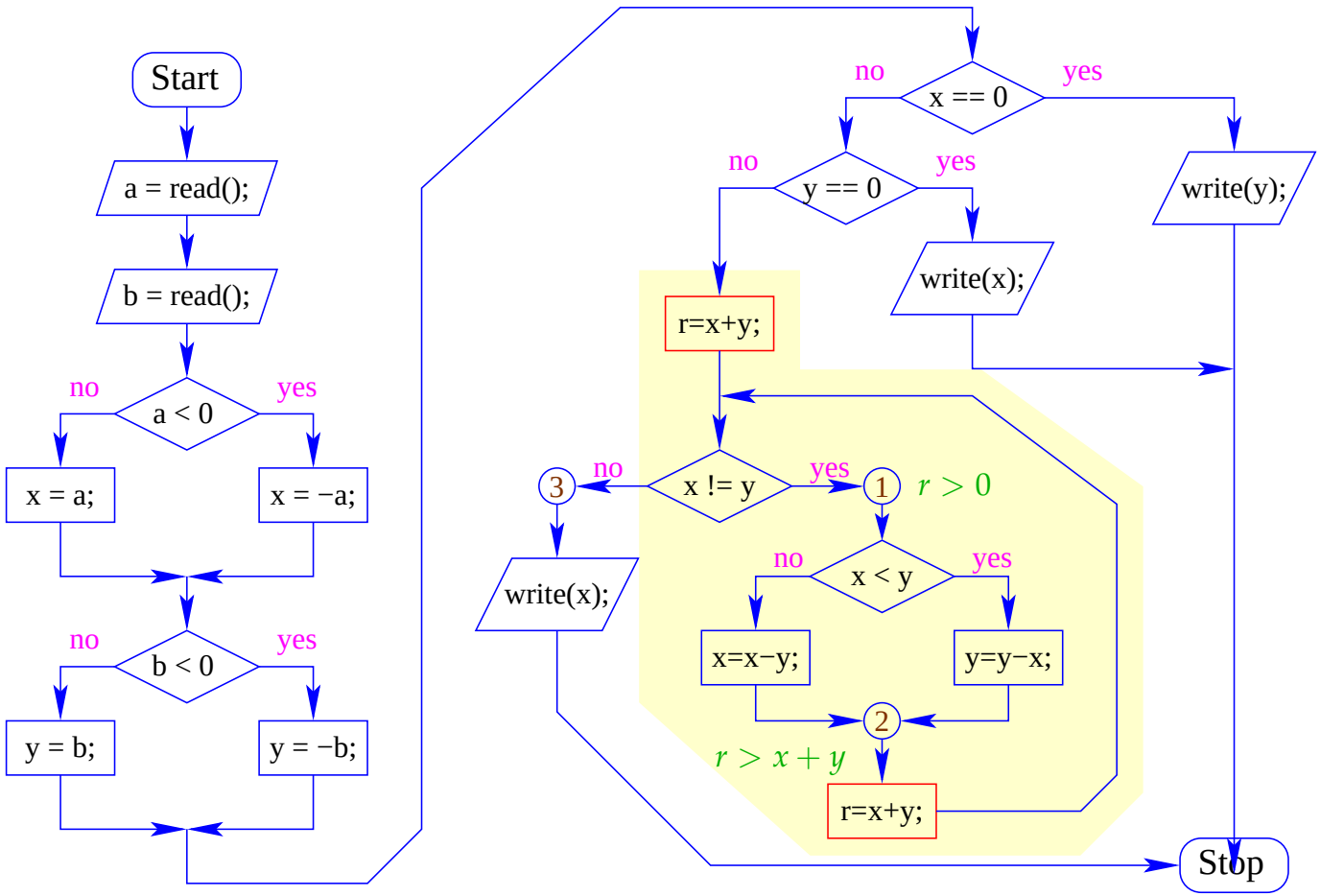

Start
a = read();
b = read();
a < 0
no
yes
x = a;
x = −a;
b < 0
no
yes
y = b;
y = −b;
x == 0
no
yes
write(y);
y == 0
no
yes
write(x);
r=x+y;
x != y
no
yes
3
1
$r > 0$
write(x);
x < y
no
yes
x=x−y;
y=y−x;
2
$r > x + y$
r=x+y;
Stop

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

$$
\begin{array}{lll}
(1) & A & \equiv \quad x \neq y \wedge x > 0 \wedge y > 0 \wedge r = x + y \\
(2) & B & \equiv \quad x > 0 \wedge y > 0 \wedge r > x + y \\
(3) & \textbf{true} &
\end{array}
$$

Dann gilt:

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

## Wir überprüfen:

$$\begin{aligned}
\mathbf{WP}[\![\texttt{x != y}]\!](\mathbf{true}, A) &\equiv x = y \vee A \\
&\Leftarrow x > 0 \wedge y > 0 \wedge r = x + y \\
&\equiv C
\end{aligned}$$

## Wir überprüfen:

$$
\begin{array}{lcl}
\mathbf{WP}[\![\texttt{x != y}]\!](\mathbf{true}, A) & \equiv & x = y \ \vee \ A \\
 & \Leftarrow & x > 0 \wedge y > 0 \wedge r = x + y \\
 & \equiv & C \\
\mathbf{WP}[\![\texttt{r = x+y;}]\!](C) & \equiv & x > 0 \wedge y > 0 \\
 & \Leftarrow & B
\end{array}
$$

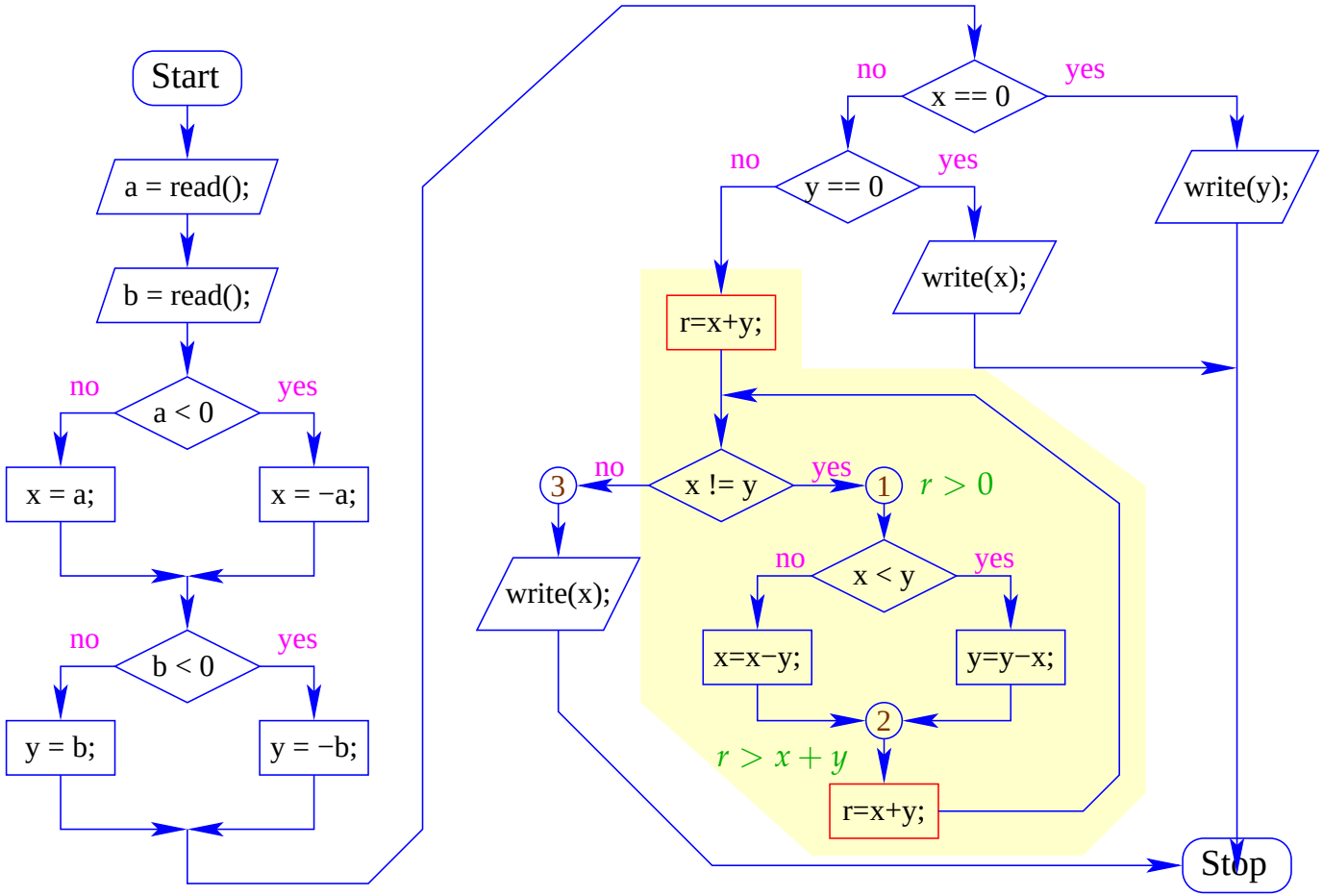
## Wir überprüfen:

$$
\begin{array}{lcl}
\mathbf{WP}[\![\texttt{x != y}]\!](\mathbf{true}, A) & \equiv & x = y \ \vee \ A \\
 & \Leftarrow & x > 0 \wedge y > 0 \wedge r = x + y \\
 & \equiv & C \\
\mathbf{WP}[\![\texttt{r = x+y;}]\!](C) & \equiv & x > 0 \wedge y > 0 \\
 & \Leftarrow & B \\
\mathbf{WP}[\![\texttt{x = x-y;}]\!](B) & \equiv & x > y \wedge y > 0 \wedge r > x \\
\mathbf{WP}[\![\texttt{y = y-x;}]\!](B) & \equiv & x > 0 \wedge y > x \wedge r > y
\end{array}
$$

## Wir überprüfen:

$$
\begin{array}{lcl}
\mathbf{WP}[\![\texttt{x != y}]\!](\mathbf{true}, A) & \equiv & x = y \ \vee \ A \\
& \Leftarrow & x > 0 \wedge y > 0 \wedge r = x + y \\
& \equiv & C \\
\mathbf{WP}[\![\texttt{r = x+y;}]\!](C) & \equiv & x > 0 \wedge y > 0 \\
& \Leftarrow & B \\
\mathbf{WP}[\![\texttt{x = x-y;}]\!](B) & \equiv & x > y \wedge y > 0 \wedge r > x \\
\mathbf{WP}[\![\texttt{y = y-x;}]\!](B) & \equiv & x > 0 \wedge y > x \wedge r > y \\
\mathbf{WP}[\![\texttt{y > x}]\!](\ldots, \ldots) & \equiv & (x > y \wedge y > 0 \wedge r > x) \ \vee \\
& & (x > 0 \wedge y > x \wedge r > y) \\
& \Leftarrow & x \neq y \wedge x > 0 \wedge y > 0 \wedge r = x + y \\
& \equiv & A
\end{array}
$$

# Orientierung:



Weitere Propagation von $C$ durch den Kontrollfluss-Graphen komplettiert die lokal konsistente Annotation mit Zusicherungen :-)

Weitere Propagation von $C$ durch den Kontrollfluss-Graphen komplettiert die lokal konsistente Annotation mit Zusicherungen :-)

## Wir schließen:

- An den Programmpunkten 1 und 2 gelten die Zusicherungen $r > 0$ bzw. $r > x + y$.
- In jeder Iteration wird $r$ kleiner, bleibt aber stets positiv.
- Folglich wird die Schleife nur endlich oft durchlaufen

  $\Longrightarrow$ das Programm terminiert :-))

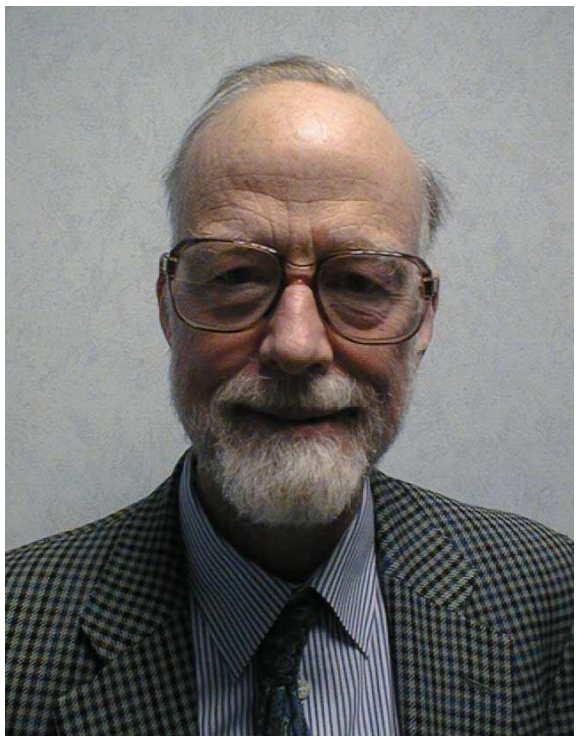
# Allgemeines Vorgehen:

- Für jede vorkommende Schleife `while (b) s` erfinden wir eine neue Variable `r`.
- Dann transformieren wir die Schleife in:

```
r = e0;
while (b) {
   assert(r>0);
   s
   assert(r > e1);
   r = e1;
}
```

für geeignete Ausdrücke `e0`, `e1` :-)

## 3.5 Der Hoare-Kalkül



Tony Hoare, Microsoft Research, Cambridge

# Idee (1):

- Organisiere den Korrektheitsbeweis entsprechend der Struktur des Programms !
- Modularisiere ihn so, dass Beweise für Teilprogramme zu einem Beweis für das gesamte Programm zusammen gesetzt werden können !

# Idee (2):

Betrachte Aussagen der Form:

$$\{A\} \quad p \quad \{B\}$$

... das heißt:

Gilt vor der Ausführung des Programmstücks $p$ Eigenschaft $A$
und terminiert die Programm-Ausführung, dann
gilt nach der Ausführung von $p$ Eigenschaft $B$.

# Idee (2):

Betrachte Aussagen der Form:

$$\{A\} \quad p \quad \{B\}$$

... das heißt:

Gilt vor der Ausführung des Programmstücks $p$ Eigenschaft $A$
und terminiert die Programm-Ausführung, dann
gilt nach der Ausführung von $p$ Eigenschaft $B$.

$A$ : Vorbedingung

$B$ : Nachbedingung

# Beispiele:

$\{x > y\}$ `z = x-y;` $\{z > 0\}$

# Beispiele:

$$\{x > y\} \quad \texttt{z = x-y;} \quad \{z > 0\}$$

$$\{\textbf{true}\} \quad \texttt{if (x<0) x=-x;} \quad \{x \geq 0\}$$

# Beispiele:

$$\{x > y\} \quad \texttt{z = x-y;} \quad \{z > 0\}$$

$$\{\textbf{true}\} \quad \texttt{if (x<0) x=-x;} \quad \{x \geq 0\}$$

$$\{x > 7\} \quad \texttt{while (x!=0) x=x-1;} \quad \{x = 0\}$$

# Beispiele:

$$\{x > y\} \quad \texttt{z = x-y;} \quad \{z > 0\}$$

$$\{\textbf{true}\} \quad \texttt{if (x<0) x=-x;} \quad \{x \geq 0\}$$

$$\{x > 7\} \quad \texttt{while (x!=0) x=x-1;} \quad \{x = 0\}$$

$$\{\textbf{true}\} \quad \texttt{while (true);} \quad \{\textbf{false}\}$$

# Ableitung korrekter Aussagen:

- Bestimme eine Menge von Grundaussagen oder Axiome, die gültige Tripel für einfache Anweisungen beschreiben.
- Gib Regeln an, wie aus bereits abgeleiteten gültigen Tripeln weitere gültige Tripel hergeleitet werden können ...

# Axiome:

| | | | |
|---|---|---|---|
| leere Anweisung: | $\{B\}$ | `;` | $\{B\}$ |
| Zuweisung: | $\{B[e/x]\}$ | `x = e;` | $\{B\}$ |
| Eingabe: | $\{\forall\, x.\, B\}$ | `x = read();` | $\{B\}$ |
| Ausgabe: | $\{B\}$ | `write(e);` | $\{B\}$ |

# Regeln:

Komposition:
$$\frac{\{A\}\ \texttt{s}\ \{B\} \qquad \{B\}\ \texttt{ss}\ \{C\}}{\{A\}\ \texttt{s ss}\ \{C\}}$$

- Oberhalb des Strichs stehen die Voraussetzungen für die Regel-Anwendung.
- Unterhalb des Strichs steht die Schlussfolgerung :-)

$\Longrightarrow$

Den Beweis eines Tripels für das zusammengesetzte Programm `s ss` setzen wir aus Beweisen für Aussagen über die Programmbestandteile `s` und `ss` zusammen :-)

# Beispiel:

Für: `x=x+2; z=z+x; i=i+1;` konstruieren wir:

$$\dfrac{\dfrac{\{z = i^2 \wedge x = 2i-1\}\ \texttt{x=x+2;}\ \{z = i^2 \wedge x = 2i+1\} \quad \dfrac{\{z = i^2 \wedge x = 2i+1\}\ \texttt{z=z+x;}\ \{z = (i+1)^2 \wedge x = 2i+1\} \quad \{z = (i+1)^2 \wedge x = 2i+1\}\ \texttt{i=i+1;}\ \{z = i^2 \wedge x = 2i-1\}}{\{z = i^2 \wedge x = 2i+1\}\ \texttt{z=z+x; i=i+1;}\ \{z = i^2 \wedge x = 2i-1\}}}{\{z = i^2 \wedge x = 2i-1\}\ \texttt{x=x+2; z=z+x; i=i+1;}\ \{z = i^2 \wedge x = 2i-1\}}$$

# Regeln (Forts.):

Bedingung:

$$\frac{\{\neg b \wedge A\}\ \texttt{ss0}\ \{B\} \qquad \{b \wedge A\}\ \texttt{ss1}\ \{B\}}{\{A\}\ \texttt{if (b) \{ss0\} else \{ss1\}}\ \{B\}}$$

- Diese Regel entspricht unserer bisherigen Behandlung bedingter Verzweigungen ...
- Falls $\{A_0\}\ \texttt{ss0}\ \{B\}$ und $\{A_1\}\ \texttt{ss1}\ \{B\}$ gelten, können wir $A$ auch wählen als

$$A \quad \equiv \quad (b \Rightarrow A_0) \wedge (\neg b \Rightarrow A_0) \quad \text{:-)}$$

# Beispiel:

Wir wollen

$$\{A\}\ \texttt{if (y>x) y=y-x; else x=x-y;}\ \{B\}$$

beweisen für:

$$\begin{aligned} A &\equiv x > 0 \wedge y > 0 \wedge x \neq y \\ B &\equiv x > 0 \wedge y > 0 \end{aligned}$$

Dazu konstruieren wir:

$$\dfrac{\dfrac{\{x > 0 \wedge y > x\}\quad \text{y=y−x;}\quad \{B\}}{\{A \wedge y > x\}\quad \text{y=y−x;}\quad \{B\}} \qquad \dfrac{\{x > y \wedge y > 0\}\quad \text{x=x−y;}\quad \{B\}}{\{A \wedge y \leq x\}\quad \text{x=x−y;}\quad \{B\}}}{\{A\}\quad \text{if (y>x) y=y−x;else x=x−y;}\quad \{B\}}$$

# Regeln (Forts.):

Schleife:
$$\frac{\{b \wedge I\}\ \texttt{ss}\ \{I\}}{\{I\}\ \texttt{while (b) \{ss\}}\ \{\neg b \wedge I\}}$$

- Die Zusicherung $I$ entspricht unserer Schleifen-Invariante :-)
- Bei Betreten des Rumpfs wissen wir zusätzlich, dass die Bedingung $b$ gilt :-)
- Bei Verlassen der Schleife wissen wir zusätzlich, dass die Bedingung $b$ nicht mehr gilt :-))

# Regeln (Forts.):

- Gelegentlich passen die Vor- bzw. Nachbedingungen der bereits als korrekt bewiesenen Tripel nicht genau zusammen ...
- Dann kann man die Vorbedingung verstärken oder die Nachbedingung abschwächen:

Abschwächung:

$$\frac{A \Rightarrow A_1 \qquad \{A_1\}\ \texttt{ss}\ \{B_1\} \qquad B_1 \Rightarrow B}{\{A\}\ \texttt{ss}\ \{B\}}$$

## 3.6 Behandlung von Prozeduren

Modulare Verifikation können wir benutzen, um die Korrektheit auch von Programmen mit Funktionen nachzuweisen :-)

# Vereinfachung:

Wir betrachten nur

- Prozeduren, d.h. statische Methoden ohne Rückgabewerte;
- nur globale Variablen, d.h. alle Variablen sind sind ebenfalls `static`.

  // werden wir später verallgemeinern :-)

# Beispiel:

```
int a,  b, x, y;

void main () {
   a = read();
   b = read();
   mm();
   write (x-y);
}
```

```
void mm() {
   if (a>b) {
      x = a;
      y = b;
   } else {
      y = a;
      x = b;
   }
}
```

# Kommentar:

- Der Einfachkeit halber haben wir alle Vorkommen von `static` gestrichen :-)
- Die Prozedur-Definitionen sind nicht rekursiv.
- Das Programm liest zwei Zahlen ein.
- Die Prozedur `minmax` speichert die größere in `x`, die kleinere in `y` ab.
- Die Differenz von `x` und `y` wird ausgegeben.
- Wir wollen zeigen, dass gilt:

$$\{a \geq b\} \ \texttt{mm();} \ \{a = x\}$$

# Vorgehen:

- Für jede Prozedur `f()` stellen wir ein Tripel bereit:

$$\{A\}\ \texttt{f();}\ \{B\}$$

- Unter dieser globalen Hypothese $H$ verifizieren wir, dass sich für jede Prozedurdefinition `void f() { ss }` zeigen lässt:

$$\{A\}\ \texttt{ss}\ \{B\}$$

- Wann immer im Programm ein Prozeduraufruf vorkommt, benutzen wir dabei die Tripel aus $H$ ...