Script generated by TTT

Title: Seidl: Info2 (03.11.2017)

Date: Fri Nov 03 08:36:50 CET 2017

Duration: 89:54 min

Pages: 52

Beispiel

```
public class GGT {
   public static void main (String[] args) {
   int x, y, a, b;
   a = read(); b = read();
   x = a; y = b;
   while (x != y)
      if (x > y) x = x - y;
      else       y = y - x;

   assert(x == y);

   write(x);
   }  // Ende der Definition von main();
}     // Ende der Definition der Klasse GGT;
```


Achtung

Der Laufzeit-Test soll eine Eigenschaft des Programm-Zustands bei Erreichen eines Programm-Punkts überprüfen.

Der Test sollte keineswegs den Programm-Zustand verändern !!!

Sonst zeigt das beobachtete System ein anderes Verhalten als das unbeobachtete ???

Tipp

Um Eigenschaften komplizierterer Datenstrukturen zu überprüfen, empfiehlt es sich, getrennt Inspector-Klassen anzulegen, deren Objekte eine Datenstruktur störungsfrei besichtigen können !

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.:

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

Triviale Eigenschaften:

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

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

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

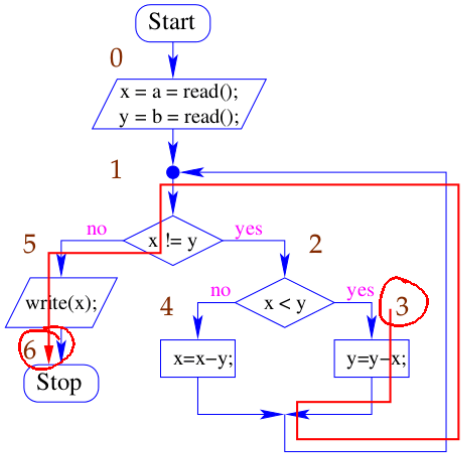
$\sigma \models A_1$ oder $\sigma \models A_2$ ist äquivalent zu $\sigma \models A_1 \vee A_2$

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

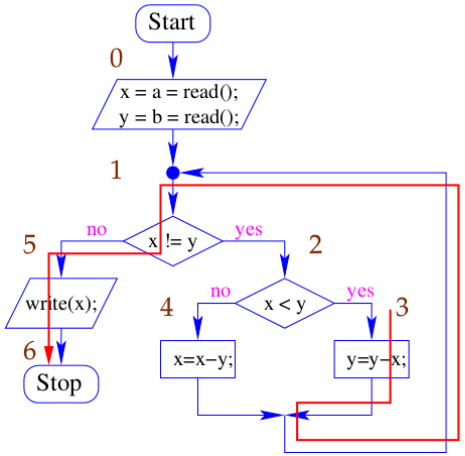
Dann ergibt sich die Programmausführung:

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

Beispiel:



Beispiel:



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

Dann ergibt sich die Programmausführung:

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

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.

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: $\sigma_m \models B$.

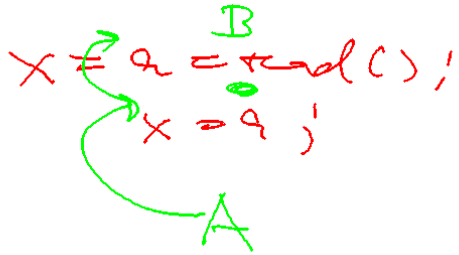
Idee:

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

    $\Longrightarrow$ Logik, automatisches Beweisen



1.3 Optimierung

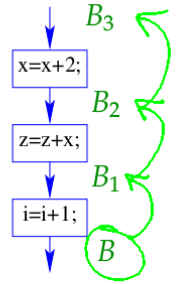
Ziel: Verringerung der benötigten Zusicherungen

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.)

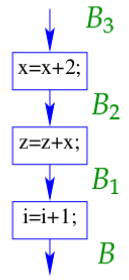
Sei $B \equiv z = i^2 \wedge x = 2i - 1$

Dann rechnen wir:

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

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:

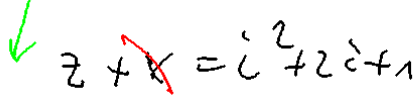
$$\begin{array}{lcll} B_1 & \equiv & \mathbf{WP}[\![\texttt{i = i+1;}]\!](B) & \equiv & z = (i+1)^2 \wedge x = 2(i+1) - 1 \\ & & & \equiv & z = (i+1)^2 \wedge x = 2i + 1 \end{array}$$

Beispiel (Fort.)

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

Dann rechnen wir:

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



Beispiel (Fort.)

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

Dann rechnen wir:

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

Idee

- Für jede Schleife wähle einen Programmpunkt aus.

  Sinnvolle Auswahlen:

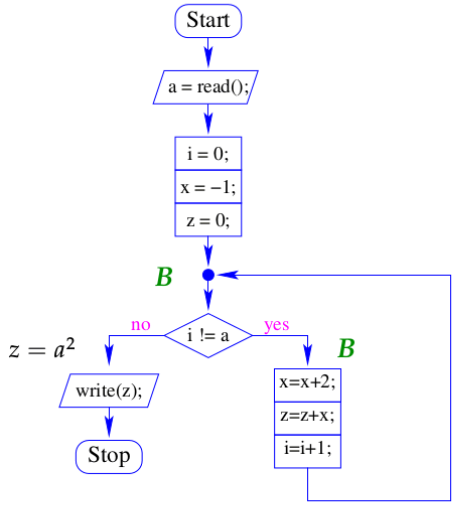
  - → Vor der Bedingung;
  - → Am Beginn des Rumpfs;
  - → Am Ende des Rumpfs ...
- Stelle für jeden gewählten Punkt eine Zusicherung bereit

  $\Longrightarrow$ 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

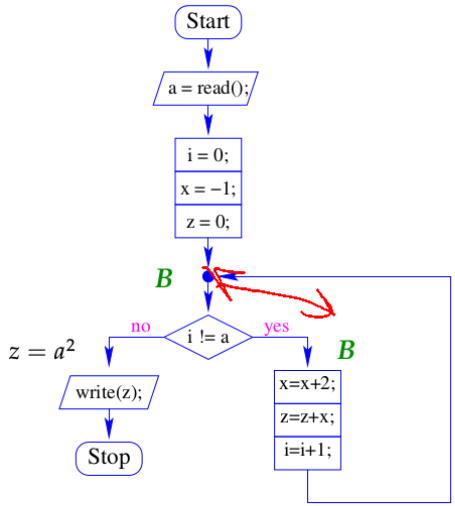


Wir überprüfen:

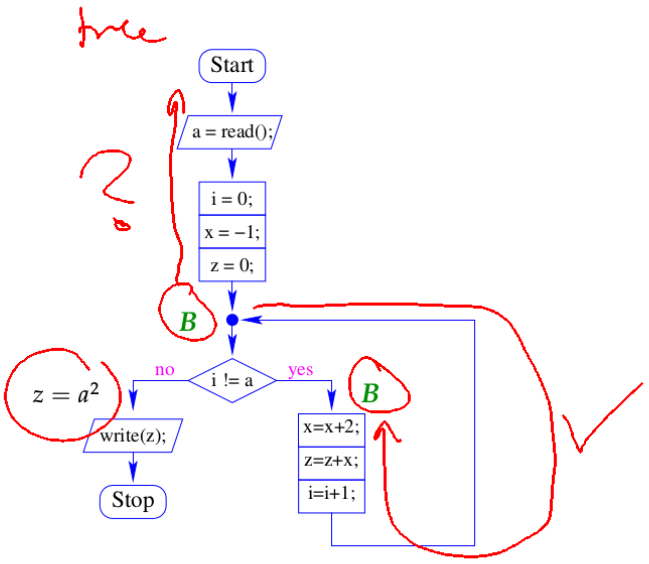
$z = i^2 \wedge x = 2i - 1$

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

Beispiel



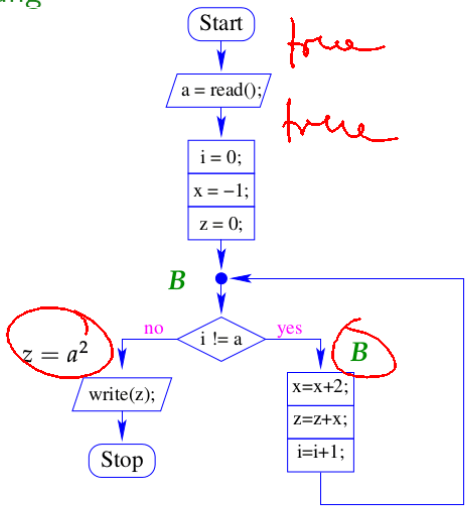
Beispiel



Wir überprüfen:

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

Orientierung



Wir überprüfen:

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

Wir überprüfen:

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

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 $a = b$ oder $a > 0$ 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 $a = b$ oder $a > 0$ 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

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 ?

Programme nur mit `for`-Schleifen der Form:

```
for (i=n; i>0; i--) {...}
// im Rumpf wird i nicht modifiziert
```

... terminieren ebenfalls immer!

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!

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);
}
```

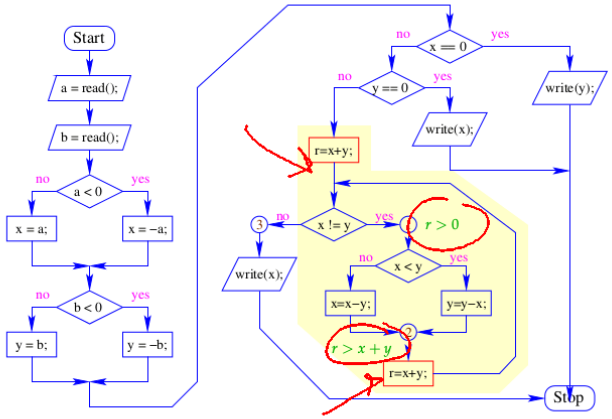

Wir wählen: $r = x + y$

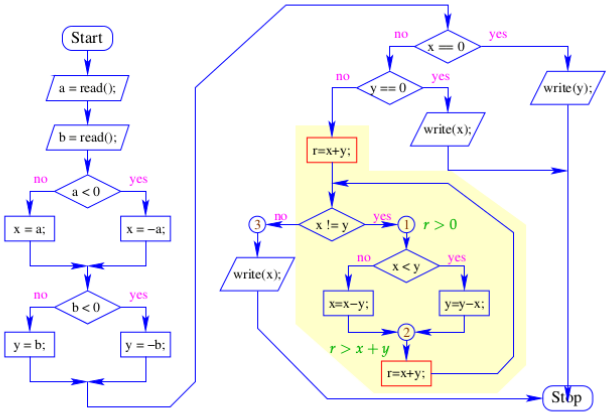
Transformation

```
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);
}
```

r > 0

r > x+y





An den Programmpunkten 1, 2 und 3 machen wir die Zusicherungen:

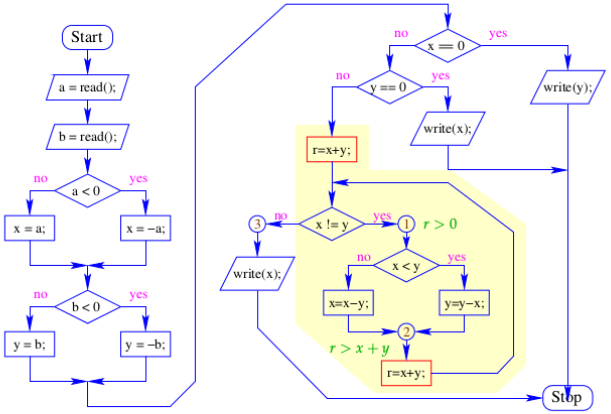
(1) $A \quad \equiv \quad x \neq y \wedge x > 0 \wedge y > 0 \wedge r = x + y$

(2) $B \quad \equiv \quad x > 0 \wedge y > 0 \wedge r > x + y$

(3) **true**

Dann gilt:

$$A \Rightarrow r > 0 \qquad \text{und} \qquad B \Rightarrow r > x + y$$



An den Programmpunkten 1, 2 und 3 machen wir die Zusicherungen:

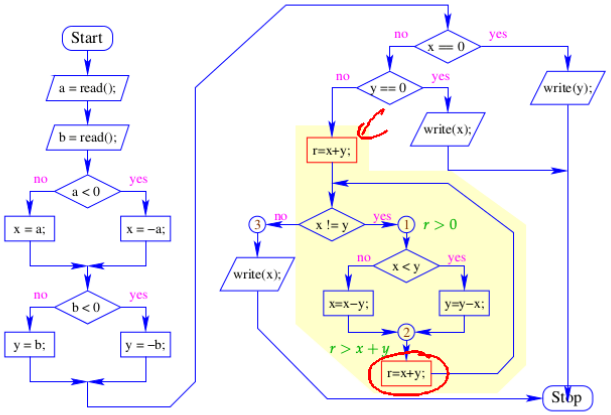
(1) $A \equiv x \neq y \wedge x > 0 \wedge y > 0 \wedge r = x + y$

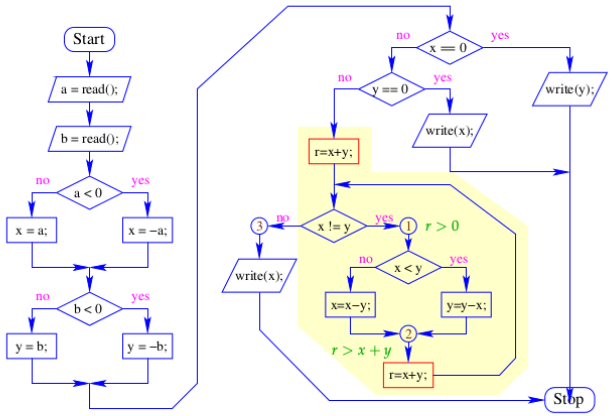
(2) $B \equiv x > 0 \wedge y > 0 \wedge r > x + y$

(3) **true**

Dann gilt:

$$A \Rightarrow r > 0 \quad \text{und} \quad B \Rightarrow r > x + y$$





An den Programmpunkten 1, 2 und 3 machen wir die Zusicherungen:

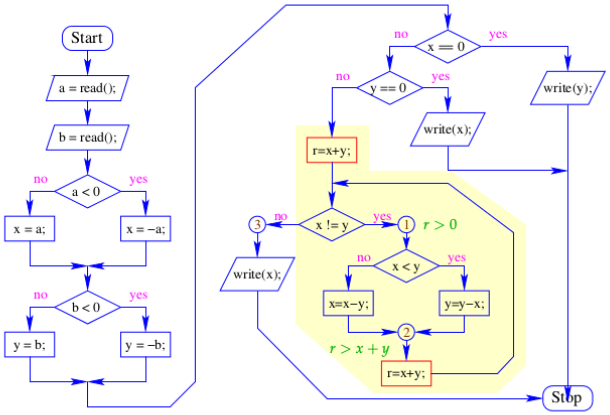
(1) $A \equiv x \neq y \wedge x > 0 \wedge y > 0 \wedge r = x + y$

(2) $B \equiv x > 0 \wedge y > 0 \wedge r > x + y$

(3) **true**

Dann gilt:

$$A \Rightarrow r > 0 \quad \text{und} \quad B \Rightarrow r > x + y$$


Start
a = read();
b = read();
a < 0
no
yes
x = a;
x = −a;
b < 0
y = b;
y = −b;

x == 0
y == 0
write(y);
write(x);
r=x+y;
x != y
write(x);
r > 0
x < y
x=x−y;
y=y−x;
r > x + y
r=x+y;
Stop