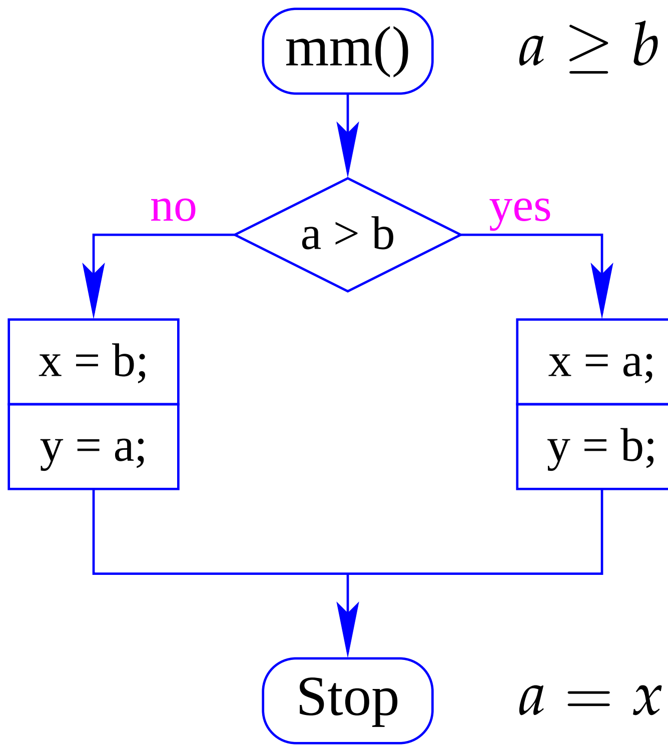
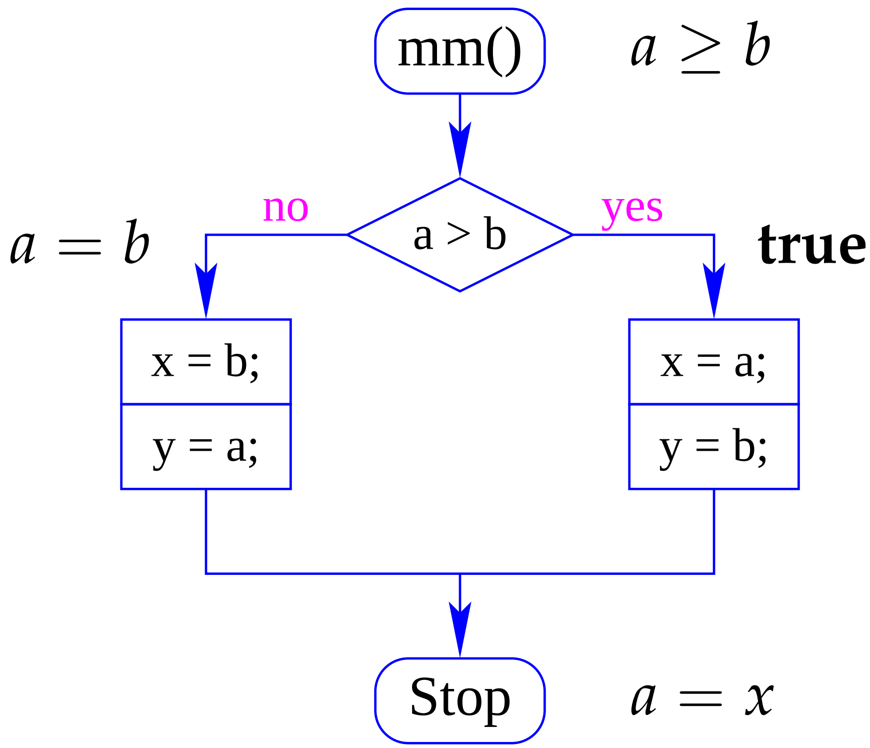
... im Beispiel:

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 $\mathsf{fib}(i-1)$ und $\mathsf{fib}(i)$ ...

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:

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

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

Allgemeines Vorgehen:

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

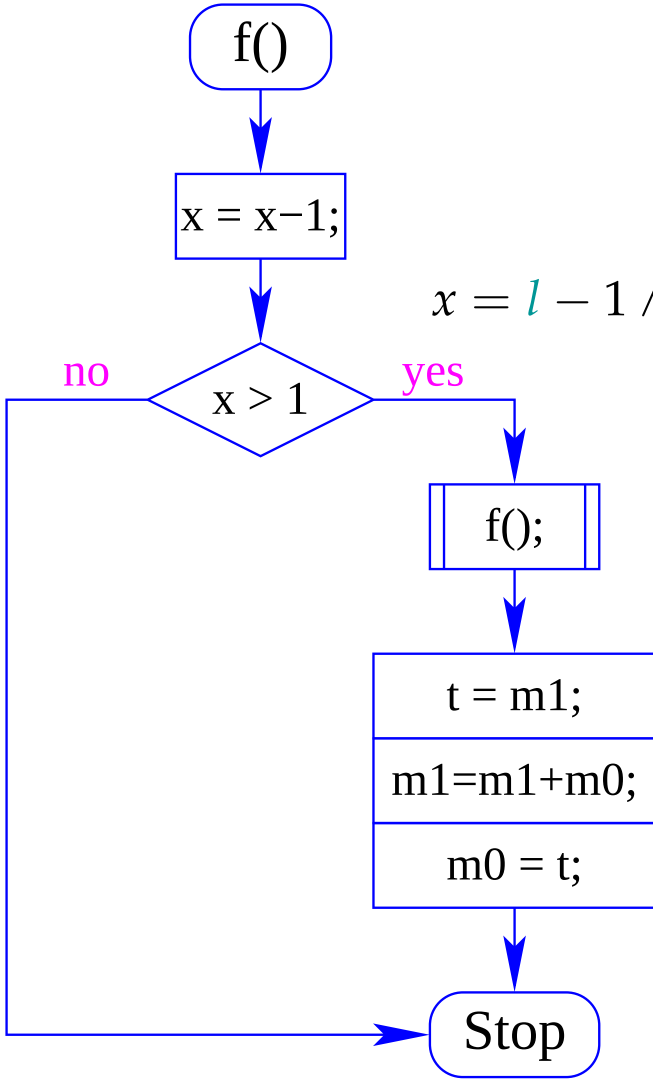
$$\{A\}\ \texttt{f();}\ \{B\}$$

// 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:



A

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

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 ...

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

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:

$$\begin{aligned} & \forall\, l.(h = l) \Rightarrow (x = 2l) \\ \equiv\quad & (x = 2h) \end{aligned}$$

Für unsere Fibonacci-Funktion berechnen wir:

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

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

Ein anderes Paar (A_1, B_1) von Zusicherungen liefert ein gültiges Tripel $\{A_1\}$ `f();` $\{B_1\}$, falls die zugehörige Tabelle implziert wird ...

Regel:

Adaption:

$$\frac{\{A\}\ \texttt{f();}\ \{B\} \in H \qquad (\forall\, \underline{l}.\ A[\underline{h}/\underline{x}] \Rightarrow B) \Rightarrow (A_1[\underline{h}/\underline{x}] \Rightarrow B_1)}{\{A_1\}\ \texttt{f();}\ \{B_1\}}$$

Praktisch müssen wir zeigen:

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

Praktisch müssen wir zeigen:

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

Beispiel:

```
double()
```

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

Praktisch müssen wir zeigen:

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

Beispiel:

```
double()
```

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

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 die Vorbedingung verstärken bzw. die Nachbedingung abschwächen:

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

Anwendung auf Fibonacci:

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

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

Anwendung auf Fibonacci:

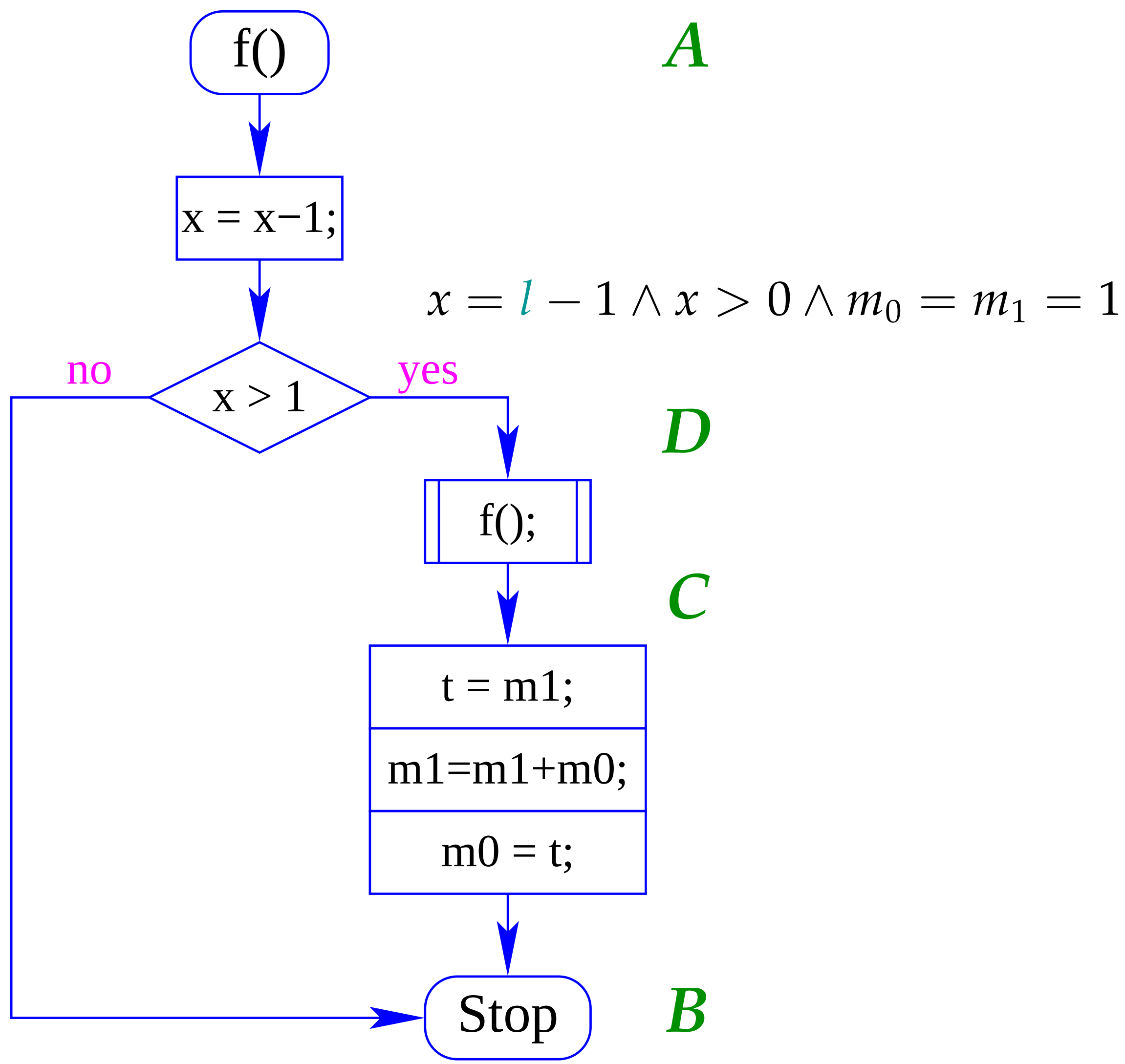
Wir wollen beweisen: $\{D\}$ `f();` $\{C\}$

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

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

:-)

Orientierung:



Für die bedingte Verzweigung verifizieren wir:

$$
\begin{array}{lll}
\mathbf{WP}[\![\texttt{x>1}]\!]\,(B, D) & \equiv & (x \leq 1 \wedge l > 1 \wedge m_1 \leq 2^l \wedge m_0 \leq 2^{l-1}) \;\vee \\
& & (x > 1 \wedge x = l - 1 \wedge m_1 = m_0 = 1) \\
\\
& \Leftarrow & x > 0 \wedge x = l - 1 \wedge m_0 = m_1 = 1
\end{array}
$$

:-))

3.7 Prozeduren mit lokalen Variablen

- Prozeduren `f()` modifizieren globale Variablen.
- Die Werte der lokalen Variablen des Aufrufers vor und nach dem Aufruf sind unverändert :-)

Beispiel:

```
{int y= 17; double(); write(y);}
```

Vor und nach dem Aufruf von `double()` gilt: $y = 17$:-)

- Der Erhaltung der lokalen Variablen tragen wir automatisch Rechnung, wenn wir bei der Adaptionsregel in

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

beachten:

  → Die Vor- und Nachbedingungen: $\{A\}, \{B\}$ für Prozeduren sprechen nur über globale Variablen !

  → Die $\underline{h}$ werden nur für die globalen Variablen eingesetzt !!

- Der Erhaltung der lokalen Variablen tragen wir automatisch Rechnung, wenn wir bei der Adaptionsregel in

$$(\forall \underline{l}.\ A[\underline{h}/\underline{x}] \Rightarrow B) \Rightarrow (A_1[\underline{h}/\underline{x}] \Rightarrow B_1)$$

beachten:

  → Die Vor- und Nachbedingungen: $\{A\}, \{B\}$ für Prozeduren sprechen nur über globale Variablen !

  → Die $\underline{h}$ werden nur für die globalen Variablen eingesetzt !!

- Als neuen Spezialfall der Adaption erhalten wir:

$$\frac{\{A\}\ \texttt{f();}\ \{B\}}{\{A \wedge C\}\ \texttt{f();}\ \{B \wedge C\}}$$

falls C nur über lokale Variablen des Aufrufers spricht :-)

3.8 Prozeduren mit Parametern

- Unsere Behandlung von lokalen Variablen ermöglicht uns, call-by-value Parameter-Übergabe von Java zu simulieren :-)
- Ein formaler Parameter `y` ist eine spezielle lokale Variable :-)
- Für die geregelte Parameter-Übergabe spendieren wir uns für `y` eine zusätzliche globale Variable `Y`.
- Vor dem Aufruf erhält `Y` den aktuellen Wert für `y`.
- Am Anfang des Rumpfs wird `Y` nach `y` kopiert ...

Aus einer Definition: `void f(int y) { ss }`

wird: `void f() {int y=Y; ss }`

Aus einem Aufruf: `f(e);`

wird: `Y=e; f();`

Um diese Transformation nicht syntaktisch ausführen zu müssen, führen wir die entsprechende Transformation nur in den Zusicherungen aus ...

- Als Spezifikation für `f` erlauben wir Tripel:

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

wobei A über globale Variablen und `Y` spricht :-)

- Als Spezifikation für `f` erlauben wir Tripel:

$$\{A\}\ \texttt{f(int y)}\ \{B\}$$

wobei A über globale Variablen und `Y` spricht :-)

- Zur Korrektheit verifizieren wir:

$$\{A \wedge Y = y\}\ \texttt{ss}\ \{B\}$$

- Als Spezifikation für `f` erlauben wir Tripel:

$$\{A\}\ \texttt{f(int y)}\ \{B\}$$

wobei A über globale Variablen und `Y` spricht :-)

- Zur Korrektheit verifizieren wir:

$$\{A \wedge Y = y\}\ \texttt{ss}\ \{B\}$$

- Bei einem Aufruf folgern wir:

$$\frac{\{A\}\ \texttt{f(int y)}\ \{B\} \ \in\ H}{\{A[e/Y]\}\ \texttt{f(e);}\ \{B\}}$$

Beispiel:

```
int ret;

void inc(int y) {
   ret = ret + y;
}
```

```
void main () {
   ret = read();
   inc(ret + 1);
   write (ret);
}
```

Beispiel:

```
int ret;

void inc(int y) {
    ret = ret + y;
}
```

```
void main () {
    ret = read();
    inc(ret + 1);
    write (ret);
}
```

Spezifikation: $\{\mathsf{ret} = l_1 \wedge Y = l_2\}$ `inc(int y)` $\{\mathsf{ret} = l_1 + l_2\}$

Beispiel:

```
int ret;

void inc(int y) {
    ret = ret + y;
}
```

```
void main () {
    ret = read();
    inc(ret + 1);
    write (ret);
}
```

Spezifikation: $\{\text{ret} = l_1 \wedge Y = l_2\}$ `inc(int y)` $\{\text{ret} = l_1 + l_2\}$

Überprüfung: $\{\text{ret} = l_1 \wedge Y = l_2 = y\}$ `ret = ret + y;` $\{\text{ret} = l_1 + l_2\}$

Beispiel:

```
int ret;

void inc(int y) {
    ret = ret + y;
}
```

```
void main () {
    ret = read();
    inc(ret + 1);
    write (ret);
}
```

Spezifikation: $\{\mathsf{ret} = l_1 \wedge Y = l_2\}$ `inc(int y)` $\{\mathsf{ret} = l_1 + l_2\}$

Überprüfung: $\{\mathsf{ret} = l_1 \wedge Y = l_2 = y\}$ `ret = ret + y;` $\{\mathsf{ret} = l_1 + l_2\}$

Anwendung: $\{\mathsf{ret} = l_1 \wedge \mathsf{ret} + 1 = l_2\}$ `inc(ret + 1);` $\{\mathsf{ret} = l_1 + l_2\}$

$\{\mathsf{ret} = l_1 \wedge l_1 + 1 = l_2\}$ `inc(ret + 1);` $\{\mathsf{ret} = 2l_1 + 1\}$