http://scam-trap.com/i/stophacking1.gif

Verifikation von Programmen

OOPM, Ralf Lämmel

Was tun?
(Mit den Spezifikationen)

- Dokumentation
- Überprüfen der Bedingungen zur Laufzeit
- **Verifizieren der Bedingungen *vor Laufzeit***
- Ableitung einer korrekten Implementation aus Spezifikation
- Verwendung der Spezifikation zur Testdatengenerierung

Entwicklung korrekter Programme

OOPM

* 1. Ansatz
  - Aufstellung einer Spezifikation (Problembeschreibung).
  - Erstellung einer Implementation (Problemlösung).
  - **Beweis** über Einhaltung der Spezifikation durch Programm.
* 2. Ansatz
  - Aufstellung einer Spezifikation.
  - Schrittweise Ableitung einer (effizienten) Implementation.
    - **Korrektheit folgt aus Konstruktion.**

¬ OOPM

Programme mit Spezifikation = Tripel

{ *P* } *S* { *Q* }

- *S* ist eine Anweisung.
- *P* und *Q* sind ***Zusicherungen*** (Bedingungen)
- *P* heißt Vorbedingung.
- *Q* heißt Nachbedingung.

Formale Bedeutung von Tripeln
{ *P* } *S* { *Q* }

Zwei Optionen:

- *Totale Korrektheit*: Jede Ausführung von *S* ausgehend von einem Zustand, in dem *P* gilt, wird normal zum Halt kommen in einem Zustand, in dem *Q* gilt.

- ***Partielle Korrektheit*:** Jede Ausführung von *S* ausgehend von einem Zustand, in dem *P* gilt, wird entweder nicht normal zum Halt kommen (z.B. nicht terminieren) oder normal zum Halt kommen in einem Zustand, in dem *Q* gilt.

Spezifikation für die Division mit Rest

```
{ x >= 0 && y > 0 }
q = 0;
r = x;
while (r >= y) {
   r = r - y;
   q = q + 1;
}
{ x == y * q + r && r < y && r >= 0 && q >= 0 }
```

Wie können wir einen ***Beweis*** angehen?

Ein Tripel wird zerlegt in viele Teilbeweise mit kleineren Tripeln.

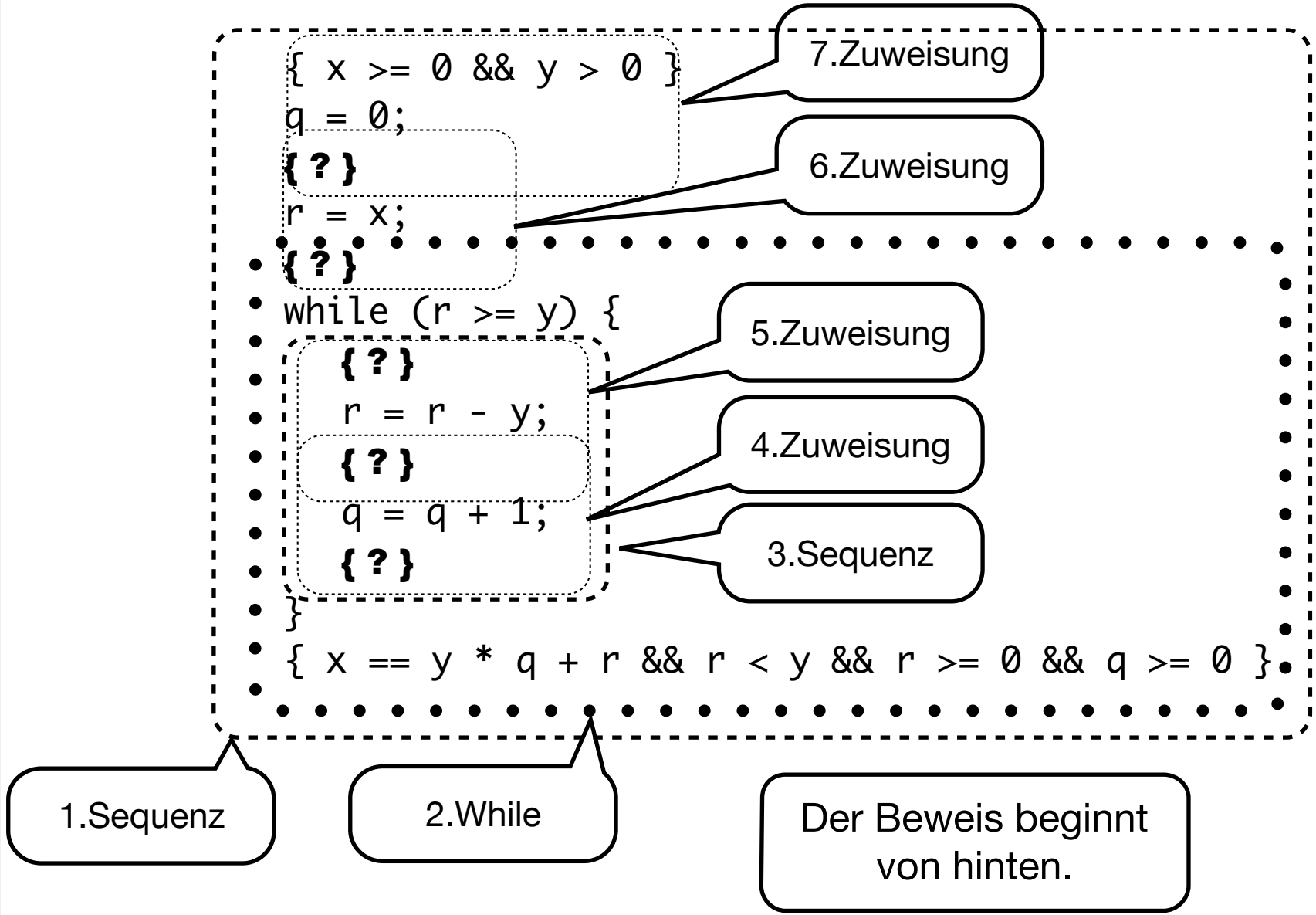
```
{ x >= 0 && y > 0 }
q = 0;
{?}
r = x;
{?}
while (r >= y) {
   {?}
   r = r - y;
   {?}
   q = q + 1;
   {?}
}
{ x == y * q + r && r < y && r >= 0 && q >= 0 }
```

Betrachtung der Optionen

für *S* in { *P* } *S* { *Q* }

- Zuweisung
- Anweisungssequenz
- While-Schleife

Zusammenfassung der Beweisschritte für “Division mit Rest”



QED

Beweisschritte für Sprachkonstruktionen

- Zuweisung
- Anweisungssequenz
- While-Schleife

Beweis einer Zuweisung
{ *P* } *v* = *t*; { *Q* }

1. OPTION

- **Annahme**
  - **Nur *Q* ist gegeben.**
- Vorgehensweise
  - Ermittle *P* aus *Q*.
    - Ersetze *v* in *Q* durch *t*.
- Beispiel
  - { *y*==0 } *x* = *y*; { *x*==0 }

Instanzierung

v	*x*
t	*y*
P	*y*==0
Q	*x*==0

Wir lesen also die Vorbedingung *P* aus dem Programm ab.

Beweis einer Zuweisung
{ *P* } *v* = *t*; { *Q* }

2. OPTION

- **Annahme**
  - ***P* und *Q* sind gegeben.**
- Vorgehensweise
  - Ermittle *P'* aus *Q*.
    - Ersetze *v* in *Q* durch *t*.
  - Überprüfe dass $P \Rightarrow P'$
- Beispiel
  - { *y*==0 } *x* = *y*; { *x*>=0 }

Instanzierung

v	*x*
t	*y*
P	*y*==0
P'	*y*>=0
Q	*x*>=0

P' is notwendig für *Q*.
P ist "stärker als *P'*.
P ist hinreichend für *Q*.

Beweis einer Zuweisung
$\{ P \}\, v = t; \{ Q \}$

3. OPTION

- Annahme
  - Nur *P* ist gegeben.
- Vorgehensweise
  - Keine allgemeine, systematische Vorgehensweise

Beweis einer Anweisungssequenz
$\{ P \}\ A_1; A_2\ \{ Q \}$

- Annahme
  - Nur Q ist gegeben.
- Vorgehensweise
  - Zerlege das Problem.
    - $\{ P \}\ A_1\ \{ R \}$
    - $\{ R \}\ A_2\ \{ Q \}$
  - Führe die Teilbeweise.
- Beispiel
  - $\{ x>0 \}\ x{+}{+};\ x{+}{+};\ \{ x>2 \}$

Instanzierung

A_1	$x{+}{+};$
A_2	$x{+}{+};$
P	$x>0$
R	x>1
Q	$x>2$

Zwischenrechnung
nächste Seite

Zwischenrechnung

- Zu zeigen:
  - $\{ x > 1 \}\ x{+}{+};\ \{ x > 2 \}$
- Beweis:
  - Zuckerexpansion für “$x{+}{+}$”
    - $\{ ? \}\ x = x + 1;\ \{ x > 2 \}$
  - Ablesen der Vorbedingung
    - $\{ x + 1 > 2 \}\ x = x + 1;\ \{ x > 2 \}$
  - Vereinfachung von “$x + 1 > 2$”
    - $\{ x > 1 \}\ x{+}{+};\ \{ x > 2 \}$

wzbw.

Beweis einer While-Schleife
{ *P* } while (*B*) *S* { *Q* }

?

Die wesentliche Idee ist, dass eine Schleife eine Invariante *I* bewahrt. Sie gilt “überall”.

// Vor der Schleife

I gilt hier.

while (*B*)

// Am Anfang des Schleifenkörpers

I gilt hier. *B* gilt **auch**.

S

// Am Ende des Schleifenkörpers

I gilt hier.

// Nach der Schleife

I gilt hier. *B* gilt **nicht**.

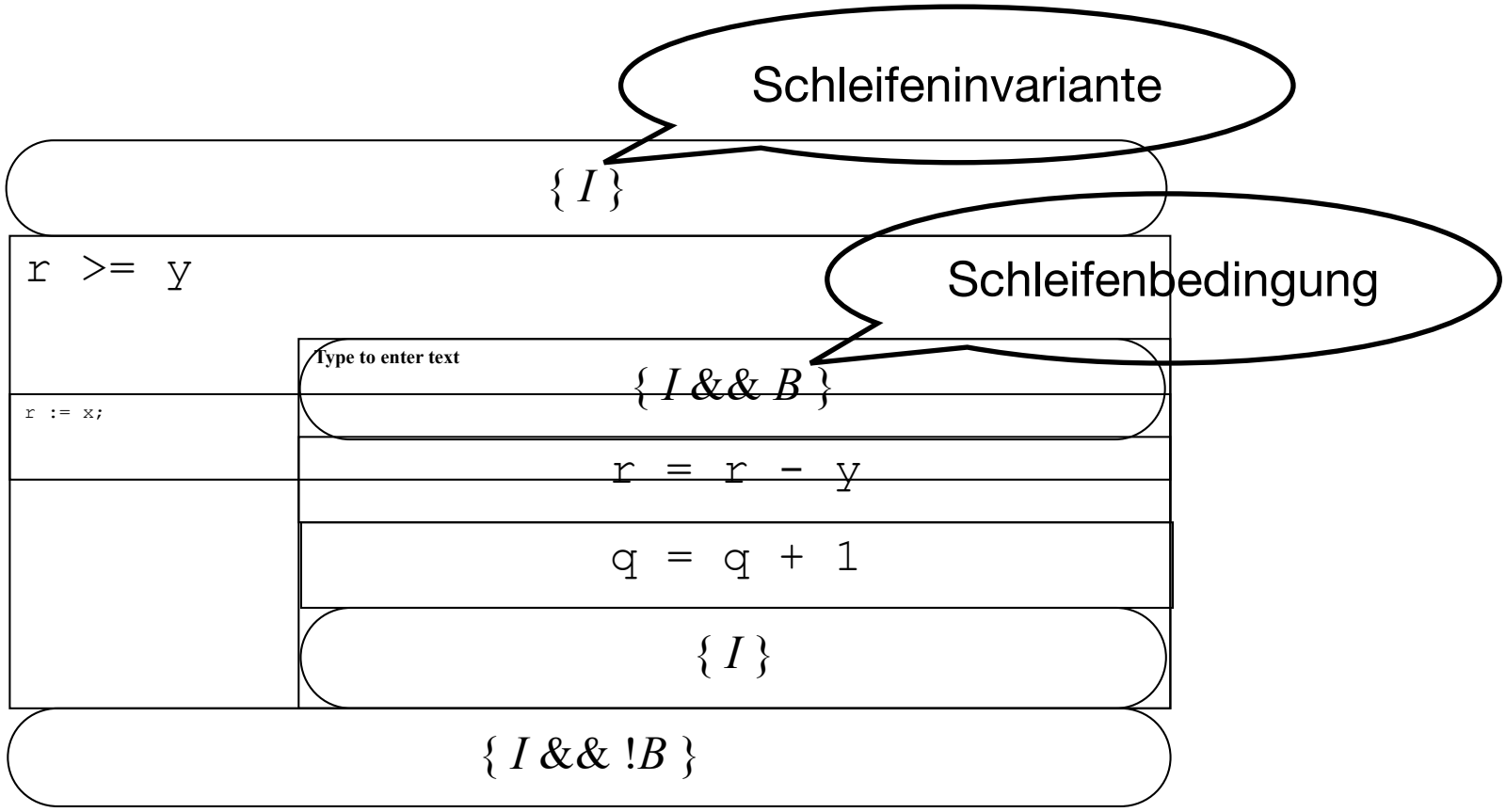
Beweis einer Schleife
{ *P* } while (*B*) *S* { *Q* }

- Annahme
  - Nur *Q* ist gegeben.
- Vorgehensweise
  - Identifiziere die Invariante *I*.
  - Zerlege das Problem.
    - { *I* && *B* } *S* { *I* }
    - *I* && !*B* ⇒ *Q*
  - Führe die Teilbeweise.
- Beispiel
  - { *x*>=*y* } while (*x*>y) *x*--; { *x*==*y* }

Instanzierung

B	*x*>*y*
S	*x*--
P	*x*>=*y*
Q	*x*==*y*
I	*x*>=*y*

Anwendung auf “Division mit Rest”



Wie ist *I* hier beschaffen?

Ermittlung der Invariante & Co

- *I* $\equiv x == q * y + r \;\&\&\; r >= 0 \;\&\&\; q >= 0$

- *I* && !*B* $\equiv x == q * y + r \;\&\&\; r < y \;\&\&\; r >= 0 \;\&\&\; q >= 0$

Letzteres ist die ***Nachbedingung*** des Programms.

$$\frac{\{ P \}\ S_1\ \{ Q \},\ \{ Q \}\ S_2\ \{ R \}}{\{ P \}\ S_1; S_2\ \{ R \}}$$

$$\{ P[x/t] \}\ x = t\ \{ P \}$$

Mechanisierung der Beweisschritte auf der Basis formaler Regeln

$$\frac{\{ I \,\&\&\, B \}\ S\ \{ I \}}{\{ I \}\ \underline{\text{while}}\ (B)\ S\ \{ I \,\&\&\, !B \}}$$

$$\frac{P \Rightarrow Q,\ \{ Q \}\ S\ \{ R \}}{\{ P \}\ S\ \{ R \}}$$

Hoare-Regel für die While-Schleife

Voraussetzung

$$\frac{\{\ I\ \&\&\ B\ \}\ S\ \{\ I\ \}}{\{\ I\ \}\ \underline{\textbf{while}}\ (B)\ S\ \{\ I\ \&\&\ !B\ \}}$$

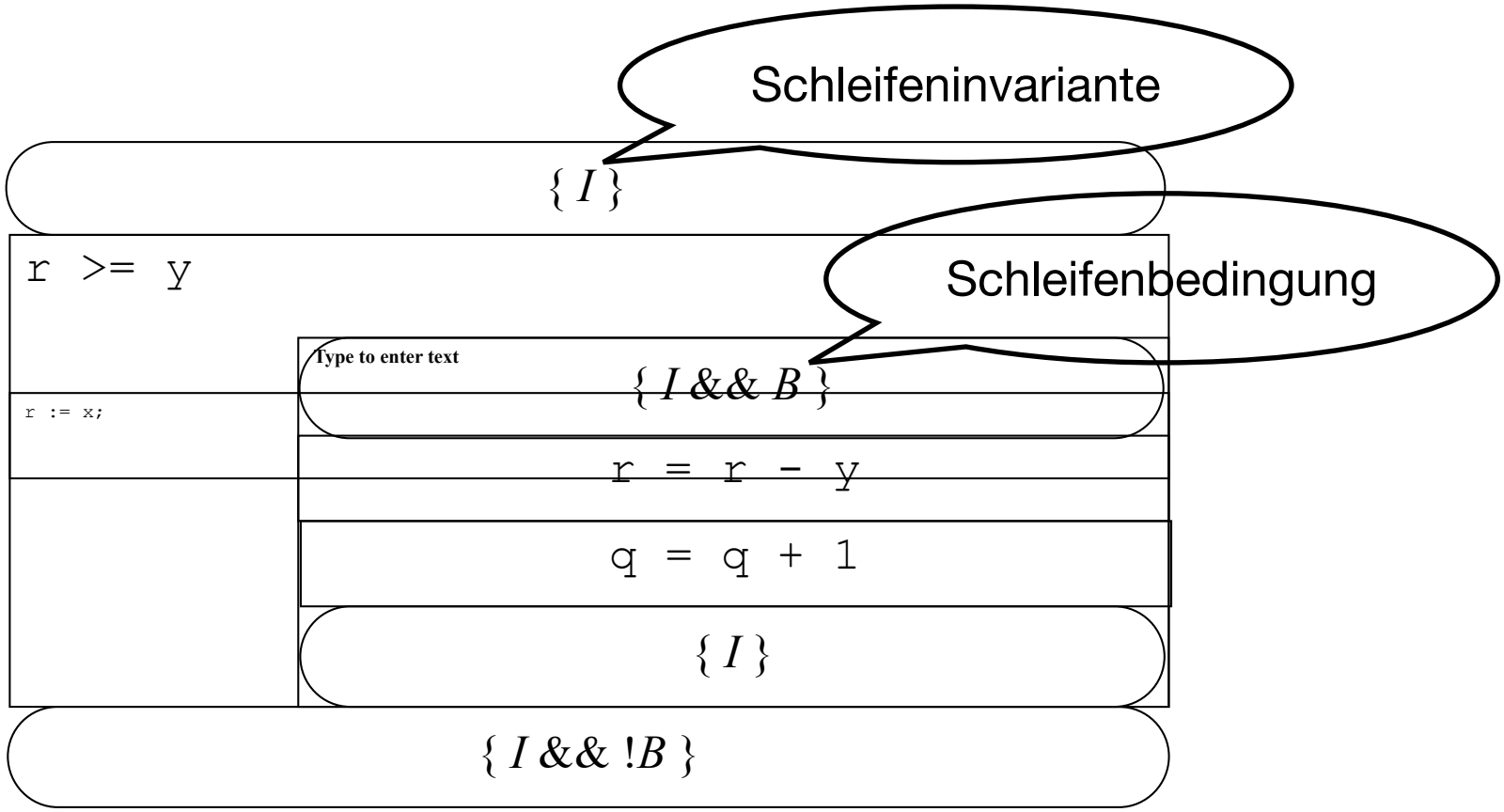
Schlussfolgerung

B ist die Schleifenbedingung.
S ist der Schleifenkörper.
I ist die Schleifeninvariante.
I muss (fast) überall gelten:
- vor und nach der Schleife.
- vor und nach jedem Schleifendurchlauf.

Einsetzen in die Schleife

{ I && B } S { I }

{ I } <u>while</u> B S { I && !B }



{ I && B } S { I }

{ I } <u>while</u> B S { I && !B }

Vorkommende Bedingungen

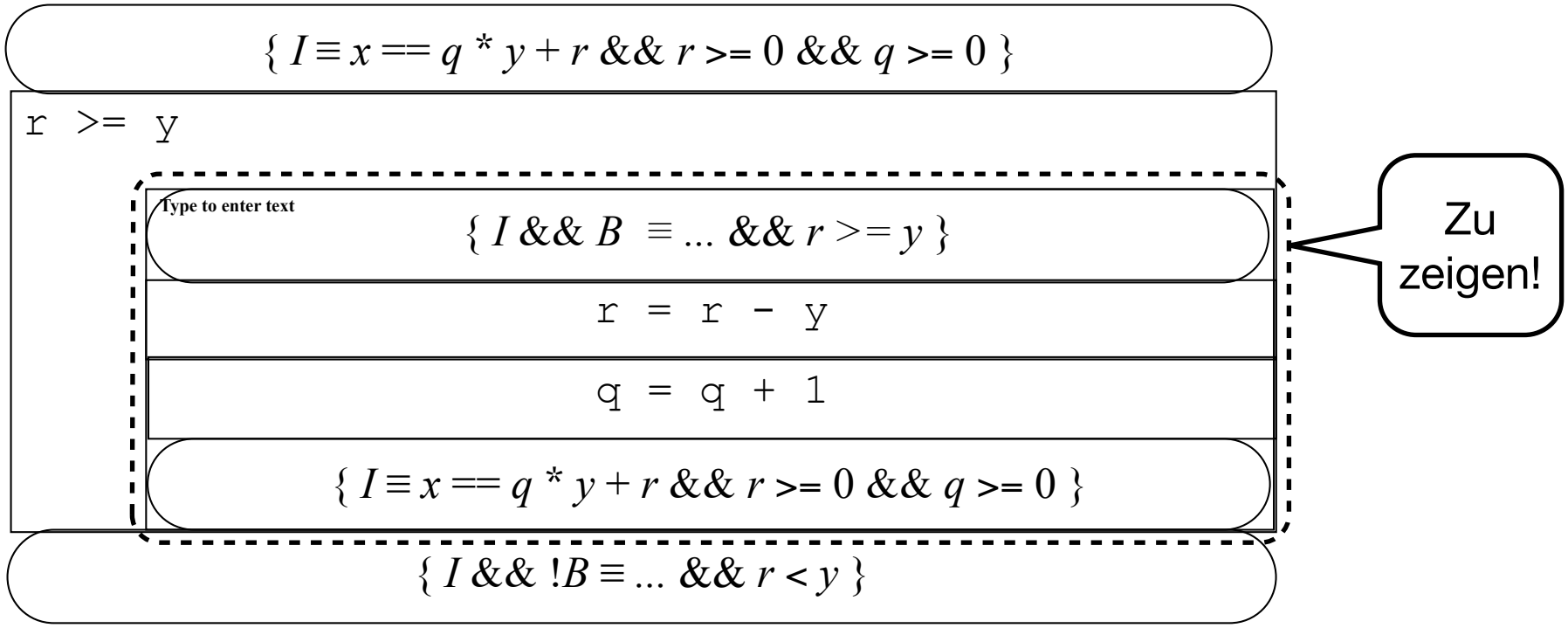
- *B* ≡ $r >= y$ (Ablesen aus dem Programm)
- *I* ≡ $x == q * y + r \;\&\&\; r >= 0 \;\&\&\; q >= 0$
- !*B* ≡ $r < y$ (Symbolische Negation von *B*)
- *I* && !*B* ≡ $x == q * y + r \;\&\&\; r < y \;\&\&\; r >= 0 \;\&\&\; q >= 0$

Letzteres ist die ***Nachbedingung*** des Programms.

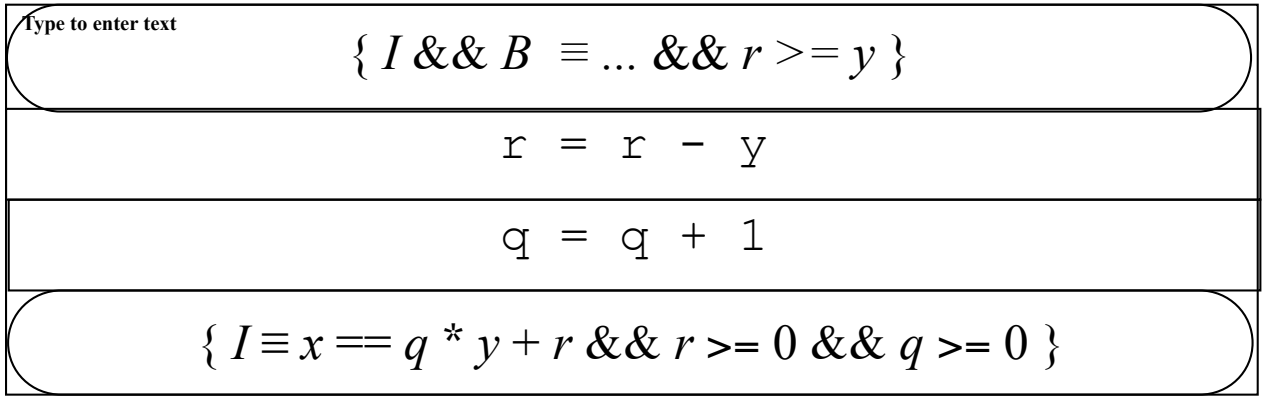
Einsetzen in die Schleife

{ I && B } S { I }

{ I } <u>while</u> B S { I && !B }

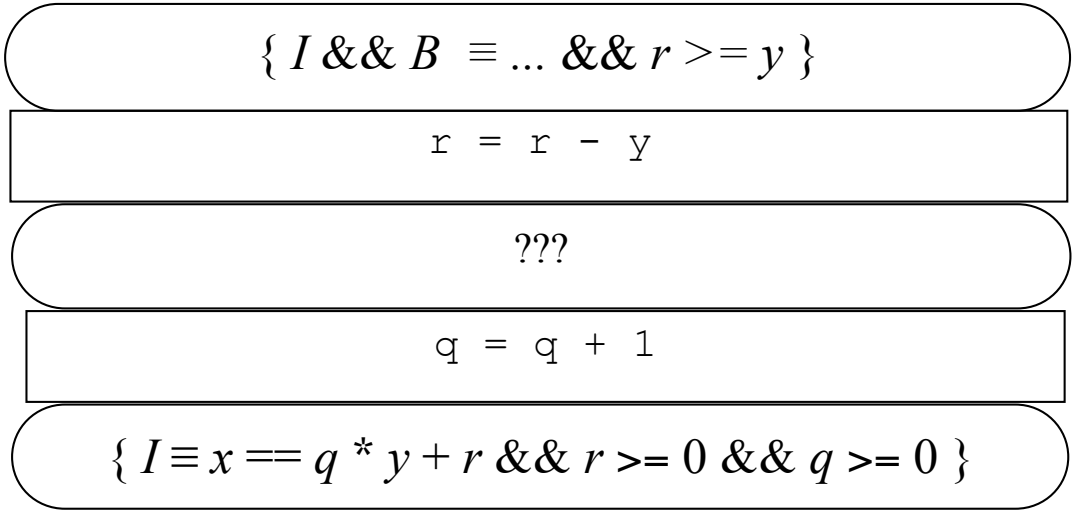


Zu zeigen: Korrektheit des Schleifenkörpers



Plan:
Wir führen unsere Beweise weiterhin "***von hinten***".

Betrachtung von Anweisungsfolgen

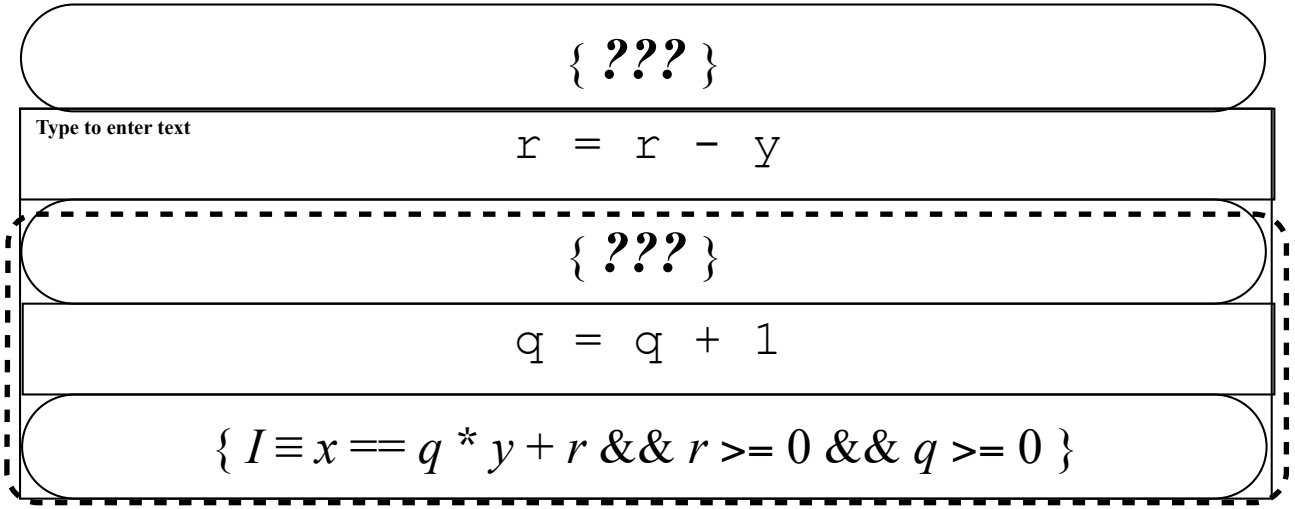


Hoare-Regel für die Anweisungsverkettung

$$\frac{\{ P \}\ S_1\ \{ Q \},\ \{ Q \}\ S_2\ \{ R \}}{\{ P \}\ S_1; S_2\ \{ R \}}$$

Zwei Anweisungen S_1 und S_2, welche nacheinander ausgeführt werden, können zu einem Programmstück $S_1\ S_2$ zusammengesetzt werden, wenn die Nachbedingung von S_1 mit der Vorbedingung von S_2 identisch ist.

TODO: Ablesen der Zwischen- und Vorbedingungen von hinten anfangend



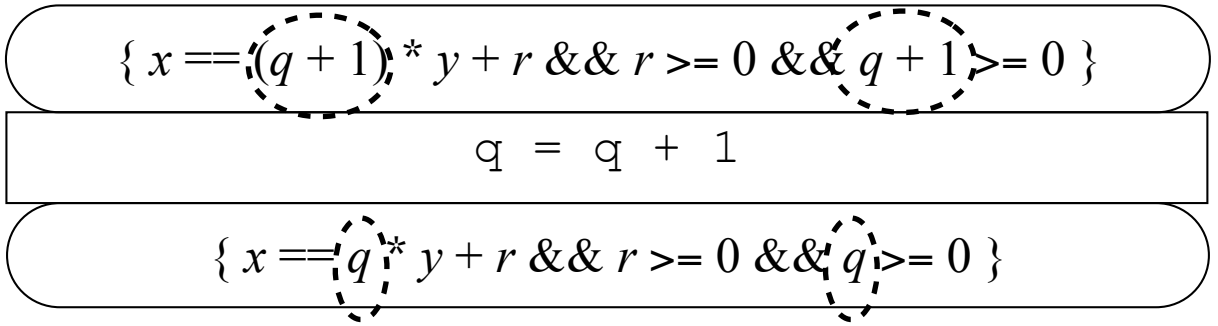
Hoare-Regel (Axiom) für Zuweisung

{ P[x/t] } x = t { P }

Das ist ein Axiom -- eine Regel ohne Prämissen.

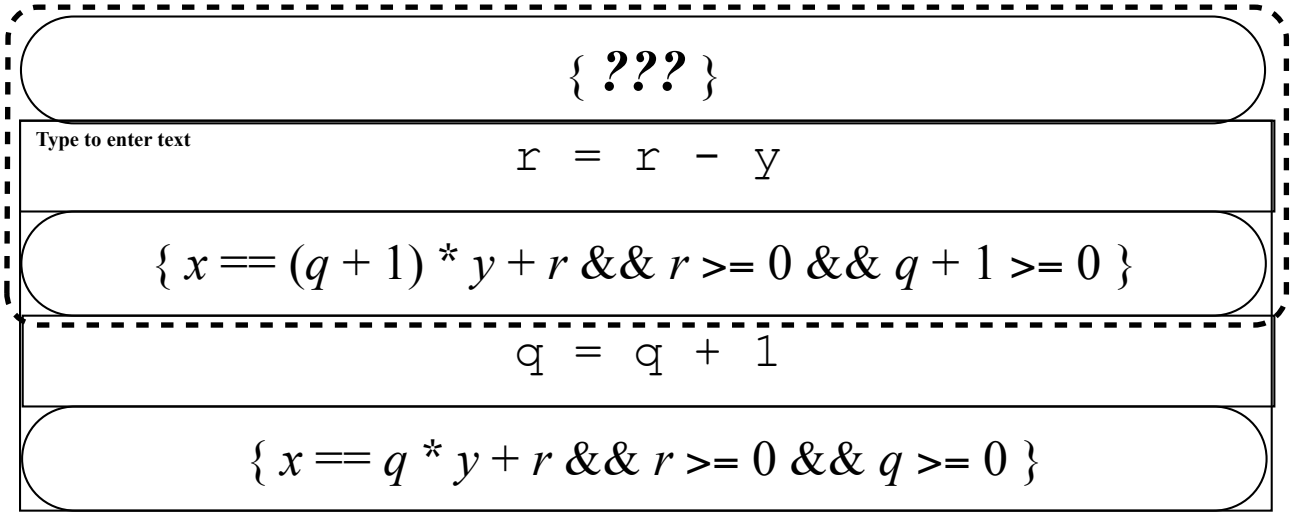
Substitution von *x* durch *t*. Wir setzen hier voraus, dass *t* keine Seiteneffekte hat.

Hintere Zuweisung in der Schleife



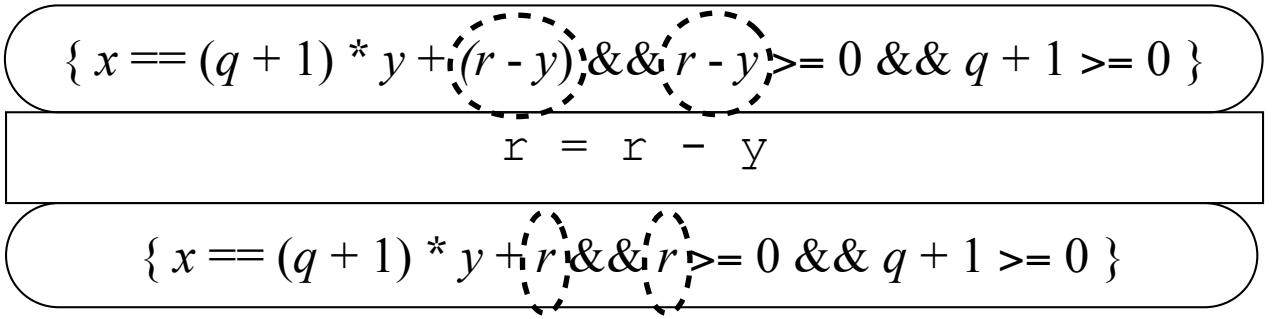
Ermittlung der Vorbedingung aus der Nachbedingung.
Substituiere *q* durch *q* + 1.

TODO: Ablesen der Vorbedingung



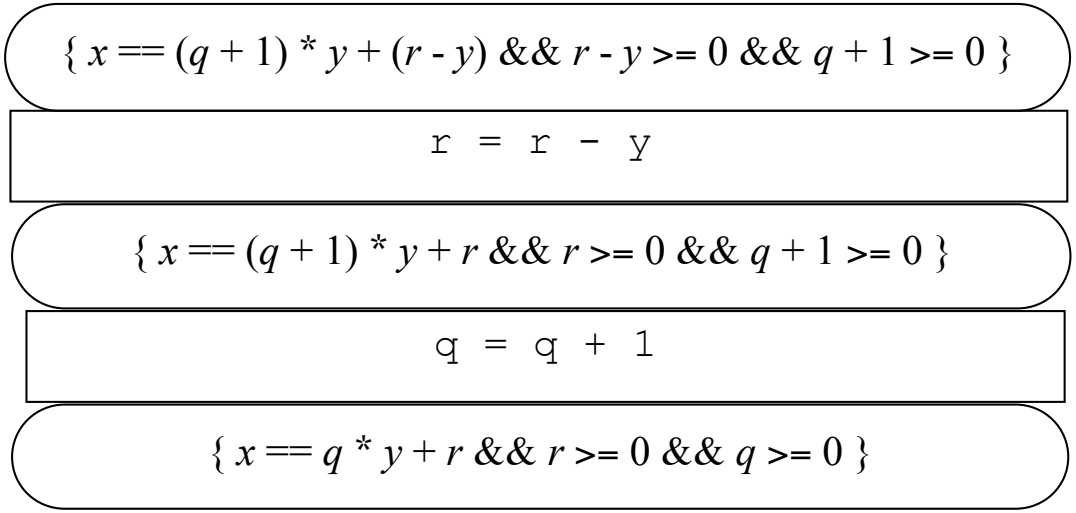
Nun müssen wir weiter nach vorn gehen.

Vordere Zuweisung in der Schleife



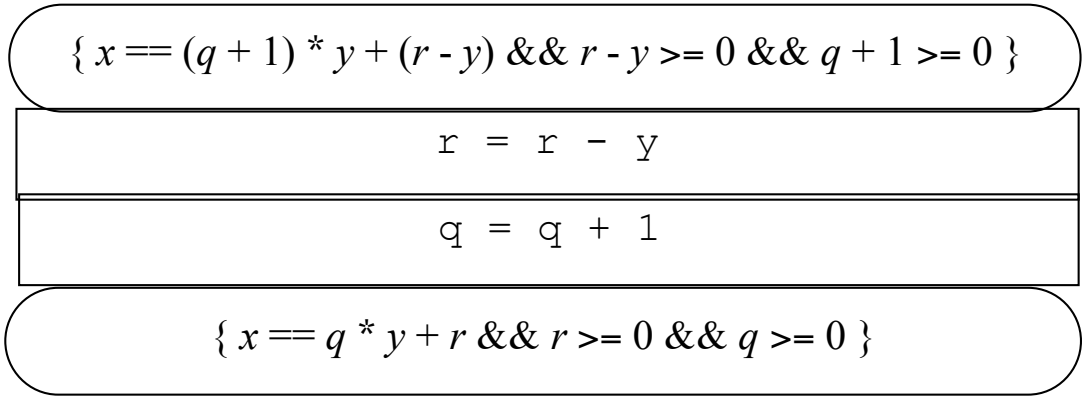
Ermittlung der Vorbedingung aus der Nachbedingung.
Substituiere *r* durch *r* - *y*.

Betrachtung von Anweisungsfolgen



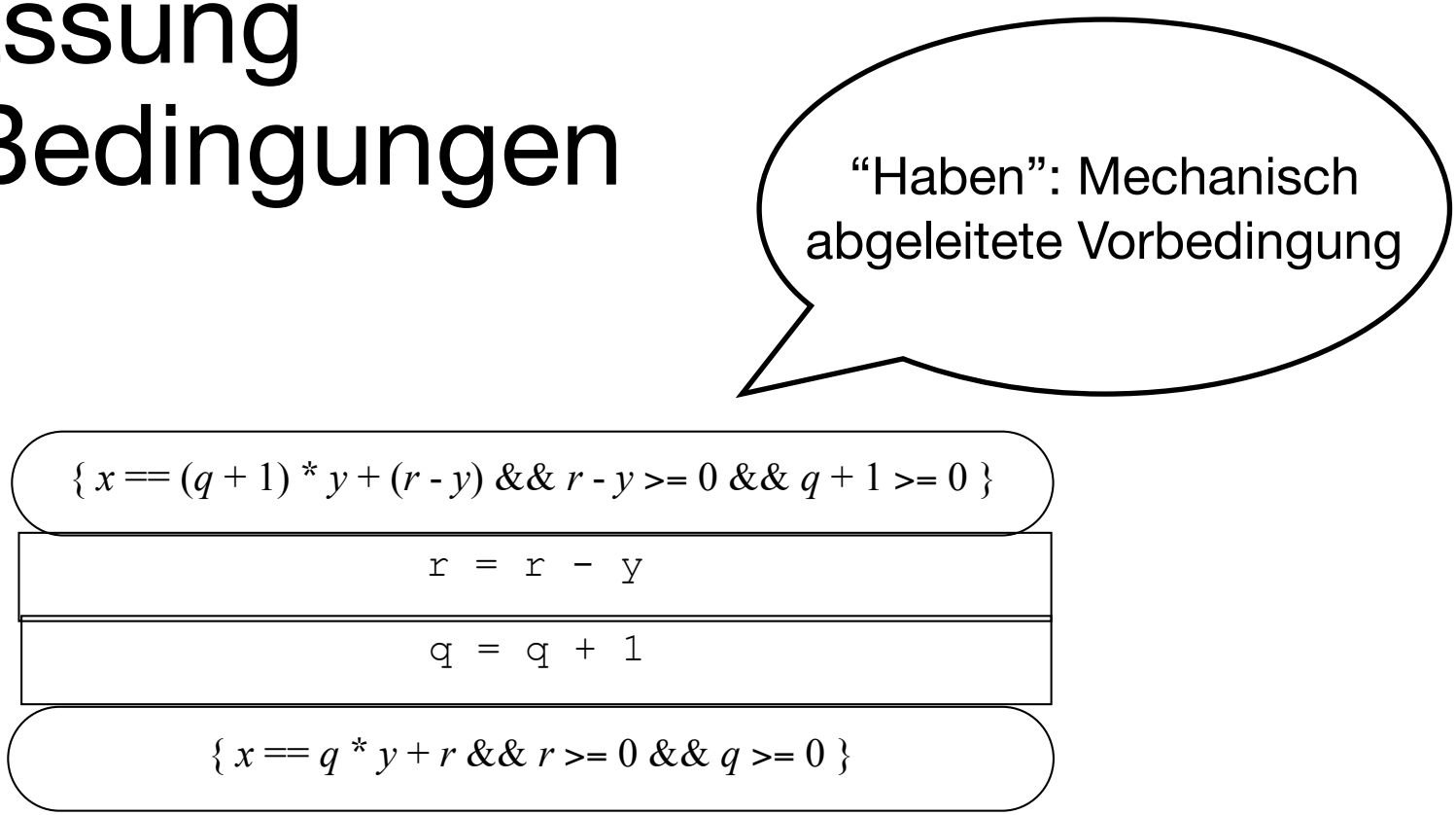
Die innere Bedingung war nur Mittel zum Zweck.

Betrachtung von Anweisungsfolgen



Die innere Bedingung war nur Mittel zum Zweck.

Anpassung von Bedingungen



Anpassung von Bedingungen

$\{ I \;\&\&\; B \equiv x == q * y + r \;\&\&\; r >= 0 \;\&\&\; q >= 0 \;\&\&\; r >= y \}$

```
r = r - y
```

```
q = q + 1
```

$\{ x == q * y + r \;\&\&\; r >= 0 \;\&\&\; q >= 0 \}$

{ I && B } S { I }

{ I } while B S { I && !B }

Anpassung von Bedingungen

- “Soll”
  - $x == q * y + r \;\&\&\; r >= y \;\&\&\; r >= 0 \;\&\&\; q >= 0$
- “Haben”
  - $x == (q + 1) * y + (r - y) \;\&\&\; r - y >= 0 \;\&\&\; q + 1 >= 0$

Plan: Wir ordnen die einzelnen Operanden der Konjunktionen aus den beiden Formeln einander zu.

Anpassung von Bedingungen

- "Soll"
  - $x == q * y + r \;\&\&\; r >= y \;\&\&\; r >= 0 \;\&\&\; q >= 0$
- "Haben"
  - $x == (q + 1) * y + (r - y) \;\&\&\; r - y >= 0 \;\&\&\; q + 1 >= 0$
- Nebenrechnung

$$x == (q + 1) * y + (r - y) \qquad \text{(Ausmultiplizieren)}$$

$$\Leftrightarrow \quad x == q * y + y + r - y \qquad \text{(... } +y \text{ und ... } -y\text{)}$$

$$\Leftrightarrow \quad x == q * y + r$$

Anpassung von Bedingungen

- "Soll"
  - ... && $r >= y$ && $r >= 0$ && $q >= 0$
- "Haben"
  - ... && $r - y >= 0$ && $q + 1 >= 0$
- Nebenrechnung

$$r - y >= 0 \qquad \text{(Addiere } y\text{)}$$

$$\Leftrightarrow r >= y$$

Anpassung von Bedingungen

- "Soll"
  - ... && ... && $r >= 0$ && $q >= 0$
- "Haben"
  - ... && ... && $q + 1 >= 0$
- **"Soll" ⇒ "Haben"**
  - $q >= 0 \Rightarrow q + 1 >= 0$

Die Ableitung der Vorbedingung aus der Nachbedingung ergibt die schwächste, solche Bedingung. Wir können auch mehr fordern.

Anpassung von Bedingungen

- “Soll”
  - ... && ... && $r >= 0$ && ...
- “Haben”
  - ... && ... && ...
- **“Soll” ⇒ “Haben”**
  - $r >= 0 \Rightarrow \text{true}$

Die Ableitung der Vorbedingung aus der Nachbedingung ergibt die schwächste, solche Bedingung. Wir können auch mehr fordern.

Schwäche/Stärke von Bedingungen

- Zielstellung:
  - Vorbedingungen so schwach wie möglich.
  - Nachbedingungen so stark wie möglich.
- Definitionen
  - *x* ist schwächer als *y* wenn $y \Rightarrow x$.
  - *x* ist *echt* schwächer als *y* wenn nicht auch $x \Rightarrow$ y.

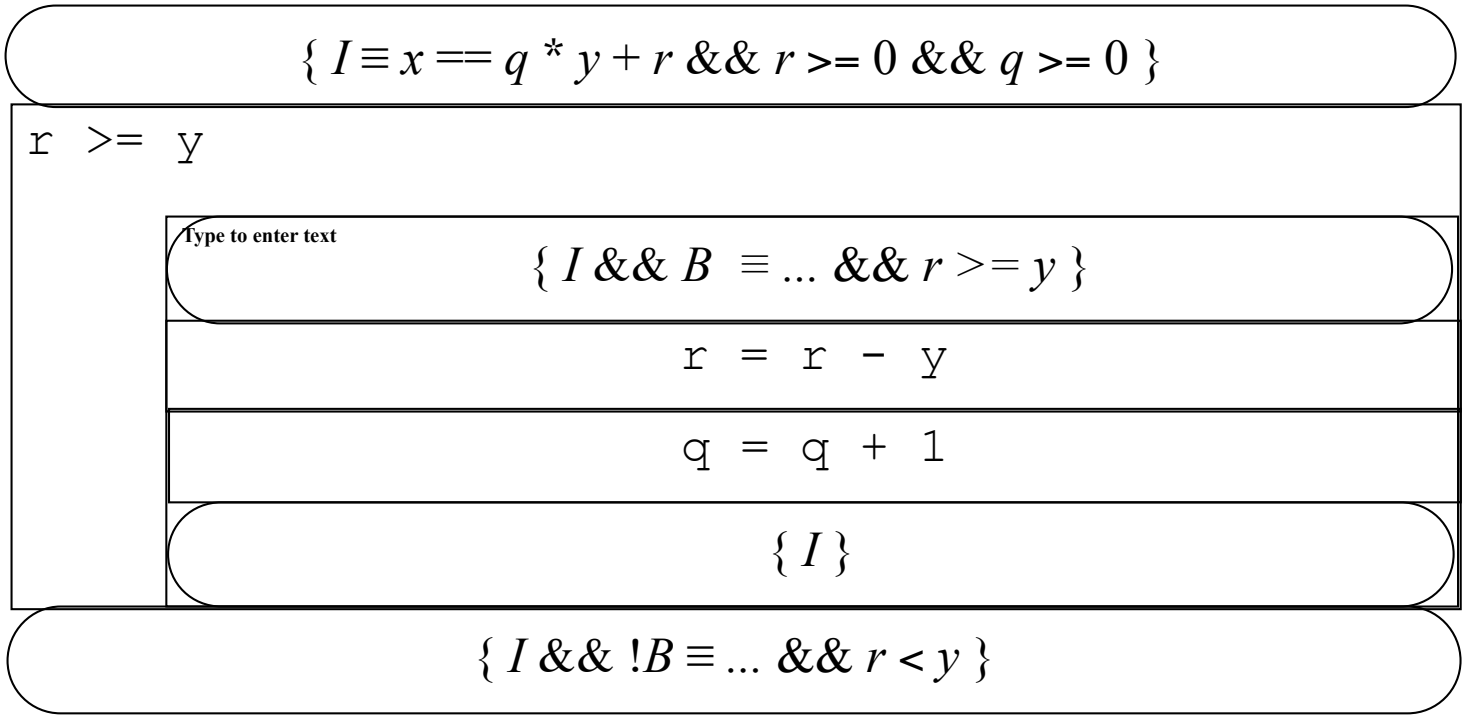
Hoare-Regel
für die Verstärkung der Vorbedingung

$$\frac{P \Rightarrow Q,\ \{Q\}\,S\,\{R\}}{\{P\}\,S\,\{R\}}$$

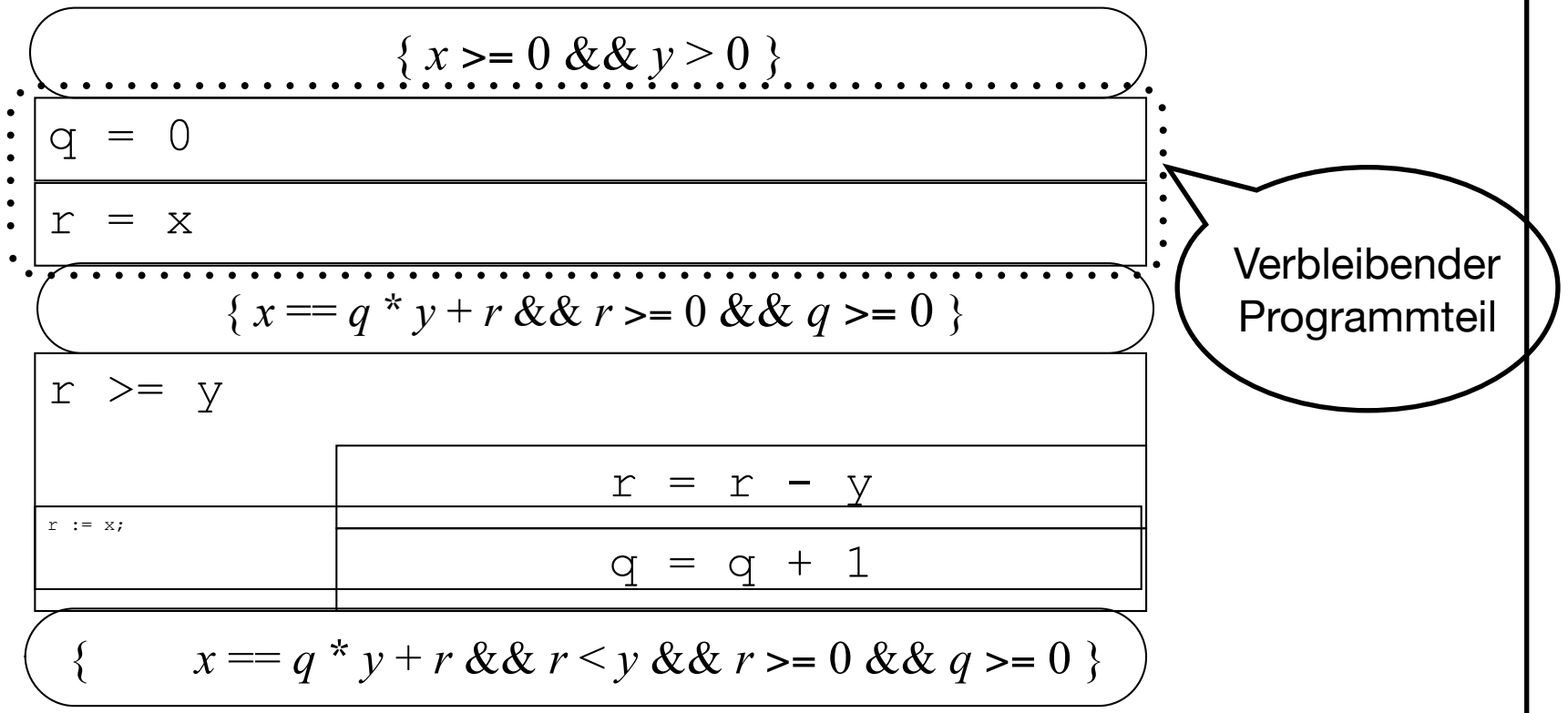
{ I && B } S { I }

{ I } while B S { I && !B }

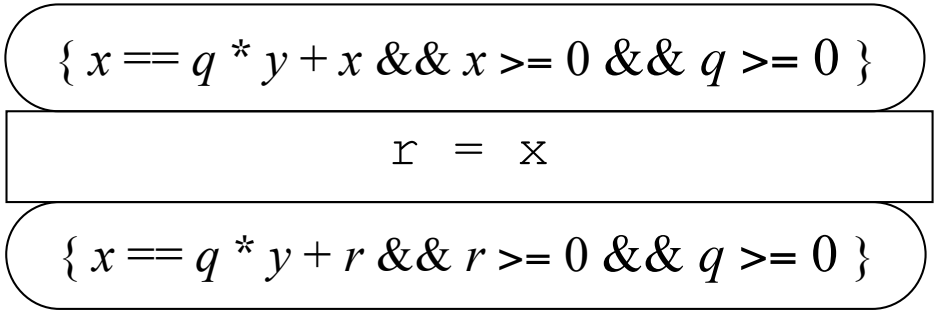
Die Schleife ist bewiesen.



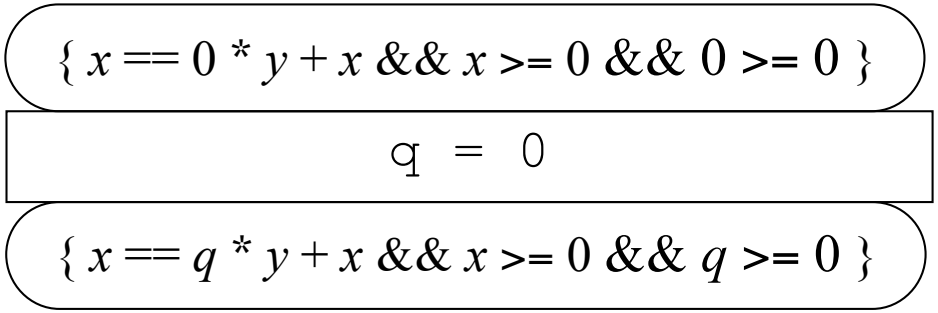
Fortsetzung des Gesamtbeweises (ohne neue Ideen)



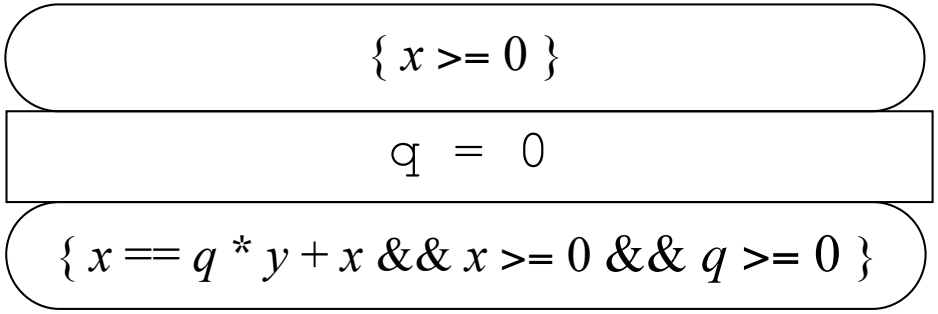
Hintere Zuweisung



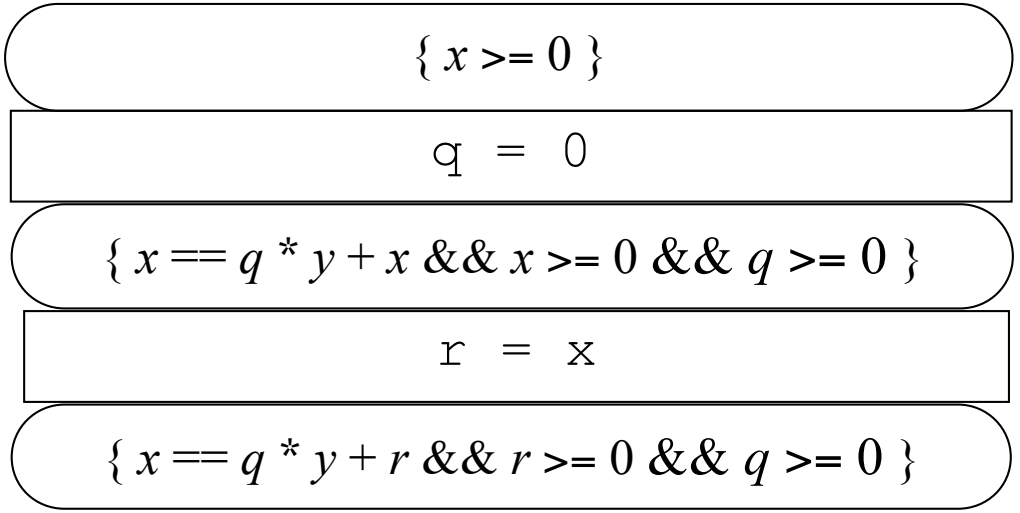
Vordere Zuweisung



Vordere Zuweisung (nach Vereinfachung)

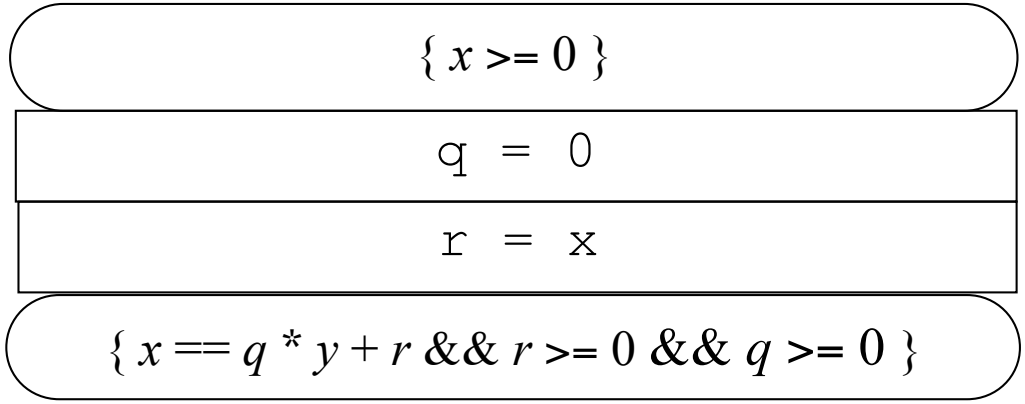


Betrachtung von Anweisungsfolgen



Die innere Bedingung war nur Mittel zum Zweck.

Betrachtung von Anweisungsfolgen



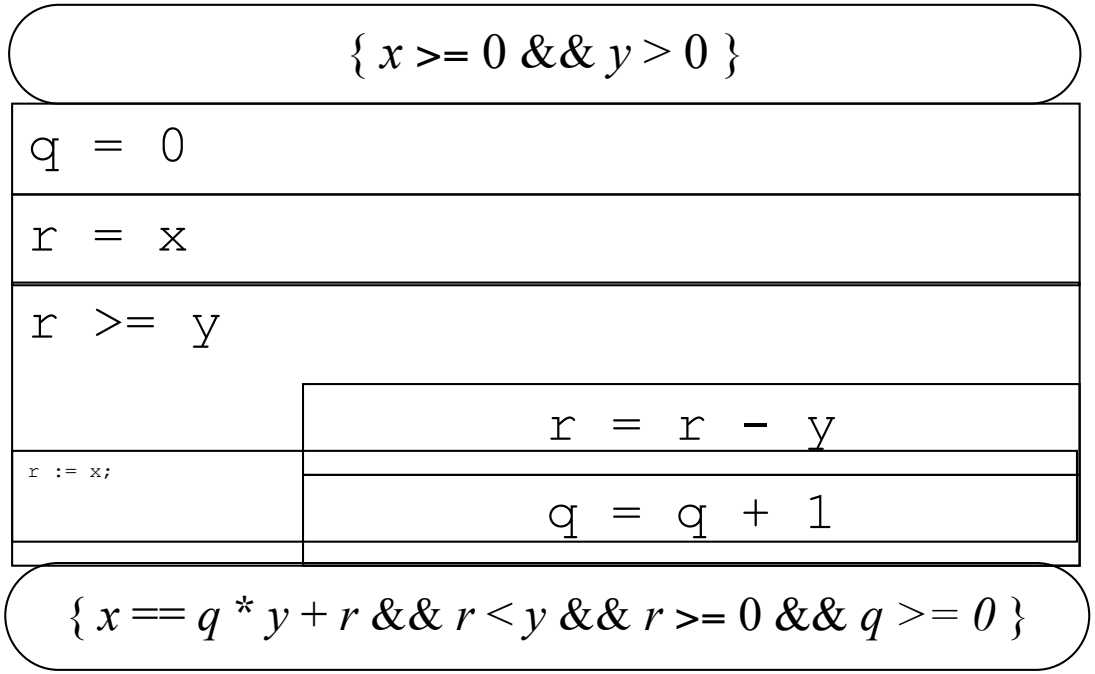
Die innere Bedingung war nur Mittel zum Zweck.

Anpassung von Bedingungen

- “Soll”
  - $x >= 0 \;\&\&\; y > 0$
- “Haben”
  - $x >= 0$
- **“Soll” ⇒ “Haben”**

Die Ableitung der Vorbedingung aus der Nachbedingung ergibt die schwächste, solche Bedingung. Wir können auch mehr fordern -- *etwa mit dem Ziel der Terminierung im Auge*.

Das Programm mit Spezifikation ist korrekt.



Q.E.D.

Zusammenfassung der Methode zum Beweis der Korrektheit mit Hoare-Regeln

- Programme mit Spezifikationen:

  { Vorbedingung } Programm { Nachbedingung }

- Vor- und Nachbedingungen sind logische Formeln:

  Formeln beschreiben Annahmen über Variablenbelegungen.

- Es gibt Korrektheitsregeln für jede Anweisungsform.
- Vorgehenweise beim Korrektheitsbeweis
  - Ergänze alle Programmstücke mit Vor- und Nachbedingungen.
  - Überprüfe alle Tripel auf Korrektheit gemäß Regeln.

Beweisregeln

$$\frac{A_1, \ldots, A_n}{A_0}$$

- Jedes A_i ist ...
  - entweder ein Hoare-Tripel
  - oder eine prädikatenlogische Formel.
- Benutzung Boolescher Ausdrücke der Programmiersprache.

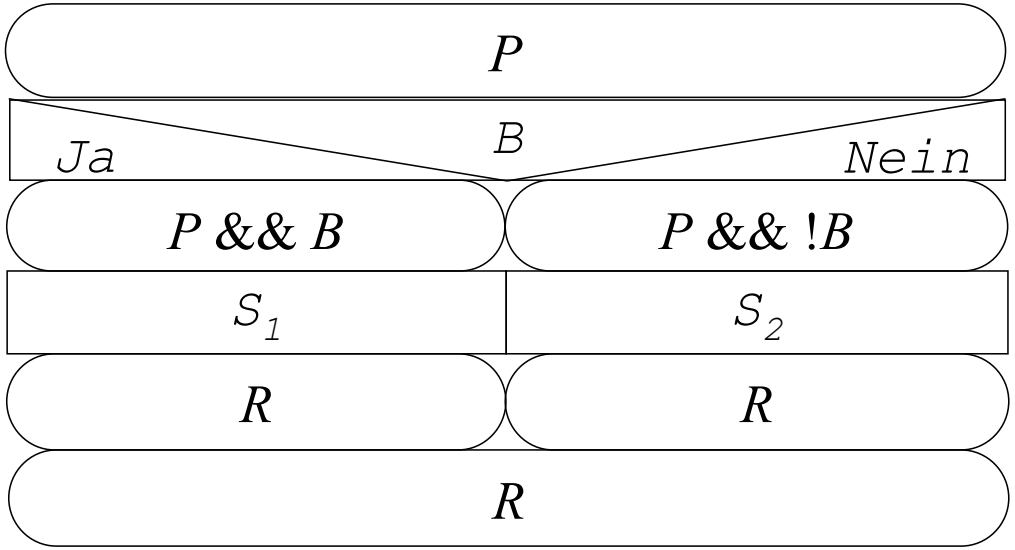
Arten von Regeln

- Programmiersprachenabhängige Regeln
  - Zuweisung
  - If-Anweisung
  - ...
- Programmiersprachen**un**abhängige Regeln
  - Anpassung von Vor- und Nachbedingungen
  - Prädikatenlogische Regeln

Hoare-Regel für If-Then-Else

$$\{ P \;\&\&\; B \}\; S_1 \;\{ R \},\quad \{ P \;\&\&\; !B \}\; S_2 \;\{ R \}$$

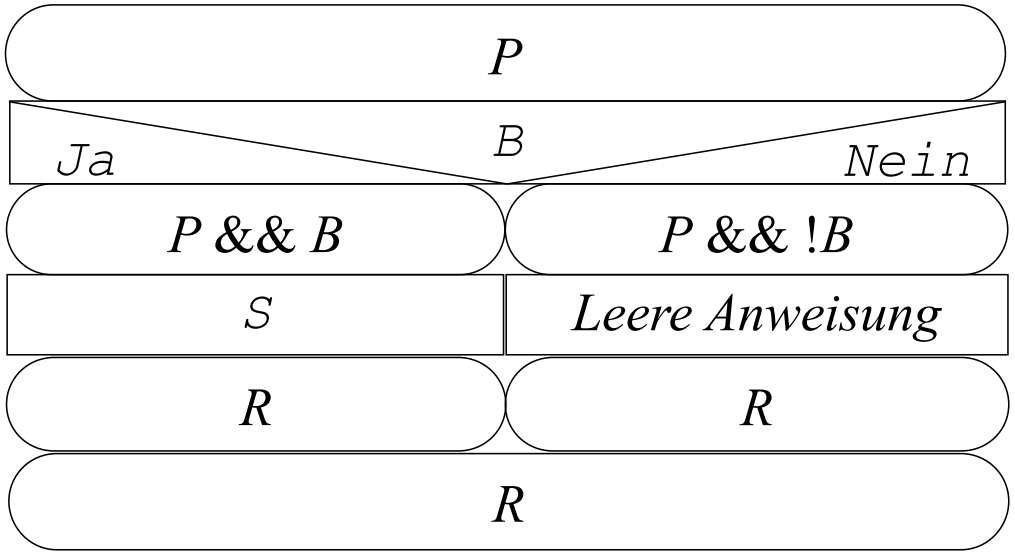
$$\{ P \}\; \underline{\text{if}}\; (B)\; S_1\; \underline{\text{else}}\; S_2 \;\{ R \}$$



Hoare-Regel für If-Then

{ P && B } S { R }, P && !B ⇒ R

{ P } <u>if</u> (B) S { R }



Hoare-Regel
für die Verstärkung der Vorbedingung

$$\frac{P \Rightarrow Q,\ \{Q\}\,S\,\{R\}}{\{P\}\,S\,\{R\}}$$

Hoare-Regel
für die Abschwächung der Nachbedingung

$$\frac{\{P\}S\{Q\},\ Q \Rightarrow R}{\{P\}S\{R\}}$$

Diese Regel wird im Beispielbeweis nicht verwendet.

Im Beispiel verwendete Regeln

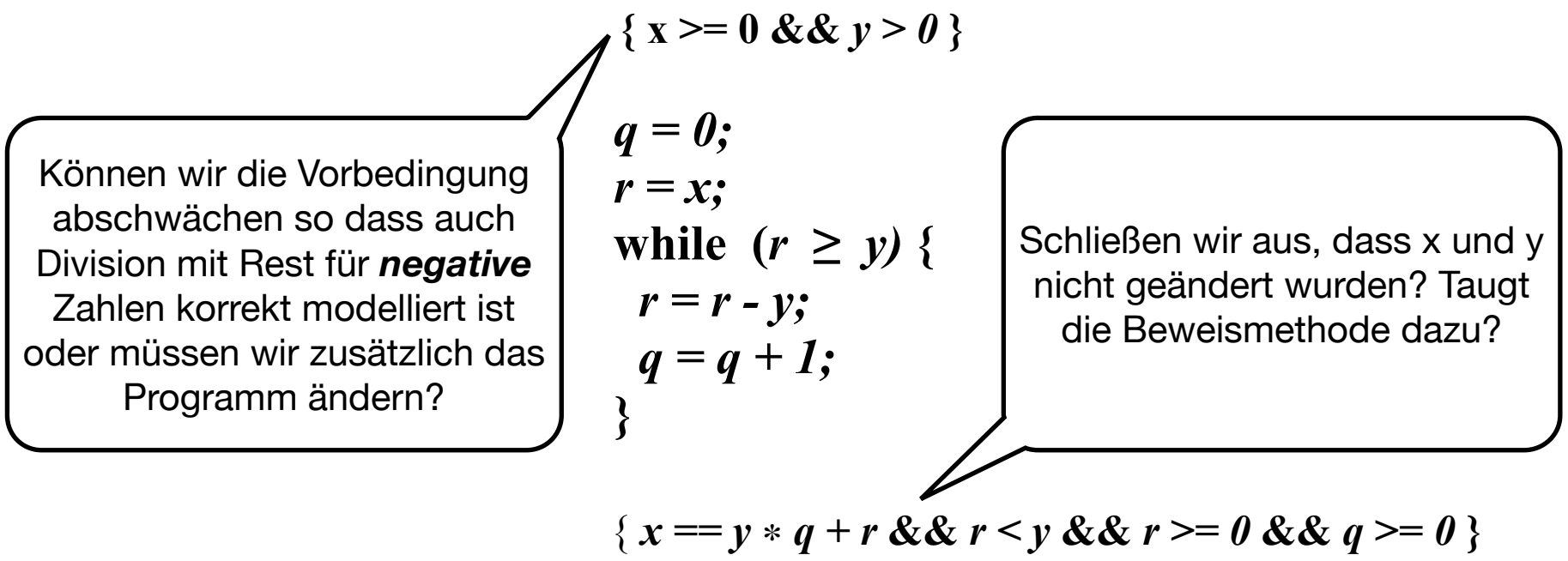
$$\frac{\{ P \}\ S_1\ \{ Q \},\ \{ Q \}\ S_2\ \{ R \}}{\{ P \}\ S_1; S_2\ \{ R \}}$$

$$\{ P[x/t] \}\ x = t\ \{ P \}$$

$$\frac{\{ I \mathbin{\&\&} B \}\ S\ \{ I \}}{\{ I \}\ \underline{\text{while}}\ (B)\ S\ \{ I \mathbin{\&\&} !B \}}$$

$$\frac{P \Rightarrow Q,\ \{ Q \}\ S\ \{ R \}}{\{ P \}\ S\ \{ R \}}$$

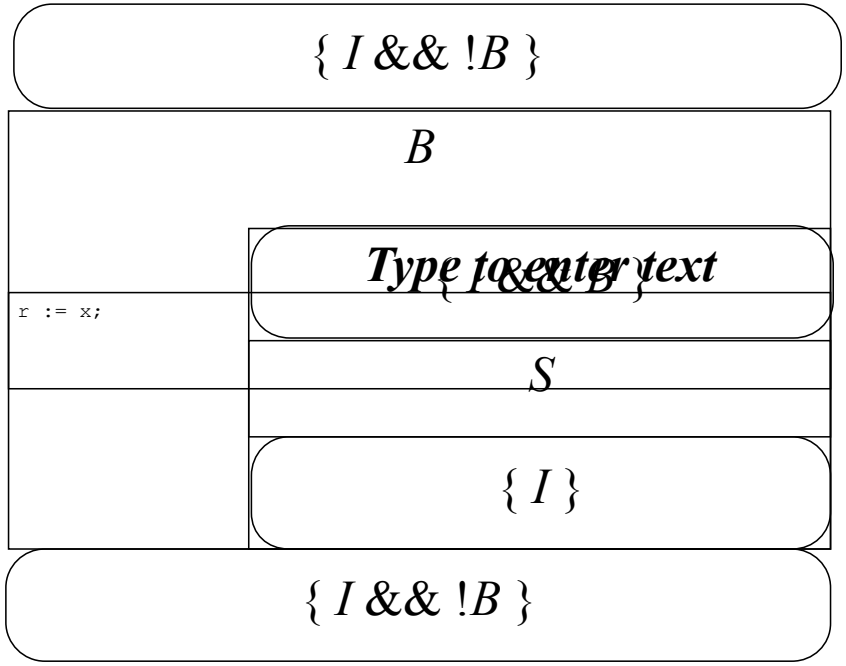
2 Fangfragen



Situationen beim Beweisen

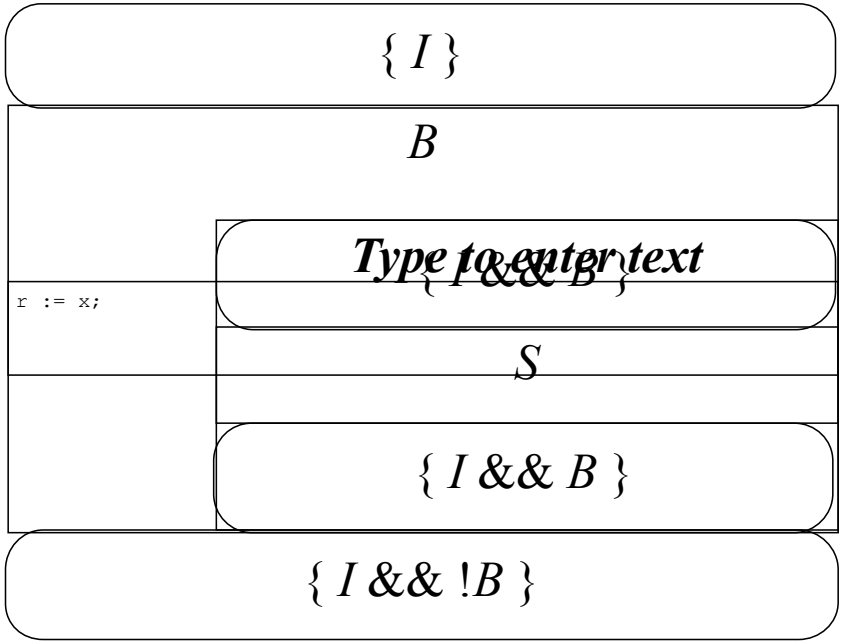
- Tote Schleife
- Endlosschleife
- Inkorrekte Schleife
- Erfolgreicher Beweis
- ...

Tote Schleife



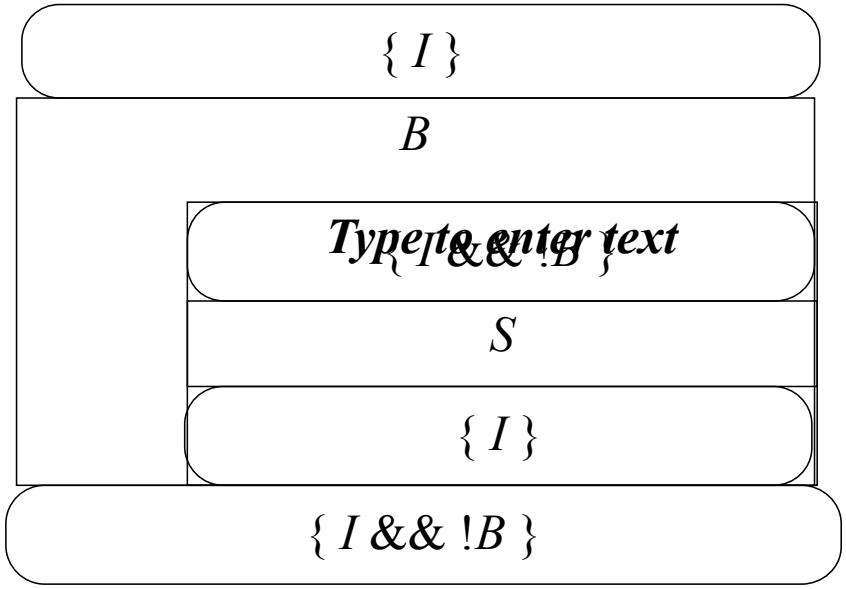
Die Schleifenbedingung ist definitiv vor der Schleife unerfüllt.

Endlosschleife



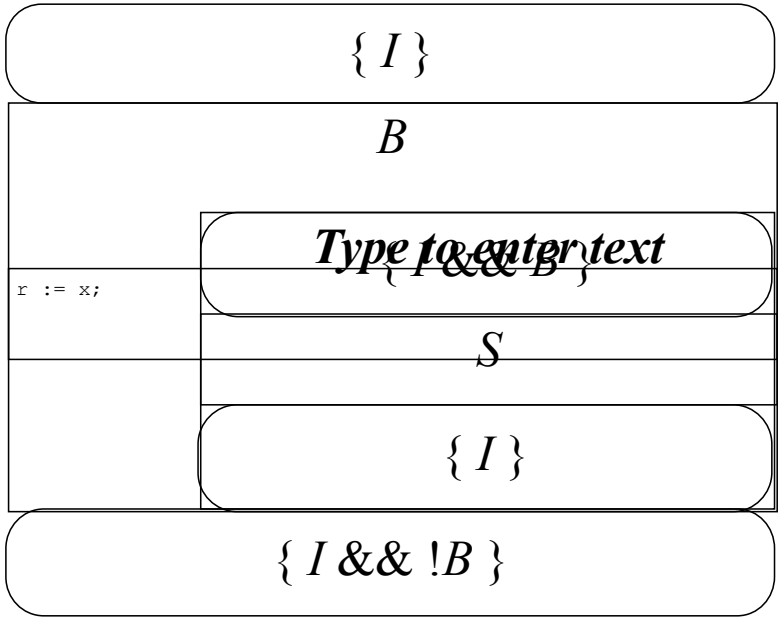
Der Schleifenkörper stellt immer wieder die Gültigkeit der Schleifenbedingung her.

Inkorrekte Schleife



Der Schleifenkörper verlangt, dass die negierte Schleifenbedingung zum Eintritt in die Schleife gelten muss.

Erfolgreicher Beweis



Die Vorbedingung (eventuell durch Abschwächung) lässt sich aus der Nachbedingung ableiten.

Zusammenfassung

- Einfache Programme sind nicht einfach korrekt.
- Softwareentwickler benötigen mathematische Fähigkeiten.
- Softwareentwicklung benötigt Beweisautomatisierung.
- Kosten von Korrektheit motivieren Wiederverwendung.

Ausblick

- Schnittstellen
- Zeiger
- Abstrakte Datentypen

Weitere Verifikationsbeispiele

Verifikationsbeispiel: Schnelle Potenzierung

- Problem
  - Berechne x^n
- Naive Methode
  - $x^n = \underbrace{x * \ldots * x}_{n \text{ mal}}$
  - Kosten: n - 1 Multiplikationen
- Wie geht das schneller?

Schnelle Potenzierung

- Zu berechnen: x^n
- Fall 1: n ist gerade, etwa $n = 2k$
  - Berechne $x' = x^2$.
  - Gib x'^k zurück.
- Fall 2: n ist ungerade, etwa $n = 2k + 1$
  - Berechne $x' = x^2$.
  - Gib x * x'^k zurück.
- Dies ergibt O(*log n*) Multiplikationen.

Imperative, iterative Implementation Schneller Exponentation

```
public static int power(int x, int n) {
    int k = n;
    int p = x;
    int y = 1;
    while (k>0)
        if (k % 2 == 0) {
            p = p * p;
            k = k / 2;
        }
        else {
            y = y * p;
            k = k - 1;
        }
    return y;
}
```

Formierung eines Hoare-Tripels

$\{ n >= 0 \}$

```
int k = n;
int p = x;
int y = 1;
while (k>0)
    if (k % 2 == 0) {
        p = p * p;
        k = k / 2;
    }
    else {
        y = y * p;
        k = k - 1;
    }
```

$\{ y == x^n \}$

Was ist die Schleifen-Invariante *I*?

- Ansatz:
  - Wie stehen die Programmvariablen in Beziehung?
  - x und n sind konstant. Wir setzen $n = 10$.

y	***p***	***k***	***x***	***n***	$y * p^k$
1	x	10	x	10	x^{10}
1	x^2	5	x	10	x^{10}
x^2	x^2	4	x	10	x^{10}
x^2	x^4	2	x	10	x^{10}
x^2	x^8	1	x	10	x^{10}
x^{10}	x^8	0	x	10	x^{10}

Wie man “leicht” sieht: $I \equiv y * p^k == x^n \;\&\&\; k >= 0$

Verifikation der While-Schleife

$\{ I \equiv \quad y * p^k == x^n \;\&\&\; k >= 0 \}$

```
while k > 0
```

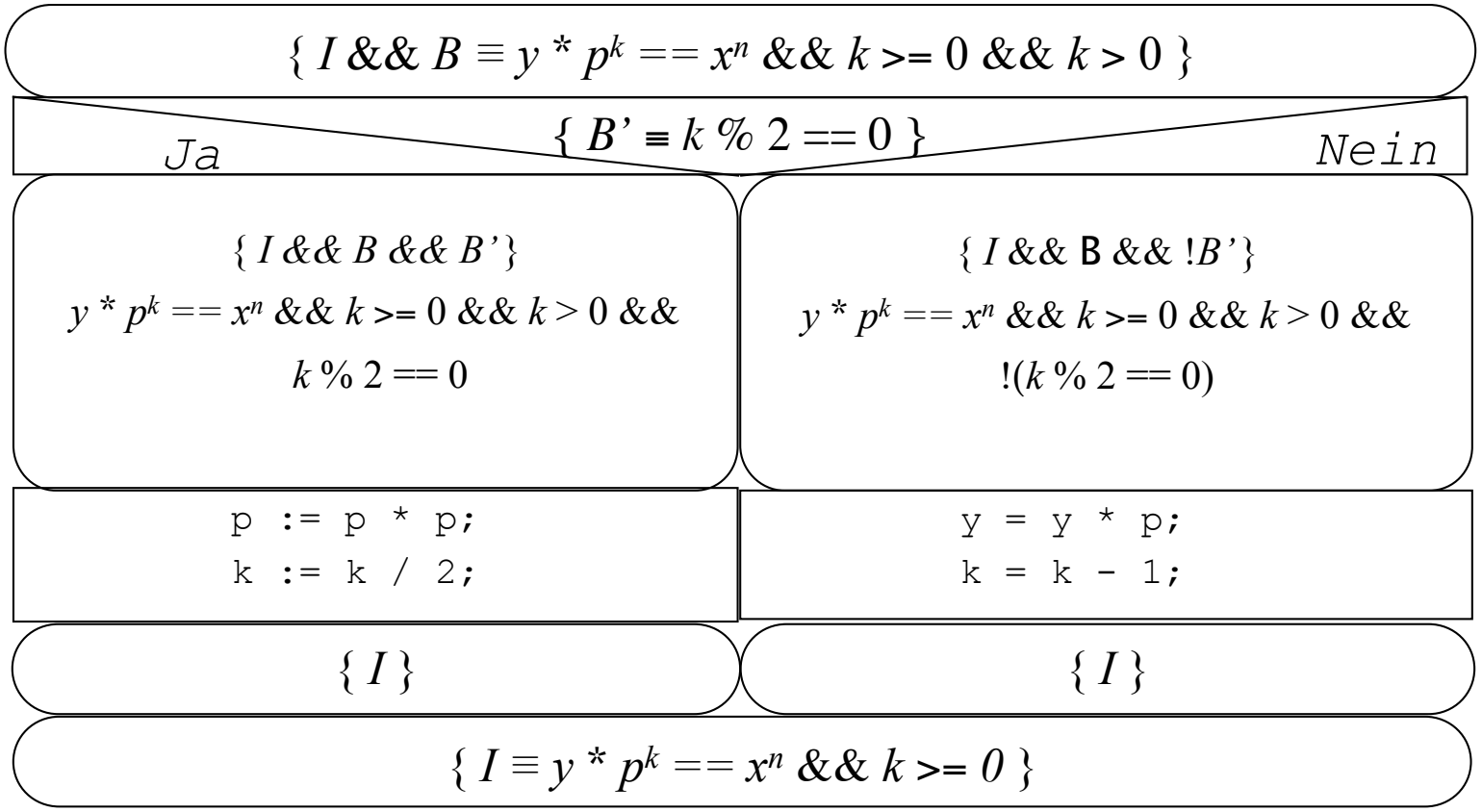
Type to enter text $\{ I \wedge B \equiv \quad \ldots \;\&\&\; k > 0 \}$

```
if (k % 2) == 0
{
        p := p * p;
        k := k / 2;
} else
{
        y = y * p;
        k = k - 1;
}
```

$\{ I \}$

$\{ I \wedge !B \equiv \quad \ldots \;\&\&\; !(k > 0) \}$

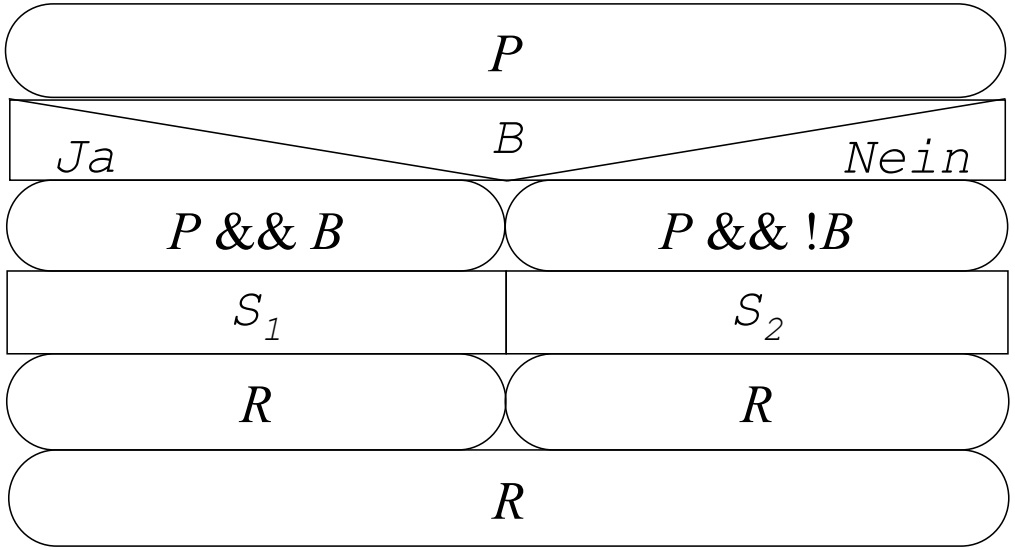
Verifikation der Fallunterscheidung



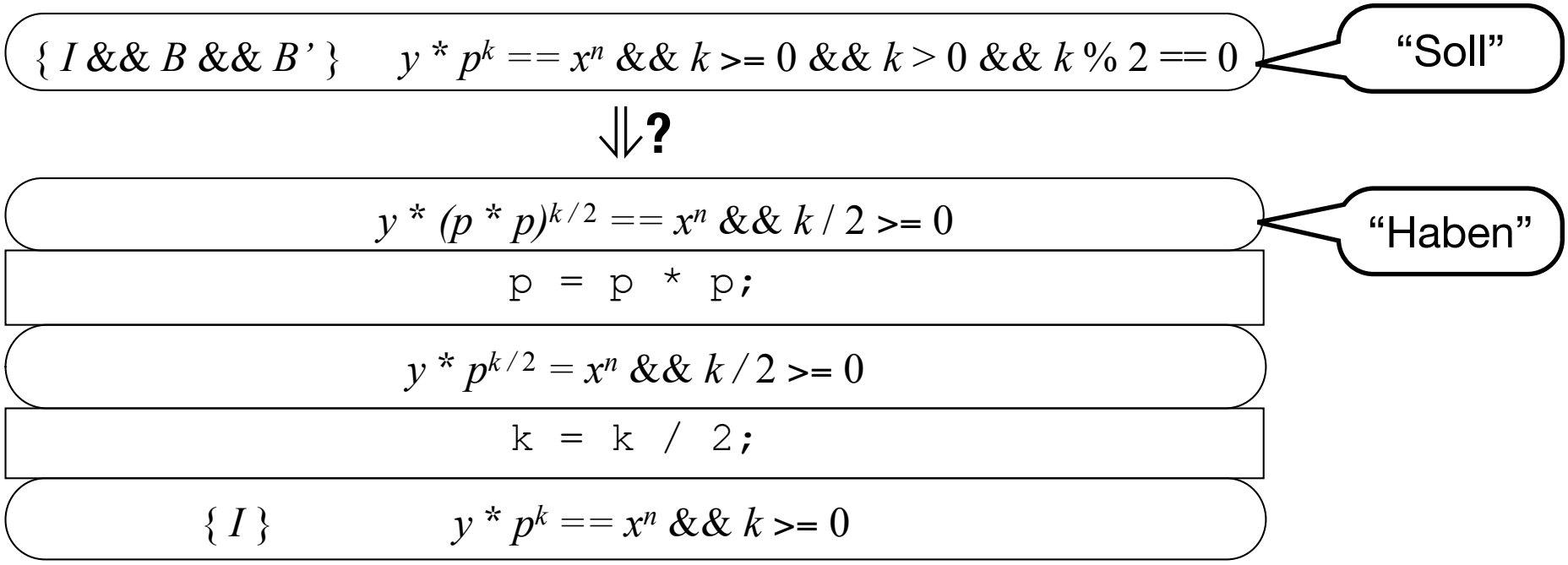
Hoare-Regel für If-Then-Else

$$\{ P \;\&\&\; B \}\; S_1 \;\{ R \},\quad \{ P \;\&\&\; !B \}\; S_2 \;\{ R \}$$

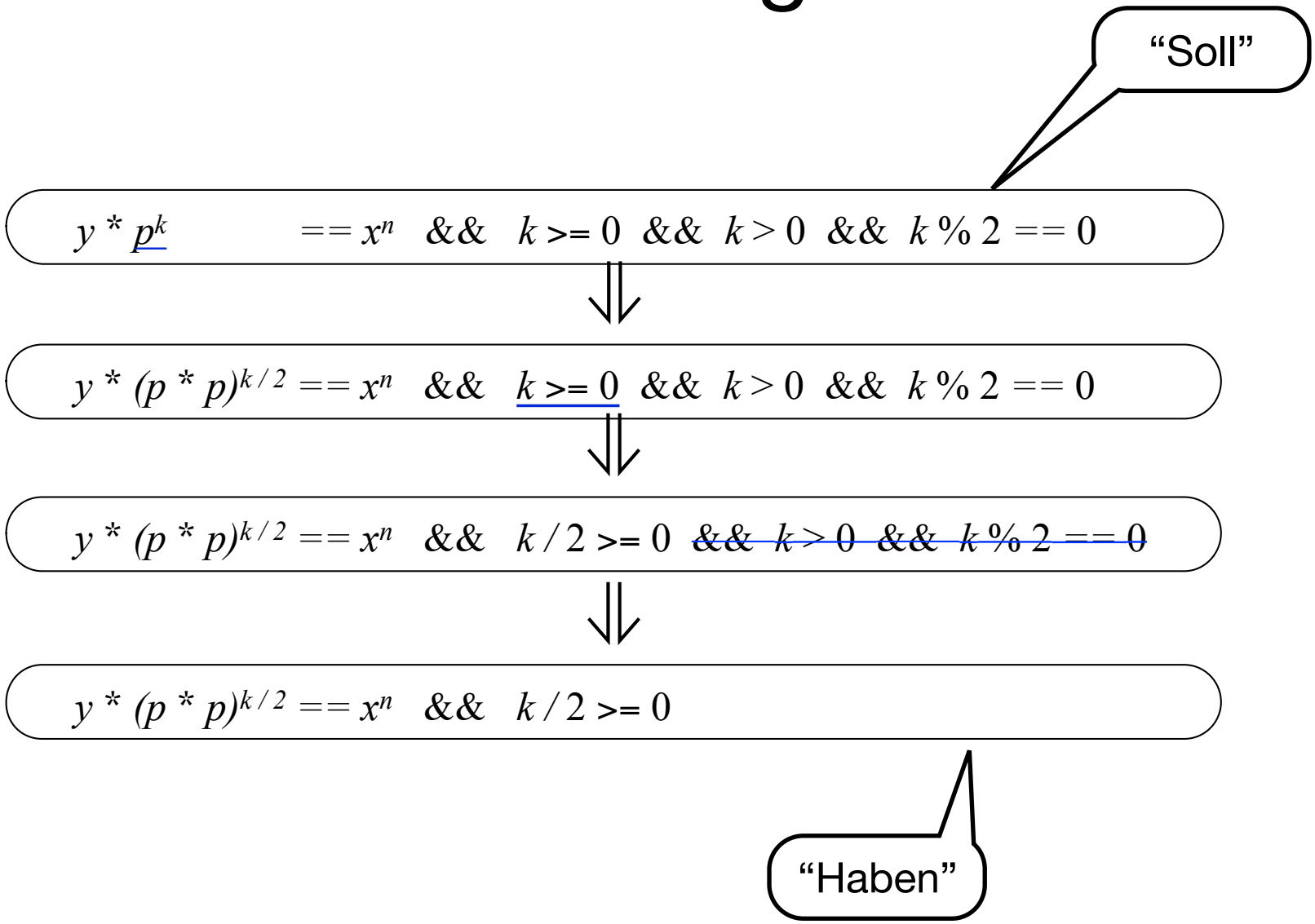
$$\{ P \}\; \underline{\text{if}}\; B\; S_1\; \underline{\text{else}}\; S_2 \;\{ R \}$$



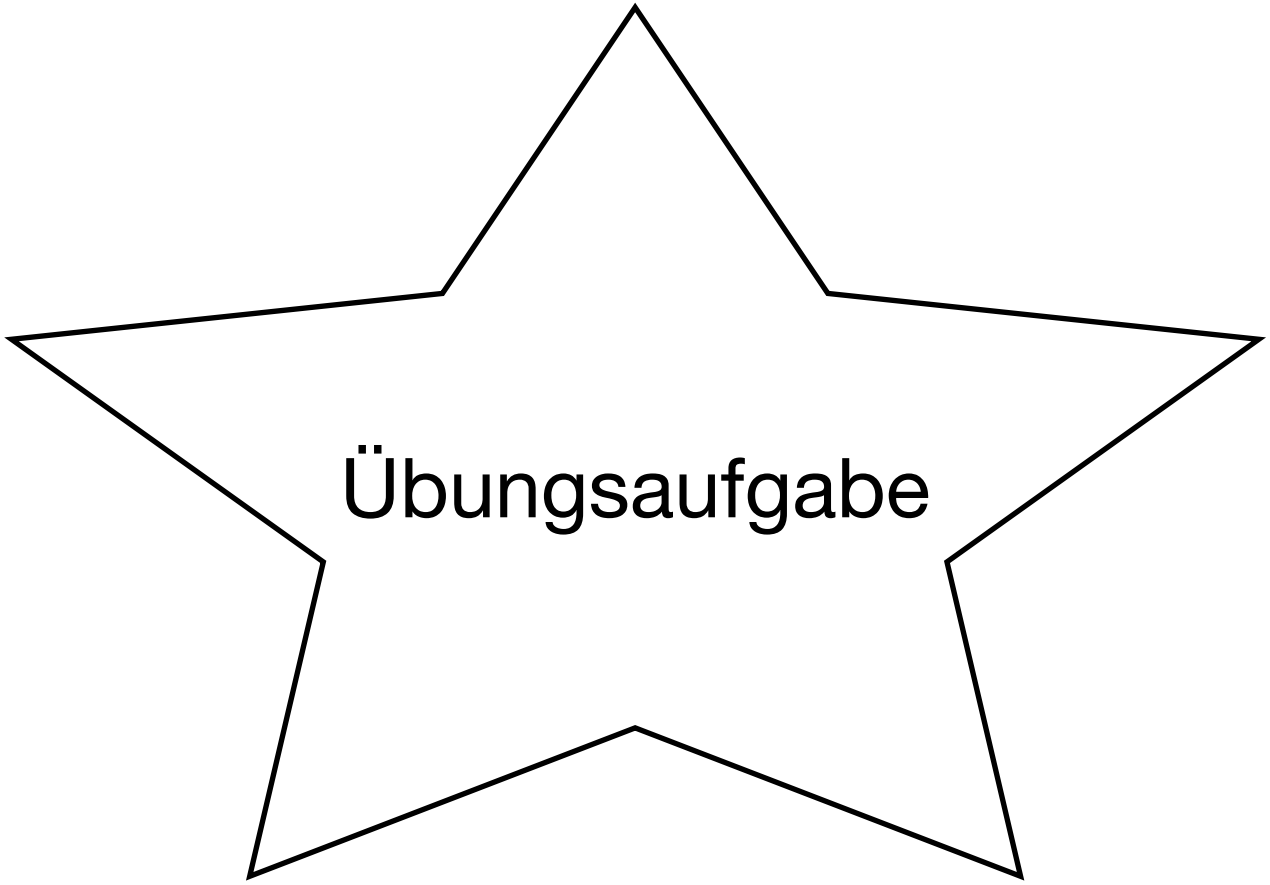
Fall 1: “Then”-Zweig



Fall 1: “Then”-Zweig



Fall 2: “Else”-Zweig



Verifikationsbeispiel: Berechnung des Quadrats einer Zahl

- Restriktionen:
  - ohne Verwendung von “*”
  - mit Verwendung von “+”.
- Eine Option für den Lösungsansatz:

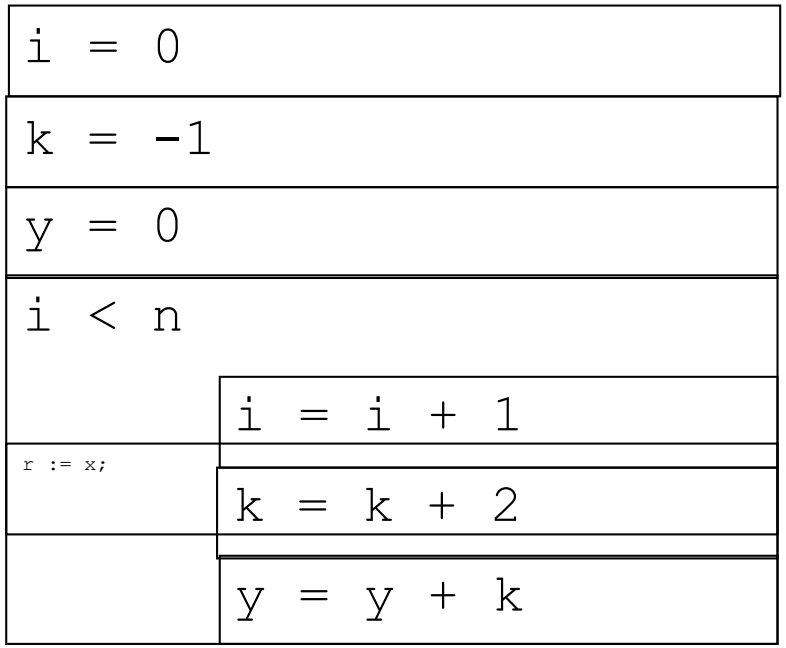
$$n^2 = \sum_{i=1}^{n}(2i-1) = 1 + 3 + 5 + \ldots + (2n-1)$$

Veranschaulichung der Generierung der Quadratzahlen

$$0 + 1 = 1 \qquad 1 + 3 = 4 \qquad 4 + 5 = 9 \qquad 9 + 7 = 16$$

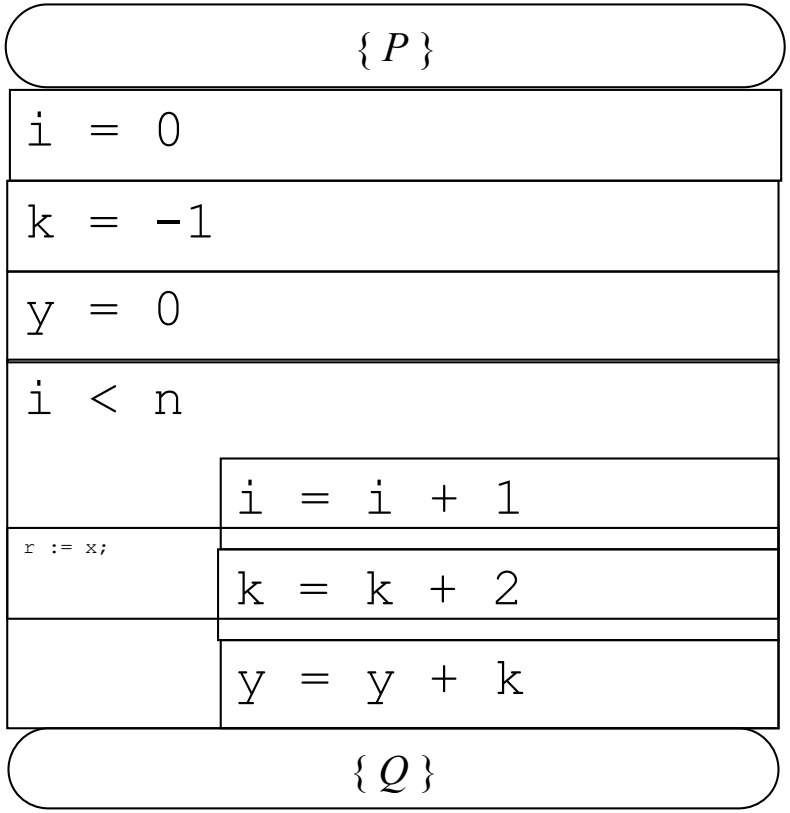
Quelle: http://de.wikipedia.org/wiki/Quadratzahl

Imperative, iterative Implementation



3

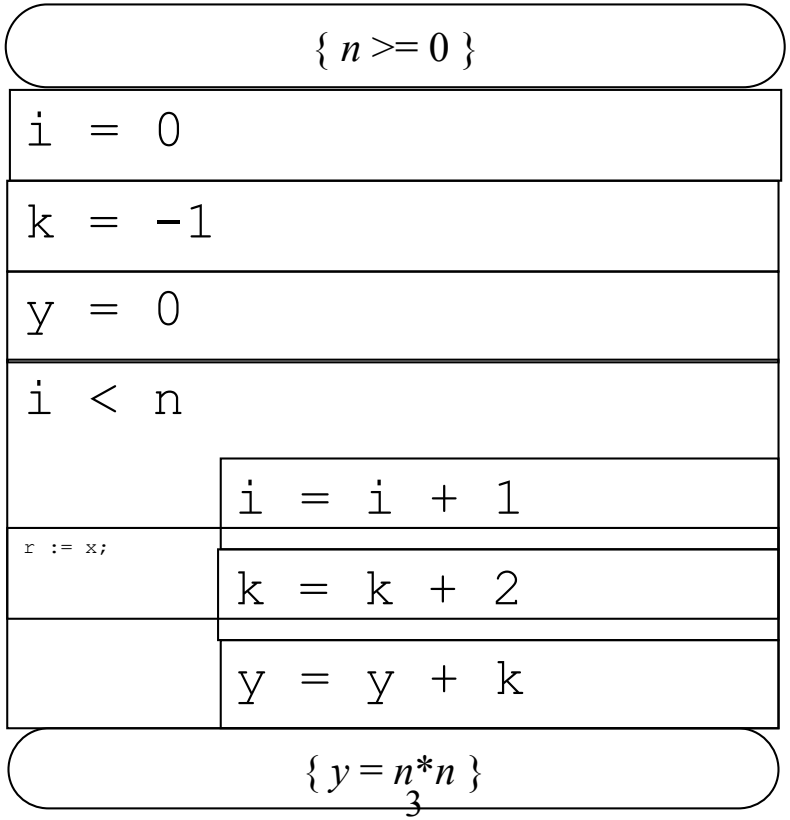
Formierung eines Hoare-Tripels



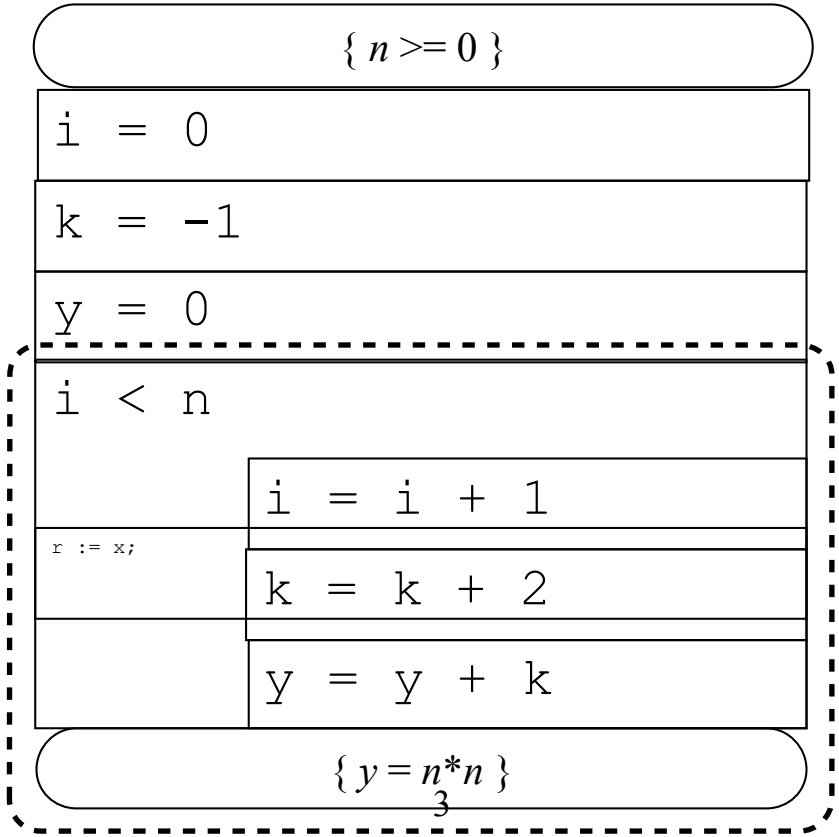
3

Welches sind die Vor- und Nachbedingungen?

Aufstellung der Spezifikation



Beweis von hinten

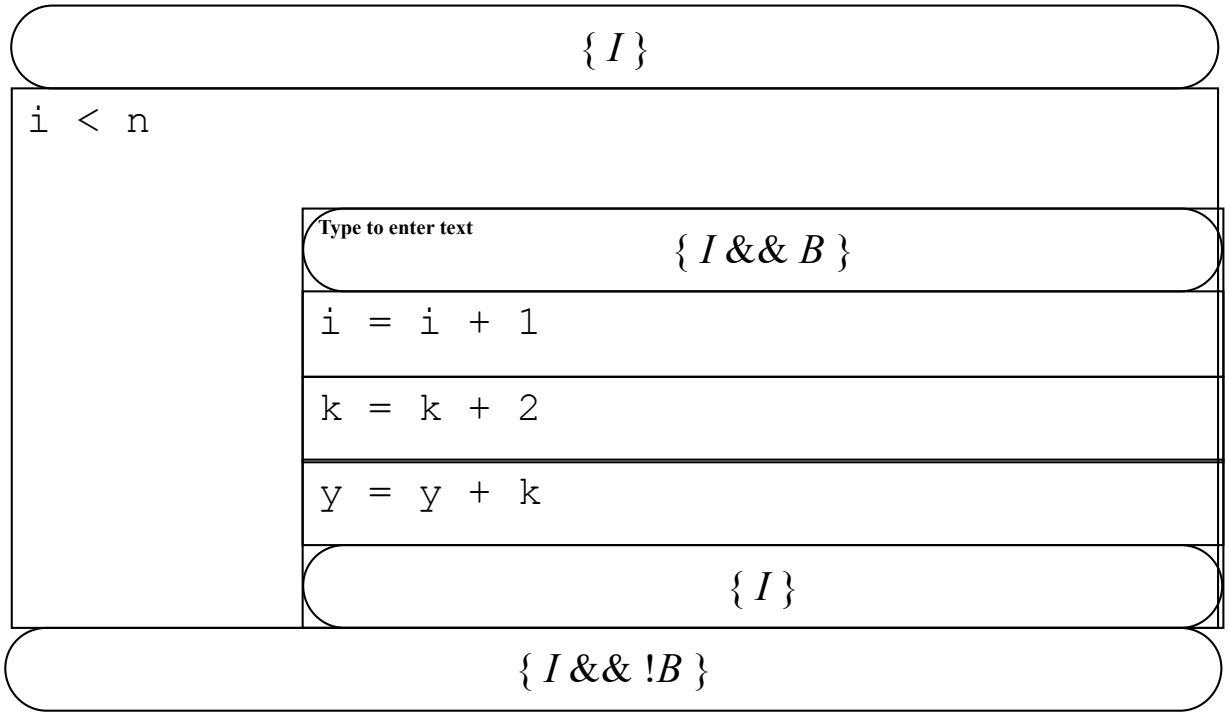


Wir wenden die Beweisregel für While-Schleifen an.

Hoare-Regel für die While-Schleife

$$\frac{\{\ I\ \&\&\ B\ \}\ S\ \{\ I\ \}}{\{\ I\ \}\ \underline{\textbf{while}}\ B\ S\ \{\ I\ \&\&\ !B\ \}}$$

Anwendung der Schleifenregel



6

Was ist die Schleifen-Invariante *I*?

- Ansatz:
  - Wie stehen die Programmvariablen in Beziehung?
  - *n* ist konstant. Wir setzen *n* = 4.

i	*k*	*y*	*n*
0	-1	0	4
1	1	1	4
2	3	4	4
3	5	9	4
4	7	16	4

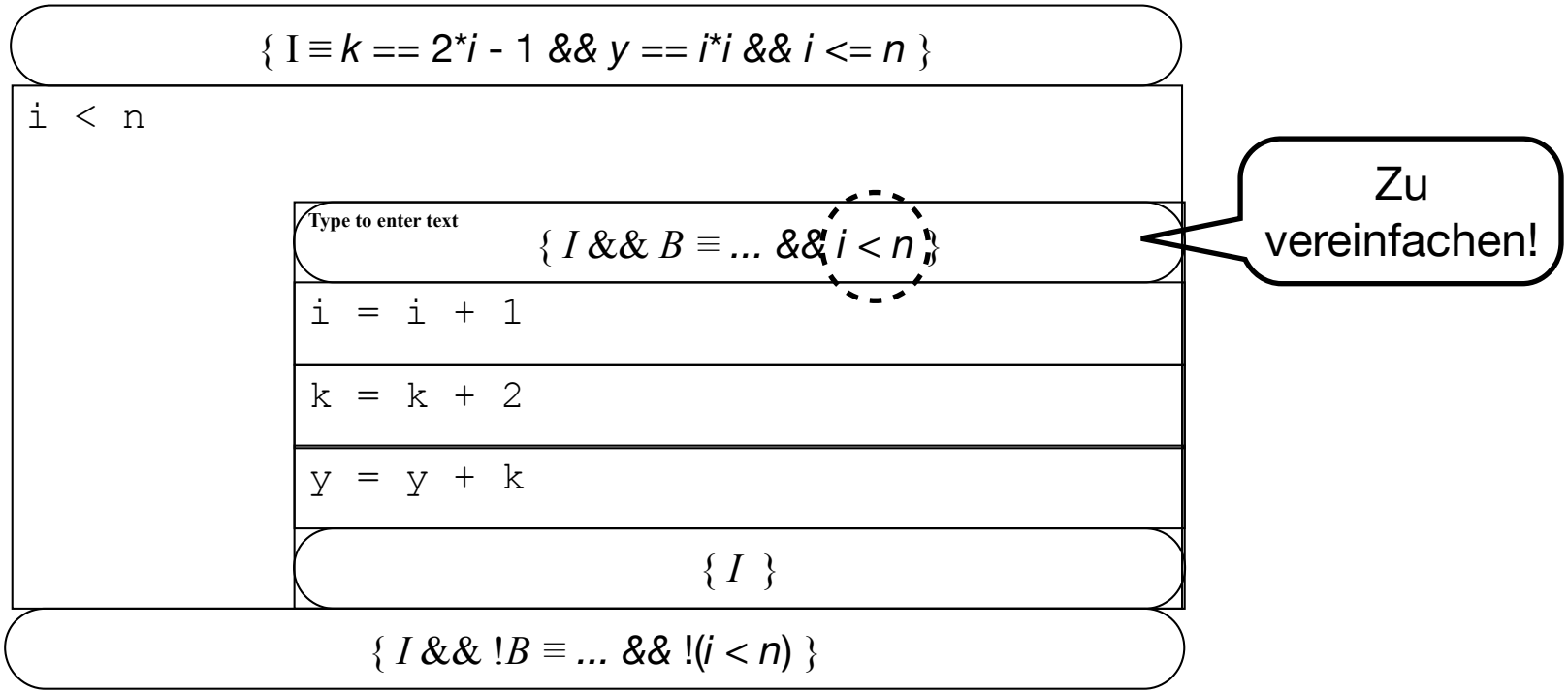
$k == 2*i - 1$

$y == i*i$

$0 <= i <= n$

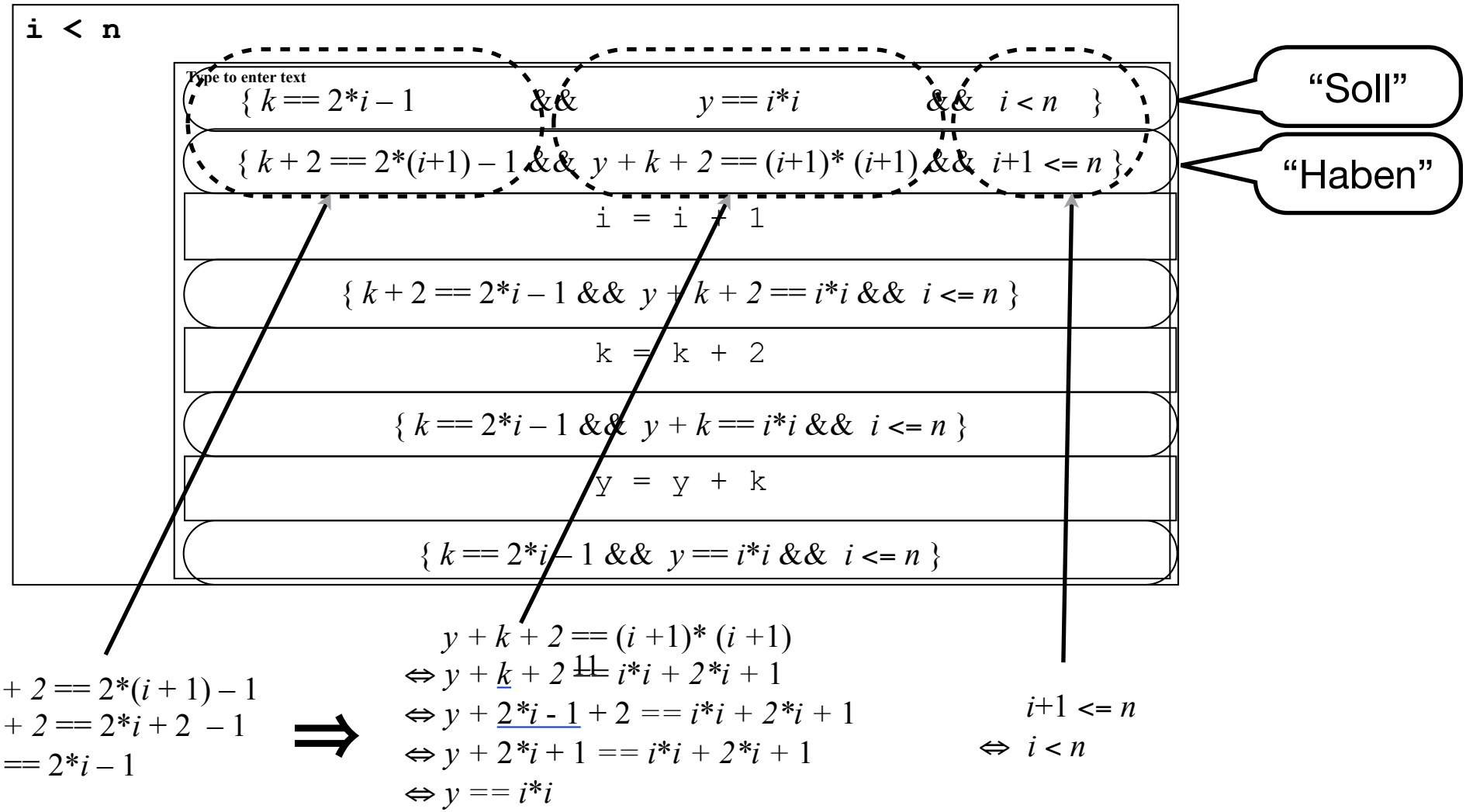
$$I \equiv k == 2*i - 1 \;\&\&\; y == i*i \;\&\&\; i <= n$$

Verifikation der Schleife

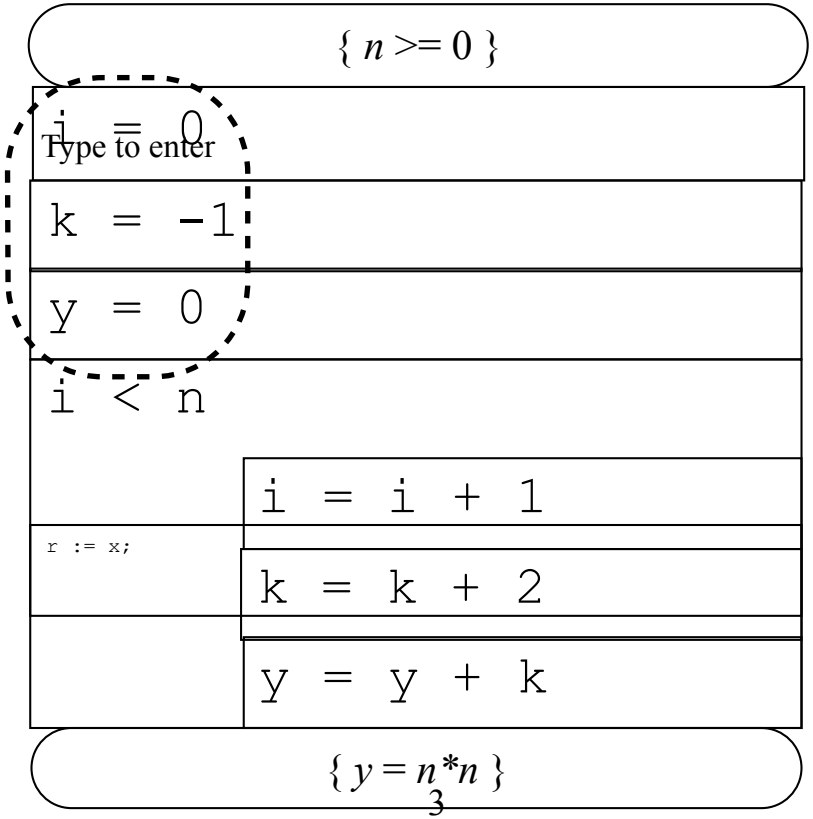


$$i <= n \;\&\&\; i < n \Leftrightarrow i < n$$

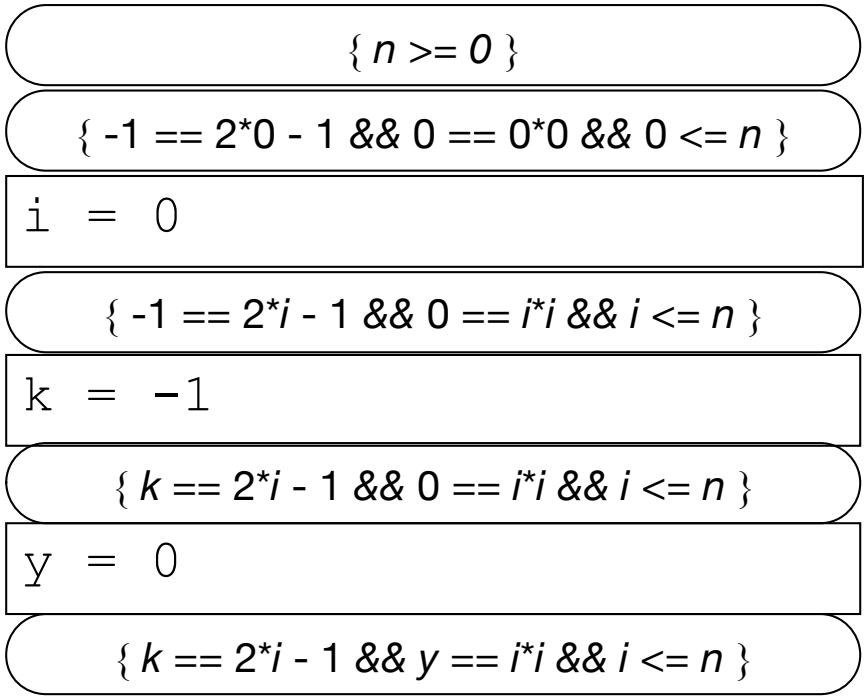
Verifikation des Schleifenkörpers



Restliche Zuweisungen zu verifizieren



Restliche Zuweisungen zu verifizieren



$$-1 == 2*0 - 1 \;\&\&\; 0 == 0*0 \;\&\&\; 0 <= n$$

$$\Leftrightarrow$$

$$n >= 0$$

Eine Übungsaufgabe

- Problem (wie zuvor):
  - Berechnen des Quadrats einer Zahl
  - ohne Verwendung von “*”
  - mit Verwendung von “+”.
- Eine andere Option für ein Lösungsansatz:
  - $n^2 = \underbrace{n + \ldots + n}_{n \text{ mal}}$
- Aufgabe:
  - Programm
  - Spezifikation
  - Korrektheitsbeweis