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:

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

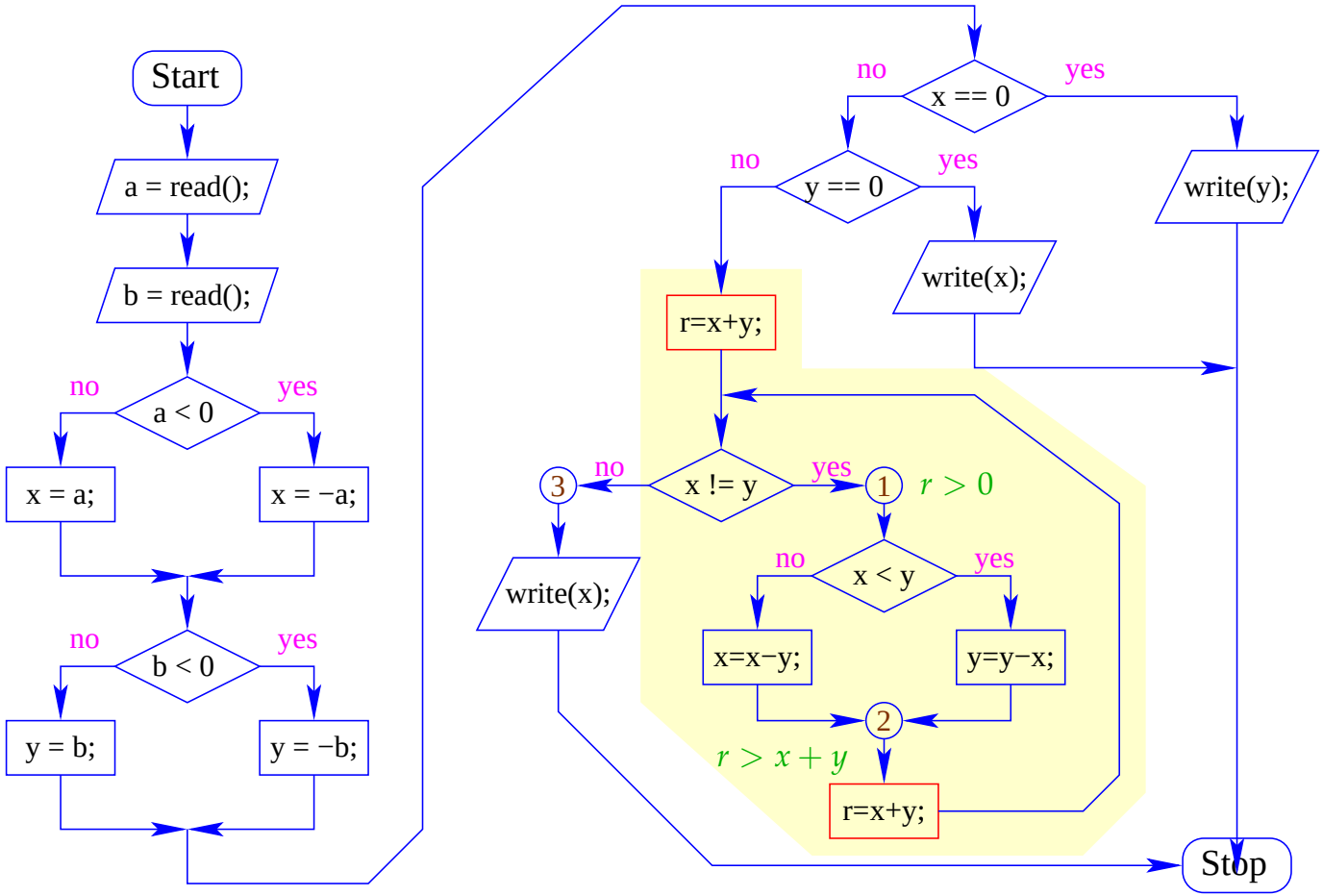
Wir überprüfen:

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

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 :-))

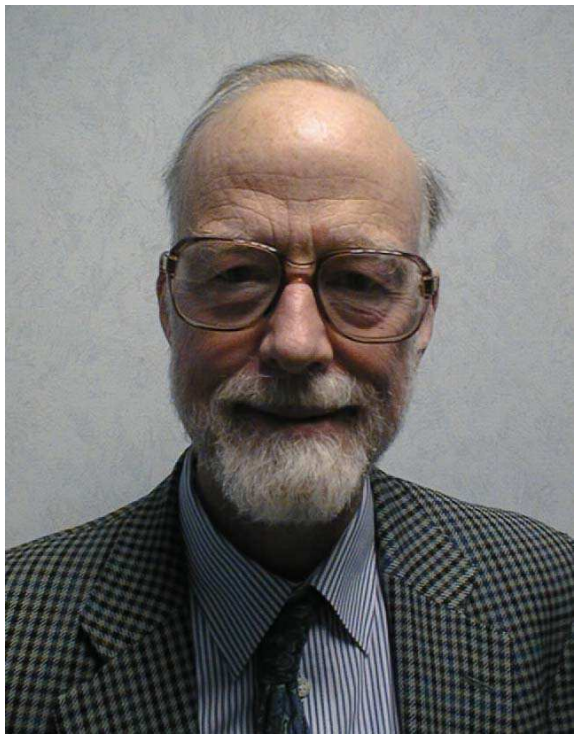
Allgemeines Vorgehen:

- 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` :-)

1.5 Modulare Verification und Prozeduren



Tony Hoare, Microsoft Research, Cambridge

Idee:

- Modularisiere den Korrektheitsbeweis so, dass Teilbeweise für wiederkehrende Aufgaben wiederverwendet werden können.
- 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:

- Modularisiere den Korrektheitsbeweis so, dass Teilbeweise für wiederkehrende Aufgaben wiederverwendet werden können.
- 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:

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

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

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

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

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:

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

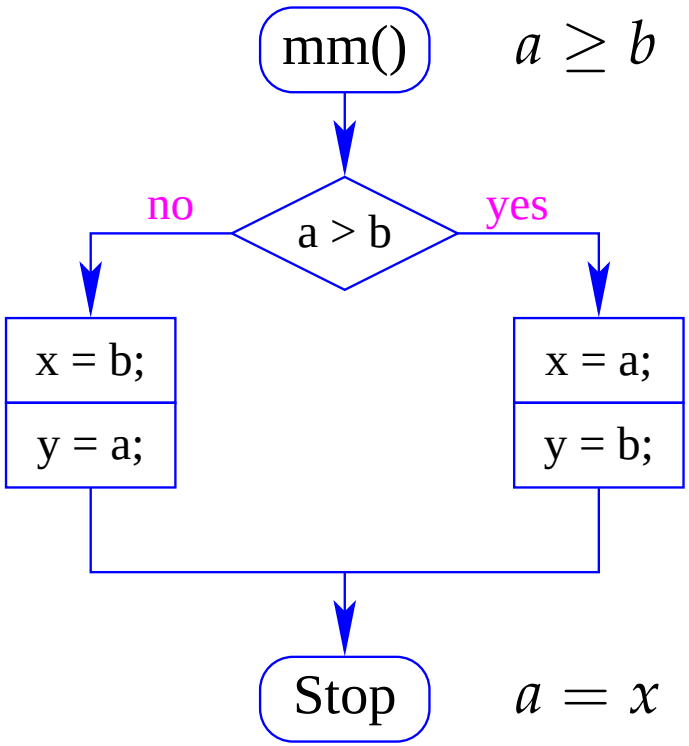
- Unter dieser globalen Hypothese H verifizieren wir, dass sich für jede Prozedurdefinition `void f() { ss }` zeigen lässt:

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

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

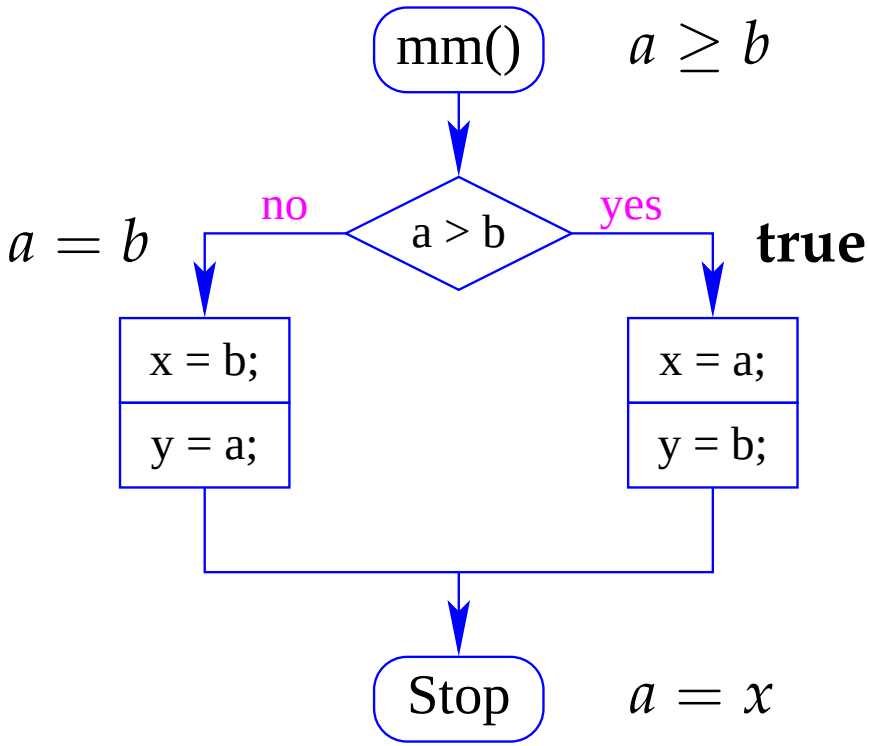
... im Beispiel:

Wir überprüfen:



... im Beispiel:

Wir überprüfen:



Diskussion:

- Die Methode funktioniert auch, wenn die Prozedur einen Rückgabewert hat: den können wir mit einer globalen Variable `return` simulieren, in die das jeweilige Ergebnis geschrieben wird :-)
- Es ist dagegen nicht offensichtlich, wie die Vor- und Nachbedingung für Prozeduraufrufe gewählt werden soll, wenn eine Funktion an mehreren Stellen aufgerufen wird ...
- Noch schwieriger wird es, wenn eine Prozedur rekursiv ist: dann hat sie potentiell unbeschränkt viele verschiedene Aufrufe !?

Beispiel:

```
int x, m0, m1, t;

void main () {
   x = read();
   m0 = 1; m1 = 1;
   if (x > 1) f();
   write (m1);
}
```

```
void f() {
   x = x-1;
   if (x>1) f();
   t = m1;
   m1 = m0+m1;
   m0 = t;
}
```

Kommentar:

- Das Programm liest eine Zahl ein.
- Ist diese Zahl höchstens 1, liefert das Programm 1 ...
- Andernfalls berechnet das Programm die Fibonacci-Funktion fib :-)
- Nach einem Aufruf von `f` enthalten die Variablen `m0` und `m1` jeweils die Werte $\mathrm{fib}(i-1)$ und $\mathrm{fib}(i)$ ...

Problem:

- Wir müssen in der Logik den i-ten vom $(i+1)$-ten Aufruf zu unterscheiden können ;-)
- Das ist einfacher, wenn wir logische Hilfsvariablen $\underline{l} = l_1, \ldots, l_n$ zur Verfügung haben, in denen wir (ausgewählte) Werte vor dem Aufruf retten können ...

Im Beispiel:

$$\{A\}\ \texttt{f();}\ \{B\} \qquad \text{wobei}$$

$$\begin{aligned} A &\equiv x = l \wedge x > 1 \wedge m_0 = m_1 = 1 \\ B &\equiv l > 1 \wedge m_1 \leq 2^l \wedge m_0 \leq 2^{l-1} \end{aligned}$$

Allgemeines Vorgehen:

- Wieder starten wir mit einer globalen Hypothese H, die für jeden Aufruf `f();` eine Beschreibung bereitstellt:

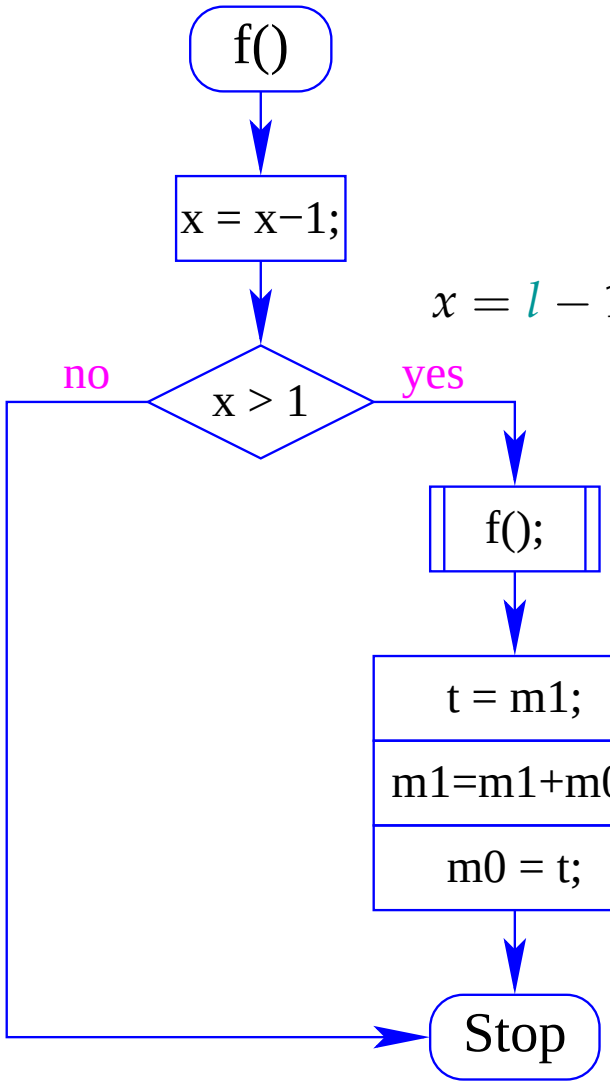
$$\{A\}\ \texttt{f();}\ \{B\}$$

// sowohl A wie B können l_i enthalten :-)

- Unter dieser globalen Hypothese H verifizieren wir, dass für jede Funktionsdefinition `void f() { ss }` gilt:

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

... im Beispiel:



$x = l - 1 \wedge x > 0 \wedge m_0 = m_1 = 1$

A

D

C

B

- Wir starten von der Zusicherung für den Endpunkt:

$$B \;\equiv\; l > 1 \wedge m_1 \leq 2^l \wedge m_0 \leq 2^{l-1}$$

- Die Zusicherung C ermitteln wir mithilfe von $\mathbf{WP}[\![\ldots]\!]$ und Abschwächung ...

$$\begin{aligned}
&\mathbf{WP}[\![\texttt{t=m1; m1=m1+m0; m0=t;}]\!]\,(B)\\
&\qquad\equiv\quad l-1 > 0 \wedge m_1 + m_0 \leq 2^l \wedge m_1 \leq 2^{l-1}\\
&\qquad\Leftarrow\quad l-1 > 1 \wedge m_1 \leq 2^{l-1} \wedge m_0 \leq 2^{l-2}\\
&\qquad\equiv\quad C
\end{aligned}$$

Frage:

Wie nutzen wir unsere globale Hypothese, um einen konkreten Prozeduraufruf zu behandeln ???

Idee:

- Die Aussage $\{A\}$ `f();` $\{B\}$ repräsentiert eine Wertetabelle für `f()` :-)
- Diese Wertetabelle können wir logisch repräsentieren als die Implikation:

$$\forall \underline{l}.\ (A[\underline{h}/\underline{x}] \Rightarrow B)$$

// $\underline{h}$ steht für eine Folge von Hilfsvariablen

Die Werte der Variablen $\underline{x}$ vor dem Aufruf stehen in den Hilfsvariablen :-)

Beispiele:

Funktion:	`void double () { x = 2*x;}`
Spezifikation:	$\{x = l\}$ `double();` $\{x = 2l\}$
Tabelle:	$\forall l.(h = l) \Rightarrow (x = 2l)$
	$\equiv \quad (x = 2h)$

Für unsere Fibonacci-Funktion berechnen wir:

$$\forall\, l.\ \ (h > 1 \wedge h = l \wedge h_0 = h_1 = 1) \ \ \Rightarrow$$
$$l > 1 \wedge m_1 \leq 2^l \wedge m_0 \leq 2^{l-1}$$

$$\equiv \quad (h > 1 \wedge h_0 = h_1 = 1) \ \Rightarrow \ m_1 \leq 2^h \wedge m_0 \leq 2^{h-1}$$

Ein anderes Paar (A_1, B_1) von Zusicherungen liefert ein gültiges Tripel $\{A_1\}$ `f();` $\{B_1\}$, falls wir zeigen können:

$$\frac{\forall\, \underline{l}.\;\; A[\underline{h}/\underline{x}] \Rightarrow B \qquad\qquad A_1[\underline{h}/\underline{x}]}{B_1}$$

Ein anderes Paar (A_1, B_1) von Zusicherungen liefert ein gültiges Tripel $\{A_1\}$ `f();` $\{B_1\}$, falls wir zeigen können:

$$\frac{\forall\, \underline{l}.\ \ A[\underline{h}/\underline{x}] \Rightarrow B \qquad\qquad A_1[\underline{h}/\underline{x}]}{B_1}$$

Beispiel: `double()`

$$\begin{array}{llllll} A & \equiv & x = l & B & \equiv & x = 2l \\ A_1 & \equiv & x \geq 3 & B_1 & \equiv & x \geq 6 \end{array}$$

Ein anderes Paar (A_1, B_1) von Zusicherungen liefert ein gültiges Tripel $\{A_1\}$ `f();` $\{B_1\}$, falls wir zeigen können:

$$\frac{\forall\, \underline{l}.\;\; A[\underline{h}/\underline{x}] \Rightarrow B \qquad\qquad A_1[\underline{h}/\underline{x}]}{B_1}$$

Beispiel: `double()`

$$\begin{array}{lcllcl} A & \equiv & x = l & B & \equiv & x = 2l \\ A_1 & \equiv & x \geq 3 & B_1 & \equiv & x \geq 6 \end{array}$$

Wir überprüfen:

$$\frac{x = 2h \qquad\qquad h \geq 3}{x \geq 6}$$

:-)

Bemerkungen:

Gültige Paare (A_1, B_1) erhalten wir z.B.,

- indem wir die logischen Variablen substituieren:

$$\frac{\{x = l\}\ \texttt{double();}\ \{x = 2l\}}{\{x = l - 1\}\ \texttt{double();}\ \{x = 2(l-1)\}}$$

Bemerkungen:

Gültige Paare (A_1, B_1) erhalten wir z.B.,

- indem wir die logischen Variablen substituieren:

$$\frac{\{x = l\}\ \texttt{double();}\ \{x = 2l\}}{\{x = l - 1\}\ \texttt{double();}\ \{x = 2(l-1)\}}$$

- indem wir eine Bedingung C an die logischen Variablen hinzufügen:

$$\frac{\{x = l\}\ \texttt{double();}\ \{x = 2l\}}{\{x = l \wedge l > 0\}\ \texttt{double();}\ \{x = 2l \wedge l > 0\}}$$

Bemerkungen (Forts.):

Gültige Paare (A_1, B_1) erhalten wir z.B. auch,

- indem wir die Vorbedingung verstärken bzw. die Nachbedingung abschwächen:

$$\frac{\{x = l\}\ \texttt{double();}\ \{x = 2l\}}{\{x > 0 \wedge x = l\}\ \texttt{double();}\ \{x = 2l\}}$$

$$\frac{\{x = l\}\ \texttt{double();}\ \{x = 2l\}}{\{x = l\}\ \texttt{double();}\ \{x = 2l \vee x = -1\}}$$

Anwendung auf Fibonacci:

Wir wollen beweisen: $\{D\}$ `f();` $\{C\}$

$$
\begin{array}{lcl}
A & \equiv & x > 1 \wedge l = x \wedge m_0 = m_1 = 1 \\
A[(l-1)/l] & \equiv & x > 1 \wedge l - 1 = x \wedge m_0 = m_1 = 1 \\
 & \equiv & D
\end{array}
$$

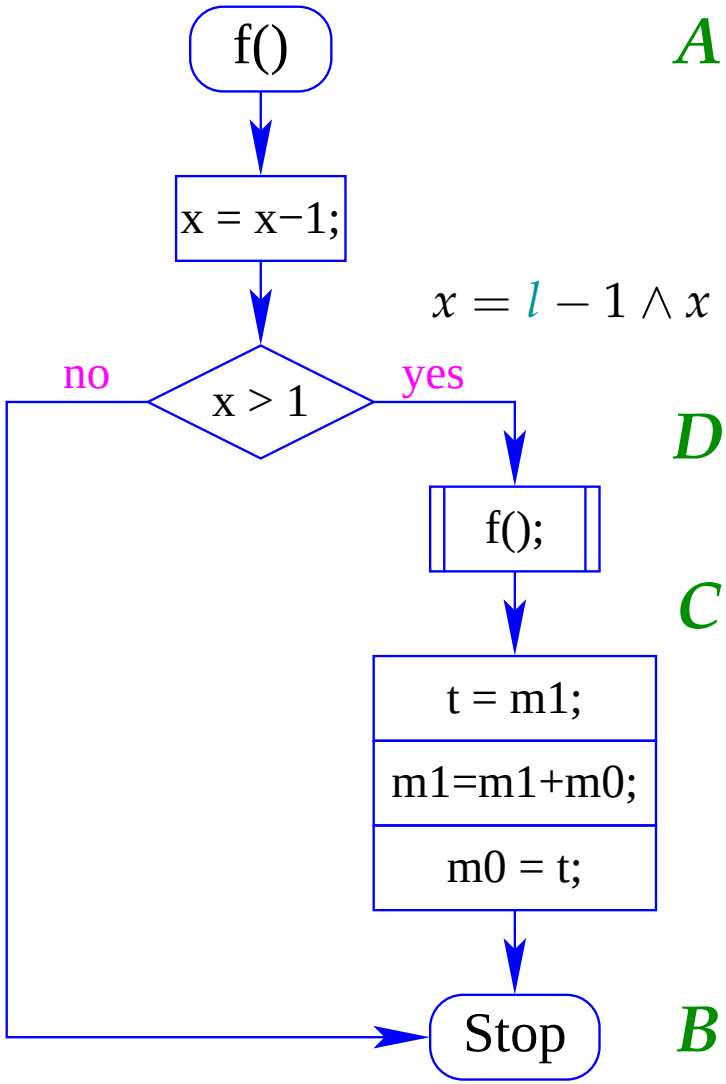
Anwendung auf Fibonacci:

Wir wollen beweisen: $\{D\}$ `f();` $\{C\}$

$$
\begin{array}{lcl}
A & \equiv & x > 1 \wedge l = x \wedge m_0 = m_1 = 1 \\
A[(l-1)/l] & \equiv & x > 1 \wedge l - 1 = x \wedge m_0 = m_1 = 1 \\
 & \equiv & D
\end{array}
$$

$$
\begin{array}{lcll}
B & \equiv & l > 1 \wedge m_1 \leq 2^l \wedge m_0 \leq 2^{l-1} & \\
B[(l-1)/l] & \equiv & l - 1 > 1 \wedge m_1 \leq 2^{l-1} \wedge m_0 \leq 2^{l-2} & \\
 & \equiv & C & \text{:-)}
\end{array}
$$

Orientierung:



$x = l - 1 \wedge x > 0 \wedge m_0 = m_1 = 1$