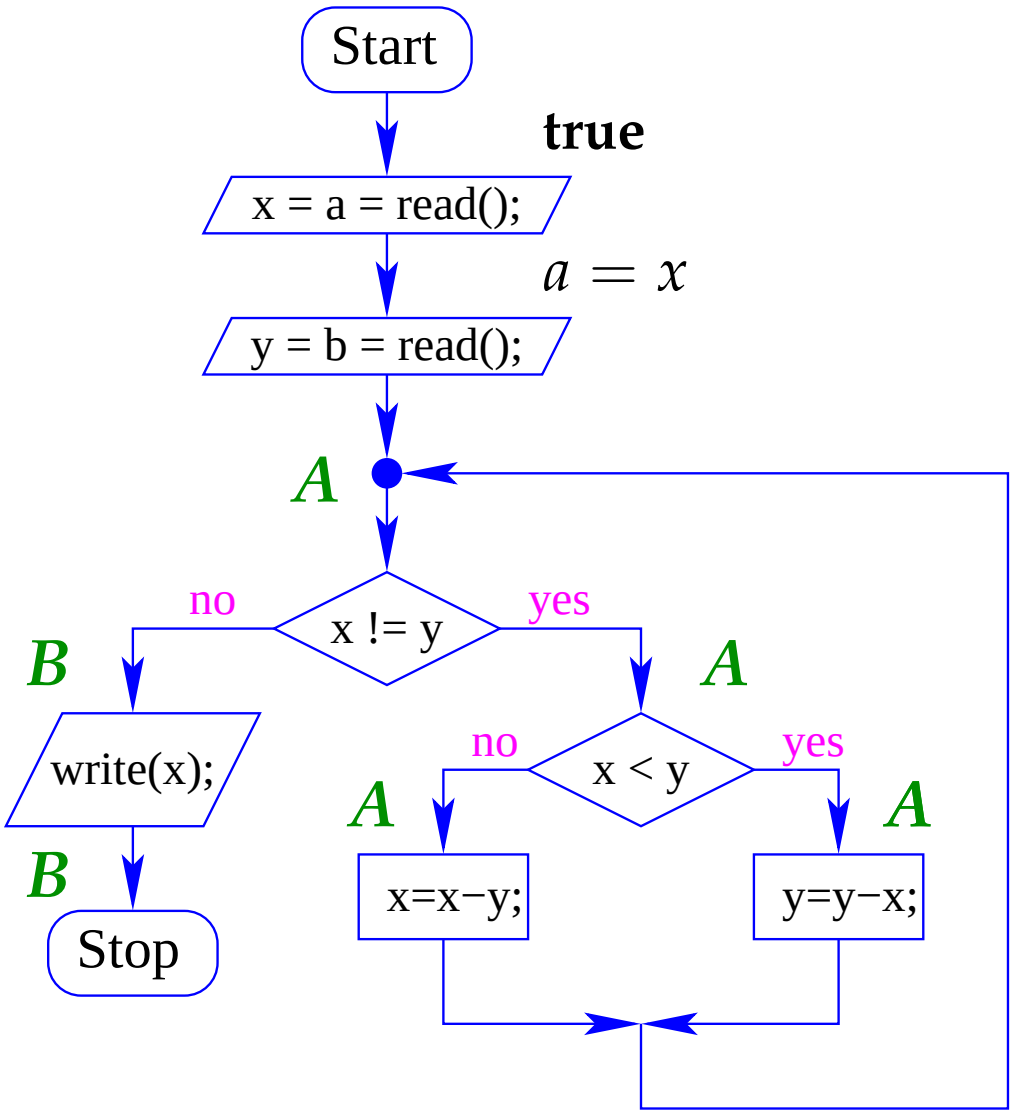
$$\begin{aligned} \mathbf{WP}[\![\texttt{y = b;}]\!]\,(A) &\equiv A[b/y] \\ &\equiv ggT(a,b) = ggT(x,b) \end{aligned}$$

Für die Anweisungen: `b = read(); y = b;` berechnen wir:

$$\begin{aligned}
\mathbf{WP}[\![\texttt{y = b;}]\!]\,(A) &\equiv A[b/y] \\
&\equiv ggT(a,b) = ggT(x,b)
\end{aligned}$$

$$\begin{aligned}
\mathbf{WP}[\![\texttt{b = read();}]\!]\,(ggT(a,b) = ggT(x,b)) & \\
&\equiv \forall\, b.\, ggT(a,b) = ggT(x,b) \\
&\Leftarrow a = x \qquad \text{:-)}
\end{aligned}$$

Orientierung:

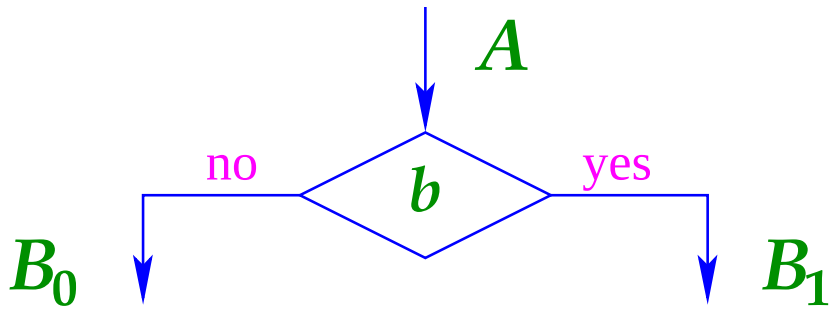


Für die Anweisungen: `a = read(); x = a;` berechnen wir:

$$\begin{aligned} \mathbf{WP}[\![\texttt{x = a;}]\!]\,(a = x) &\equiv a = a \\ &\equiv \mathbf{true} \end{aligned}$$

$$\begin{aligned} \mathbf{WP}[\![\texttt{a = read();}]\!]\,(\mathbf{true}) &\equiv \forall\, a.\ \mathbf{true} \\ &\equiv \mathbf{true} \qquad \text{:-)} \end{aligned}$$

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:

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

Beispiel:

$$B_0 \equiv x > y \wedge y > 0 \qquad\qquad B_1 \equiv x > 0 \wedge y > x$$

Sei b die Bedingung $y > x$.

Dann ist die schwächste Vorbedingung:

Beispiel:

$$B_0 \equiv x > y \wedge y > 0 \qquad\qquad B_1 \equiv x > 0 \wedge y > x$$

Sei b die Bedingung $y > x$.

Dann ist die schwächste Vorbedingung:

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

... im GGT-Beispiel:

$$
\begin{aligned}
b &\equiv y > x \\
A \wedge A &\equiv A \\
\neg b \wedge A &\equiv x \geq y \wedge ggT(a,b) = ggT(x,y) \\
b \wedge A &\equiv y > x \wedge ggT(a,b) = ggT(x,y)
\end{aligned}
$$

... im GGT-Beispiel:

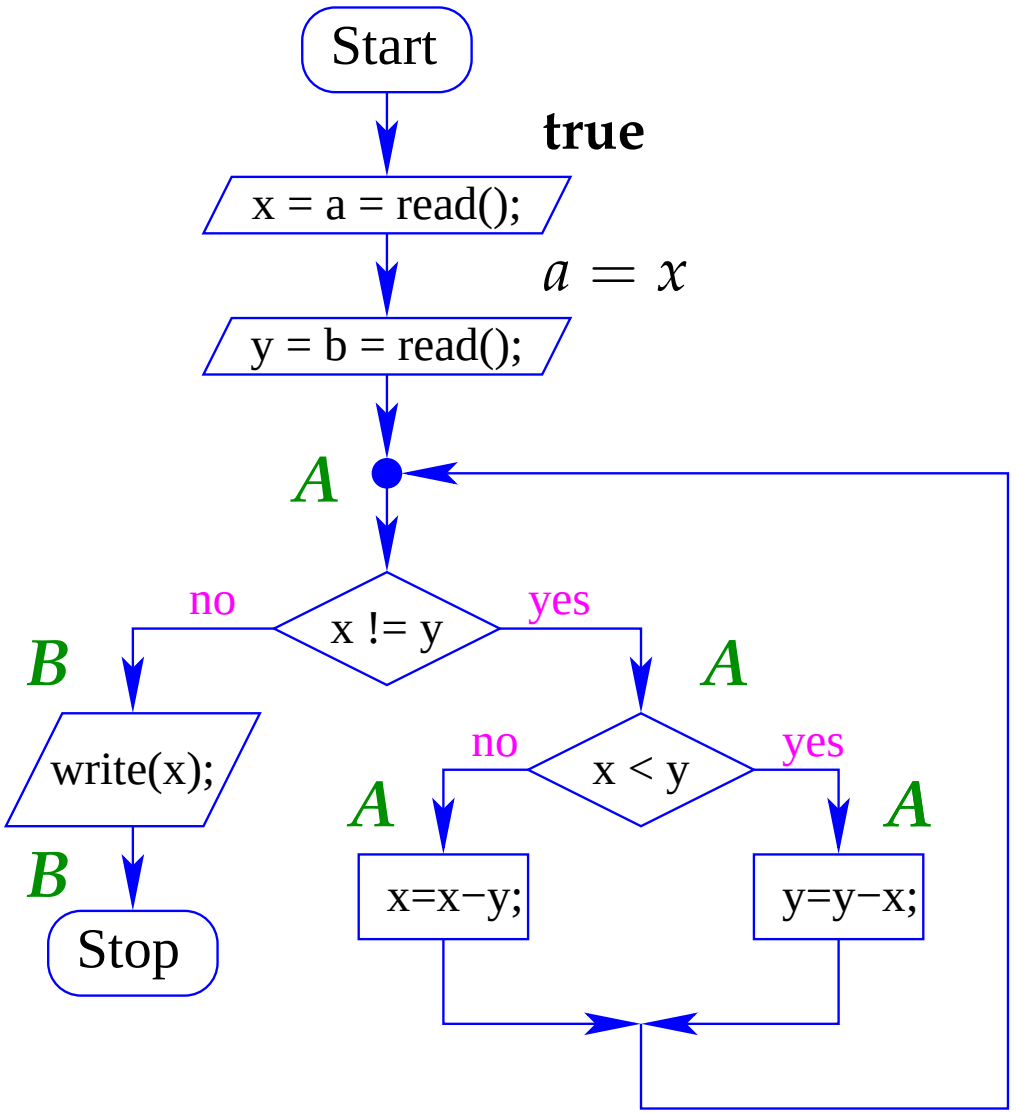
$$
\begin{aligned}
b &\equiv y > x \\
A \wedge A &\equiv A \\
\neg b \wedge A &\equiv x \geq y \wedge ggT(a,b) = ggT(x,y) \\
b \wedge A &\equiv y > x \wedge ggT(a,b) = ggT(x,y)
\end{aligned}
$$

$\Longrightarrow$ Die schwächste Vorbedingung ist:

$$ggT(a,b) = ggT(x,y)$$

... also genau A :-)

Orientierung:



Analog argumentieren wir für die Zusicherung vor der Schleife:

$$
\begin{aligned}
b &\equiv y \neq x \\
B \wedge A &\equiv B \\
\neg b \wedge B &\equiv B \\
b \wedge A &\equiv A \wedge x \neq y
\end{aligned}
$$

$\Longrightarrow$ $A \equiv (A \wedge x = y) \vee (A \wedge x \neq y) \vee B$ 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.

3.2 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:

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

Beispiel:

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

$$
\begin{array}{lcl}
\sigma & = & \{x \mapsto 5, y \mapsto 12\} \\
A & \equiv & (x > y) \\
A[5/x, 12/y] & \equiv & (5 > 12) \\
 & \equiv & \textbf{false}
\end{array}
$$

Triviale Eigenschaften:

$$\sigma \models \textbf{true} \quad \text{für jedes} \quad \sigma$$

$$\sigma \models \textbf{false} \quad \text{für kein} \quad \sigma$$

$$\sigma \models A_1 \text{ und } \sigma \models A_2 \quad \text{ist äquivalent zu} \quad \sigma \models A_1 \wedge A_2$$

$$\sigma \models A_1 \text{ oder } \sigma \models A_2 \quad \text{ist äquivalent zu} \quad \sigma \models A_1 \vee A_2$$

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):

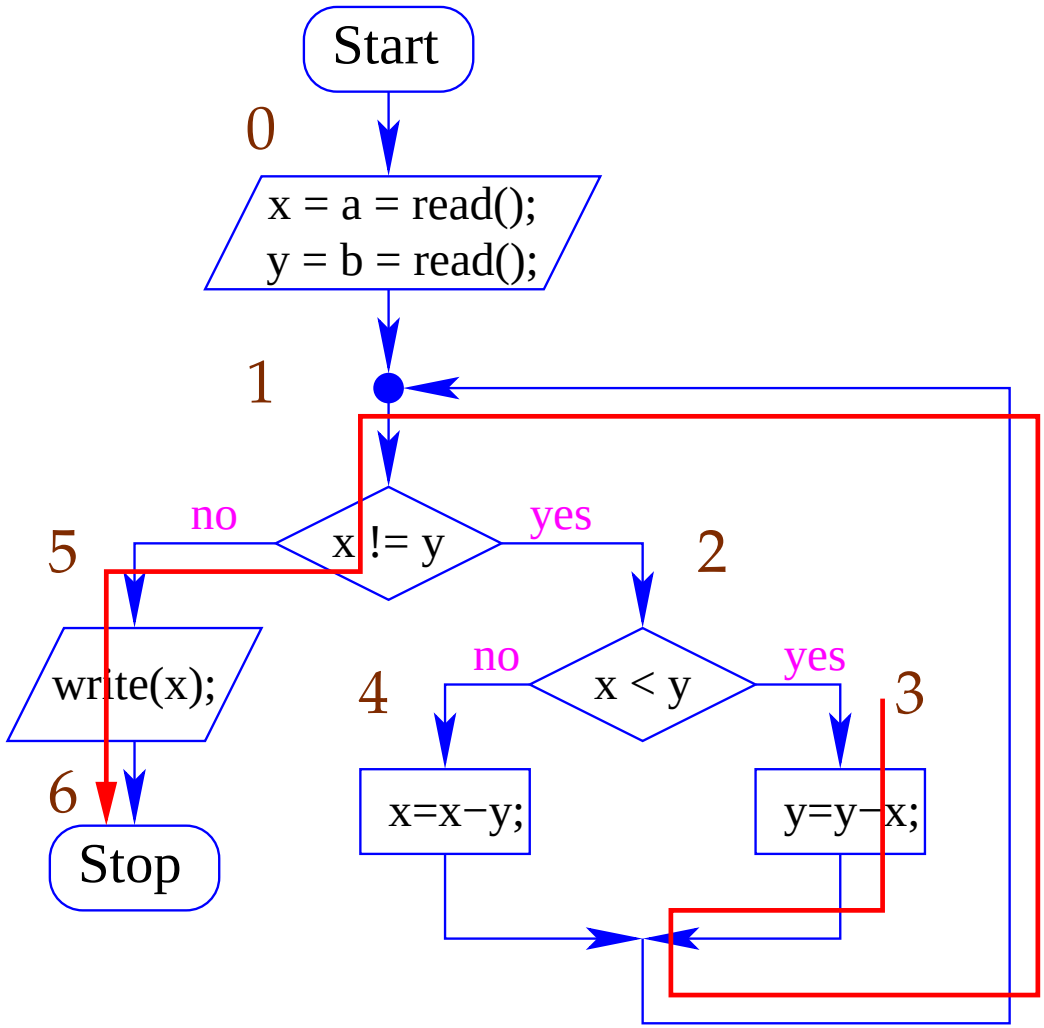
- Eine Programmausführung π durchläuft einen Pfad im Kontrollfluss-Graphen :-)
- Sie beginnt in einem Programmpunkt u_0 in einem Anfangszustand σ_0. und führt in einen Programmpunkt u_m und einen Endzustand σ_m.
- Jeder Schritt der Programm-Ausführung führt eine Aktion aus und ändert Programmpunkt und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

  $\Longrightarrow$ Wir können π als Folge darstellen:

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

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

Beispiel:


Start
0
x = a = read();
y = b = read();
1
no
x != y
yes
5
2
write(x);
no
x < y
yes
4
3
x=x−y;
y=y−x;
6
Stop

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

Dann ergibt sich die Programmausführung:

$$
\begin{array}{lll}
\pi = & (3, \{x \mapsto 6, y \mapsto 12\}) & \texttt{y = y-x;} \\
& (1, \{x \mapsto 6, y \mapsto 6\}) & \texttt{(x != y)} \\
& (5, \{x \mapsto 6, y \mapsto 6\}) & \texttt{write(x);} \\
& (6, \{x \mapsto 6, y \mapsto 6\}) &
\end{array}
$$

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

Gelte: $\sigma_0 \models A$.

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

Idee:

Induktion nach der Länge m der Programmausführung :-)