Algorithmen und Datenstrukturen

Wiederholung zu Kongruenzen

Prof. Dr. rer. nat. Jörg Striegnitz

Fachhochschule Aachen
striegnitz@fh-aachen.de

19. April 2024

Der Modulo-Operator nach Knuth

Definition 1.8 (mod - der Modulo-Operator)

Der **Modulo-Operator** mod : $\mathbb{Z} \times \mathbb{Z} - \{0\} \to \mathbb{Z}$ ist definiert durch

$$a \bmod b \quad = \quad a - \left\lfloor \frac{a}{b} \right\rfloor \cdot b$$

Beispiel 1.4 (mod - der Modulo-Operator)

- $9 \bmod 5 \; = \; 4$, denn $9 - \left\lfloor \frac{9}{5} \right\rfloor \cdot 5 = 9 - \lfloor 1.8 \rfloor \cdot 5 = 9 - 1 \cdot 5 = 4$
- $9 \bmod 3 = 0$, denn $9 - \left\lfloor \frac{9}{3} \right\rfloor \cdot 3 = 9 - \lfloor 3 \rfloor \cdot 3 = 9 - 3 \cdot 3 = 0$
- $-16 \bmod 5 = 4$, denn $-16 - \left\lfloor \frac{-16}{5} \right\rfloor \cdot 5 = -16 - \lfloor -3.2 \rfloor \cdot 5 = -16 - (-4) \cdot 5 = 4$
- $16 \bmod -5 = -4$, denn
  $16 - \left\lfloor \frac{16}{-5} \right\rfloor \cdot (-5) = 16 - \lfloor -3.2 \rfloor \cdot (-5) = 16 - (-4) \cdot (-5) = -4$

Erinnerung: $\lfloor x \rfloor = \max \{ z \in \mathbb{Z} \mid z \leq x \}$ und daher $\lfloor -3.2 \rfloor = -4$

Definition von `mod` nicht einheitlich!

! Die Definition des Divisionsrestes ist nicht einheitlich!
Und variiert mit verschiedenen Programmiersprachen

Die Wahl des Restes variiert; z.B.:

- $-16 = -4 \cdot 5$ Rest 4 $\quad$ $16 = 4 \cdot -5$ Rest -4
  z.B. Definition von Donald E. Knuth, Ada, Python
- $-16 = -3 \cdot 5$ Rest -1 $\quad$ $16 = 4 \cdot -5$ Rest 1
  z.B. JAVA, JAVAScript, C++11 (davor: implementierungsabhängig)

Absolutwert des Ergebnisses immer kleiner als Absolutwert des Divisors $\left(\frac{\text{Dividend}}{\text{Divisor}}\right)$.
Aber: Vorzeichen gemäß

- Divisor (Knuth, Ada, Python)
- Dividend (JAVA, JAVASctipt, C++11)

Definition 1.9 (Kongruenz)

Sei $m \in \mathbb{Z} - \{0\}$. Zwei Zahlen $a \in \mathbb{Z}$ und $b \in \mathbb{Z}$ heißen **kongruent modulo** m genau dann, wenn

$$a \bmod m \quad = \quad b \bmod m$$

und wir schreiben

$$a \equiv b \pmod{m}$$

Die Menge

$$[a]_m = \{z \in \mathbb{Z} \mid z \equiv a \bmod m\}$$

nennen wir dann **Kongruenzklasse** (auch **Restklasse**) von a modulo m.

Satz 1 über Kongruenzen - 1

Satz 1.1 (Über Kongruenzen)

Seien $m \in \mathbb{Z} - \{0\}$ *und* $\{a, b\} \subseteq \mathbb{Z}$, *dann gilt*

$$a \equiv b \pmod{m} \quad \Leftrightarrow \quad a - b \text{ ist Vielfaches von } m$$

Beweis.

Wir zeigen beide Richtungen:

- $\Rightarrow$: Gelte $a \equiv b \pmod{m}$, d.h. $a \bmod m = b \bmod m$. Dann folgt mit Definition 1.8 für geeignete $\{l, m\} \subseteq \mathbb{Z}$:

$$\begin{aligned} a - b &= a \bmod m + k \cdot m - (b \bmod m + l \cdot m) \\ &= k \cdot m - l \cdot m \quad (* \text{ wegen } a \equiv b \pmod{m} *) \\ &= (k - l) \cdot m \end{aligned}$$

□

Satz 1 über Kongruenzen - 2

Beweis.

- $\Leftarrow$: Gelte $a - b = k \cdot m$ für ein $k \in \mathbb{Z}$. Falls $a = b$ folgt das Gewünschte sofort, ansonsten:

$$\begin{aligned}
a \bmod m &= a - \left\lfloor \frac{a}{m} \right\rfloor \cdot m \\
&= b + k \cdot m - \left\lfloor \frac{b + k \cdot m}{m} \right\rfloor \cdot m \quad (*\ a = b + k \cdot m\ *) \\
&= b + k \cdot m - \left\lfloor \frac{b}{m} + \frac{k \cdot m}{m} \right\rfloor \cdot m \\
&= b + k \cdot m - \left(\left\lfloor \frac{b}{m} \right\rfloor + k \right) \cdot m \quad (*\ k \in \mathbb{Z}\ *) \\
&= b + k \cdot m - \left\lfloor \frac{b}{m} \right\rfloor \cdot m - k \cdot m \\
&= b - \left\lfloor \frac{b}{m} \right\rfloor \cdot m \\
&= b \bmod m
\end{aligned}$$

$\square$

Satz 2 über Kongruenzen

Satz 1.2

Seien $m \in \mathbb{Z} - \{0\}$ *und* $\{a, b\} \subseteq \mathbb{Z}$, *dann gilt*

$$a - b \text{ ist Vielfaches von } m \quad \Leftrightarrow \quad \left\lfloor \frac{a-b}{m} \right\rfloor = \left\lfloor \left\lfloor \frac{a}{m} \right\rfloor - \left\lfloor \frac{b}{m} \right\rfloor \right\rfloor$$

Beweis.

$\Rightarrow$: ergibt sich sofort mit der Definition der Modulo-Operation und Satz 1.1:

$$\left\lfloor \frac{\left\lfloor \frac{a}{m} \right\rfloor \cdot m + a \bmod m}{m} - \frac{\left\lfloor \frac{b}{m} \right\rfloor \cdot m + b \bmod m}{m} \right\rfloor \underset{a \equiv b \pmod m}{=} \left\lfloor \left\lfloor \frac{a}{m} \right\rfloor - \left\lfloor \frac{b}{m} \right\rfloor \right\rfloor$$

$\Leftarrow$:

$$\begin{aligned} \left\lfloor \frac{a}{m} - \frac{b}{m} \right\rfloor &= \left\lfloor \frac{\left\lfloor \frac{a}{m} \right\rfloor \cdot m + a \bmod m}{m} - \frac{\left\lfloor \frac{b}{m} \right\rfloor \cdot m + b \bmod m}{m} \right\rfloor \\ &= \left\lfloor \left\lfloor \frac{a}{m} \right\rfloor + \frac{a \bmod m}{m} - \left\lfloor \frac{b}{m} \right\rfloor - \frac{b \bmod m}{m} \right\rfloor \end{aligned}$$

$\square$

Algorithmen und Datenstrukturen

Algorithmische Analyse mit der RAM

Prof. Dr. rer. nat. Jörg Striegnitz

Fachhochschule Aachen
striegnitz@fh-aachen.de

19. April 2024

Analyse von Algorithmen (Wdh.)

- RAM bietet elementaren Operationen und (darauf aufbauend) klar definierte **Semantik**
- **Damit**: Fundierte Analyse von Algorithmen möglich

Zentrale Fragen der Analyse von Algorithmen

- **Korrektheit**: berechnet der Algorithmus das gewünschte?
- **Termination**: terminiert der Algorithmus immer?
- **Geschwindigkeit**: wie lange läuft der Algorithmus?
- **Speicherverbrauch**: wie viel Speicher verbraucht der Algorithmus?

- Die ersten beiden Aspekte werden nicht (immer) im Detail behandelt.
- Geschwindigkeit und Speicherverbrauch
  - relative Angaben
    z.B. Zahl der RAM-Instruktionen, Anzahl der RAM-Speicherzellen
  - in Abhängigkeit von der Größe der Eingabedaten
    z.B. Anzahl Daten, Größe der Zahlen etc.

Korrektheit: Die Behauptung ...

Behauptung: rechts stehendes RAM-Programm berechnet die Funktion

$$f(A,B) = A - \left\lfloor \frac{A}{B} \right\rfloor \cdot B$$

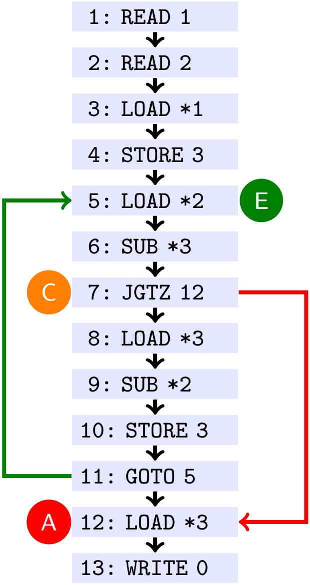
bzw. $f(A,B) = A \bmod B$:

$$(1,(A,B),(),\sigma_0) \overset{*}{\vdash} (0,(),(A \bmod B),\sigma)$$

- A und B stehen auf dem Eingabeband
- die Schrittrelation endet in Zeile 14 (implizit hinzugefügtes `HALT`)
- auf dem Ausgabeband steht am Ende der Wert $A \bmod B$

```
 1: READ 1      a ≅ σ(1)
 2: READ 2      b ≅ σ(2)
 3: LOAD *1     a→Akku
 4: STORE 3     x ≅ σ(3) = a
 5: LOAD *2     b→Akku
 6: SUB *3      b-x→Akku
 7: JGTZ 12     (b-x)>0?
 8: LOAD *3     x→Akku
 9: SUB *2      x-b→Akku
10: STORE 3     x=x-b
11: GOTO 5      LOOP
12: LOAD *3     x->Akku
13: WRITE 0     x auf Band
```

! Einfaches "Berechnen"der Schrittrelation $\vdash$ mit den Variablen A und B wegen Schleife **nicht trivial**! Versuchen wir es ...



Links sehen Sie unser Programm als **Flussgraph**

so erkennt man die Schleife besser

Die RAM startet in der Konfiguration

$$(1, (A, B), (), \sigma_0)$$

Vor Eintritt in die Schleife (vor **E**):

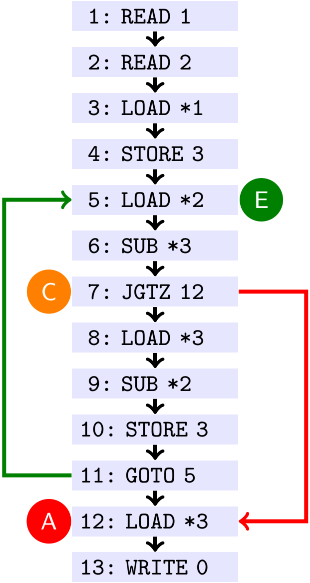
$$(5, (), (), \sigma_0[\underbrace{0 \mapsto A, 1 \mapsto A, 2 \mapsto B, 3 \mapsto A}_{\text{sortiert nach Adresse}}])$$

Vor Test der Abbruchbedingung (**C**):

$$(7, (), (), \sigma_0[0 \mapsto B - A, 1 \mapsto A, 2 \mapsto B, 3 \mapsto A])$$

Da wir, abgesehen von den Vorbedingungen $A \geq 0$ und $B > 0$, keine weitere Information zu A und B haben, können wir **nicht so einfach *„zu Ende rechnen"***.

Korrektheit: Probleme mit Schleifen - 2



Schleife wird wiederholt durchlaufen

Vor 1. Test der Abbruchbedingung (**C**):

$$(7, (), (), \sigma_0[0 \mapsto \mathsf{B} - \mathsf{A}, 1 \mapsto A, 2 \mapsto B, 3 \mapsto A])$$

Vor **2. Schleifendurchlauf** (vor **E**):

$$(5, (), (), \sigma_0[0 \mapsto \mathsf{A} - \mathsf{B}, 1 \mapsto A, 2 \mapsto B, 3 \mapsto \mathsf{A} - \mathsf{B}])$$

Beachte:

- Zeilen 8-11 wurden abgearbeitet
- Speicherzellen 1 und 2 bleiben unverändert

Vor **3. Schleifendurchlauf** (vor **E**):

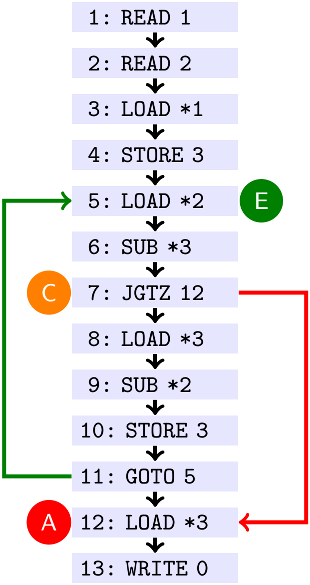
$$(5, (), (), \sigma_0[0 \mapsto \mathsf{A} - \mathsf{B} - \mathsf{B}, \cdots, 3 \mapsto \mathsf{A} - \mathsf{B} - \mathsf{B}])$$

Vor *n*-**tem Schleifendurchlauf** (vor **E**):

$$(5, (), (), \sigma_0[0 \mapsto \omega, \cdots, 3 \mapsto \omega])$$

mit $\omega = \mathsf{A} - (\mathsf{n} - 1) \cdot \mathsf{B}$

Korrektheit: Probleme mit Schleifen - 3



Schleife wird im *n*-ten Durchlauf verlassen:

Vor *n*. Eintritt in die Schleife (vor **E**):

$$(5, (), (), \sigma_0[\ldots, 2 \mapsto B, \ldots, 3 \mapsto A - (n-1) \cdot B])$$

Vor Test der Abbruchbedingung (**C**):

$$(7, (), (), \sigma_0[0 \mapsto \mathsf{B} - (\mathsf{A} - (\mathsf{n} - 1) \cdot \mathsf{B}), \cdots])$$

Annahme: $A - (n-1) \cdot B < B$ ist gegeben $\Rightarrow$ Abbruch

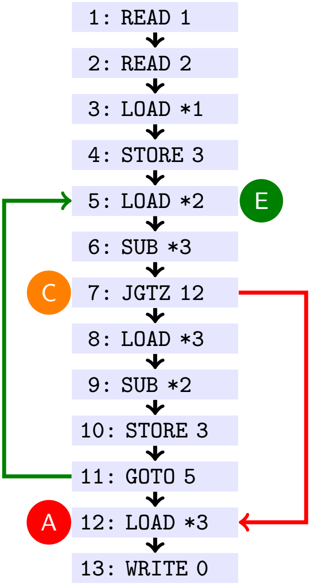
Bei Austritt aus Schleife (**A**):

$$(12, (), (), \sigma_0[\cdots, 3 \mapsto \mathsf{A} - (\mathsf{n} - 1) \cdot \mathsf{B}])$$

Somit **terminiert** die RAM in der Konfiguration:

$$(0, (), (\mathsf{A} - (\mathsf{n} - 1) \cdot \mathsf{B}), \sigma_0[\cdots])$$

Problem: wie groß wird n?



Endkonfiguration war:

$$(0, (), (A - (n-1) \cdot B), \sigma_0[\cdots])$$

In unserem Fall kann man sich leicht überlegen, dass n die kleinste natürliche Zahl ist, so dass

$$\begin{aligned}
& A-(n-1)\cdot B &<& \; B \\
\Leftrightarrow\quad & \tfrac{A}{B}-(n-1) &<& \; 1 \\
\Leftrightarrow\quad & n &>& \; \tfrac{A}{B} \\
\Leftrightarrow\quad & n &=& \; \left\lfloor \tfrac{A}{B} \right\rfloor + 1 \\
& & & \; /{*}\, n \in \mathbb{N} {*}/ \\
\Leftrightarrow\quad & (n-1) &=& \; \left\lfloor \tfrac{A}{B} \right\rfloor
\end{aligned}$$

Damit ergibt sich sofort, dass der Algorithmus gem. der Spezifikation arbeitet.

Alternativen

- „Durchrechnen" für einfache Algorithmen ggf. noch möglich
- Für komplexe Algorithmen (z.B. viele Fallunterscheidungen, geschachtelten Schleifen) schwer handhabbar

Besser: Ansatz der unabhängig von Anzahl der Schleifendurchläufe ist.

Idee:

- Verzahne Ausdruck der Prädikatenlogik mit aktueller Konfiguration
  Ausdruck darf Komponenten der Konfiguration als Konstanten nutzen
- Für jede Schleife:
  - suche Ausdruck, der unabhängig von der Zahl der Durchläufe gilt **und**
  - gemeinsam mit der Abbruchbedingung das Gewünschte zeigen lässt.

  ein solcher Ausdruck heißt **Schleifeninvariante**.

Schleifeninvariante für `mod` - 1

Zur besseren Lesbarkeit setzen wir

- $\alpha = \sigma(0)$,
- $a = \sigma(1)$,
- $b = \sigma(2)$ und
- $x = \sigma(3)$.

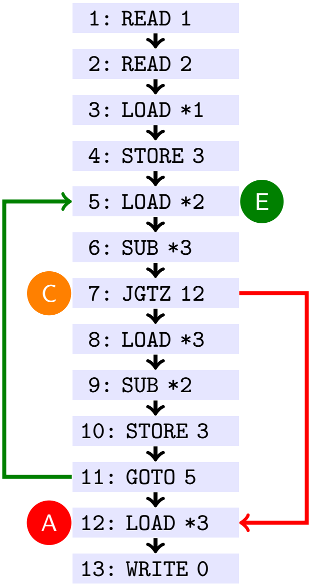
Behauptung: Der Ausdruck

$$\eta \;:=\; x = a - \left\lfloor \frac{a-x}{b} \right\rfloor \cdot b \;\wedge\; a = A \;\wedge\; b = B$$

ist eine **Schleifeninvariante** ist; d.h. η gilt

- zu Beginn einer jeden Iteration (**Zeile 5**)
- am Ende der Schleife (**Zeile 12**)

! RAM-Befehle nehmen Einfluss auf Schleifeninvariante!



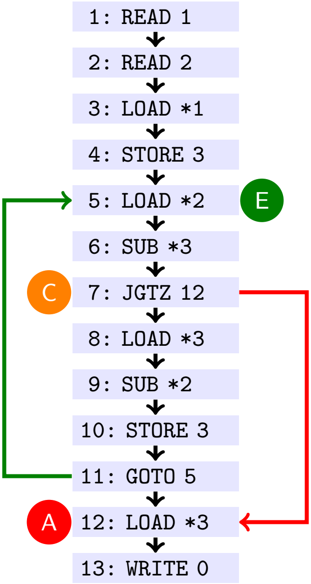
Schleifeninvariante für `mod` - 2

Unsere vermutete Schleifeninvariante:

$$\eta \;:=\; a = \left\lfloor \frac{a-x}{b} \right\rfloor \cdot b + x \;\wedge\; a = A \;\wedge b = B$$

- A und B sind **gültige** Eingabeparameter
  vom Eingabeband mit $A \geq 0$ und $B > 0$
- Term $a = A \;\wedge\; b = B$ lassen wir aus
  Aussage trivial, da die Speicherzellen für a ($\sigma(1)$) und b ($\sigma(2)$) nach Zeile 4 nicht mehr geändert werden -
  ▸ Korrektheit: Probleme mit Schleifen - 2

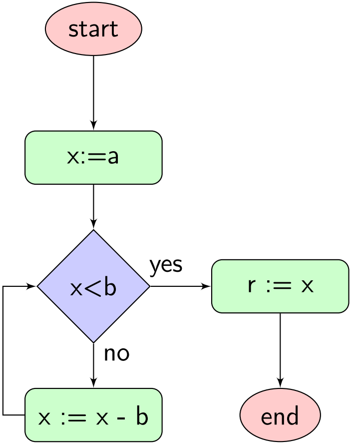
! Wie müssen sicherstellen, dass RAM-Instruktionen die Invariante tatsächlich erhalten - **für jeden denkbaren Programmlauf!**



Zunächst ein **Test** als „erste Rechtfertigung" für unsere Invariante.

Betrachten wir den Programmlauf am Beispiel $f(10, 3)$:

a	b	x	$\left\lfloor \frac{a-x}{b} \right\rfloor$	$x = a - \left\lfloor \frac{a-x}{b} \right\rfloor \cdot b$
10	3	10	0	*true*
10	3	7	1	*true*
10	3	4	2	*true*
10	3	1	3	*true*



! Das ist natürlich noch **kein Beweis!**

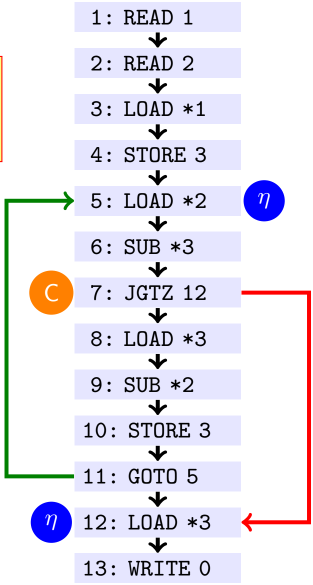
Beweis der Schleifeninvariante

Zu zeigen: egal wie der Graph durchlaufen wird, an den markierten Stellen gilt immer die Schleifeninvariante η!

Wir beweisen per **Induktion** über die Länge einer Berechnung (die Länge eines Pfades durch den rechten Graphen):

- **Induktionsanfang**: Invariante gilt bei erstem Erreichen von Zeile 5
- **Induktionsschritt**: Wir setzen voraus, dass η in Zeile 5 gilt und zweigen, dass die Pfade
  - $5 \to 6 \to 7 \to 8 \to 9 \to 10 \to 11 \to 5$ und
  - $5 \to 6 \to 7 \to 12$

  in Knoten Enden, in denen die Invariante gilt.



Beweis der Schleifeninvariante: Induktionsanfang

Unsere Schleifeninvariante:

$$\eta \;:=\; x = a - \left\lfloor \frac{a-x}{b} \right\rfloor \cdot b \;\wedge\; a = A \;\wedge\; b = B$$

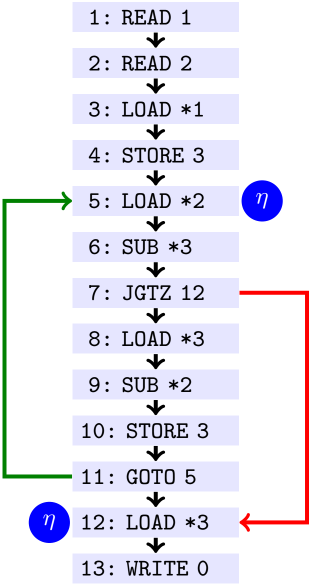
gilt vor erstem Eintritt in die Schleife (Zeile 5):

A und B auf Eingabeband; Zeilen 1-4 führen zur Speicherbelegung

$$\sigma_0[\underbrace{0 \mapsto A}_{\alpha},\ \underbrace{1 \mapsto A}_{a},\ \underbrace{2 \mapsto B}_{b},\ \underbrace{3 \mapsto A}_{x}]$$

Einsetzen in **Invariante** η:

$$\begin{aligned} & A = A - \left\lfloor \tfrac{A-A}{B} \right\rfloor \cdot B \;\wedge\; A = A \;\wedge\; B = B \\ \Leftrightarrow\quad & A = A \;\wedge\; A = A \;\wedge\; B = B \\ \Leftrightarrow\quad & \mathit{true} \end{aligned}$$



Beweis der Schleifeninvariante: Induktionsschritt - 1

Voraussetzung: Invariante η gilt in Zeile 5.

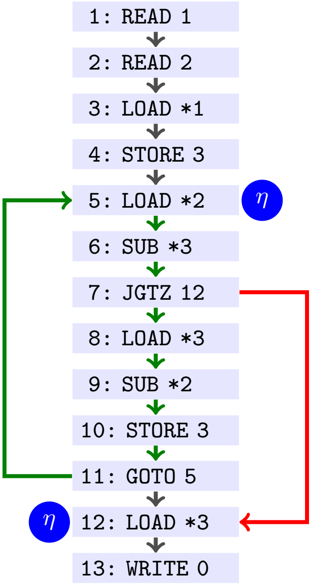
Jetzt müssen wir zeigen, dass die Invariante

1. durch die Iteration erhalten bleibt
   JGTZ$_2$ wird angewendet - **grüner** Pfad
2. m bei Verlassen der Schleife gilt
   JGTZ wird angewendet - **roter** Pfad

Wir betrachten zuerst den **2. Fall**:

Es wirken nur Zeilen 5 und 6 auf den Speicher: nur der Akkumulator α wird geändert $\Rightarrow$ η gilt trivialerweise in Zeile 12

$$\eta \;=\; x = a - \left\lfloor \frac{a-x}{b} \right\rfloor \cdot b$$



Beweis der Schleifeninvariante: Induktionsschritt - 2

Voraussetzung: Invariante η gilt in Zeile 5.

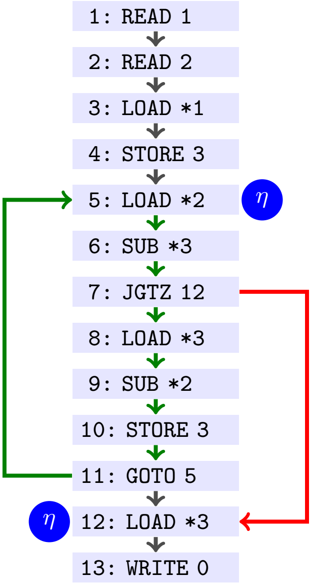
Wir betrachten den **1. Fall**: es wirken Zeilen 5-11 auf den Speicher und wir erhalten durch Auswertung der Schrittrelation $\vdash$ die Belegung

$$\alpha = x - b \quad a = A \quad b = B \quad x = x - B$$

Für η ergibt sich:

$$\begin{array}{rrcl} & x - B & = & A - \left\lfloor \frac{A-(x-B)}{B} \right\rfloor \cdot B \\ \Leftrightarrow & x - B & = & A - \left\lfloor \frac{A-x}{B} + \frac{B}{B} \right\rfloor \cdot B \\ \Leftrightarrow & x - B & = & A - \left(\left\lfloor \frac{A-x}{B} \right\rfloor \cdot B + B \right) \\ \Leftrightarrow & x & = & A - \left\lfloor \frac{A-x}{B} \right\rfloor \cdot B \end{array}$$

Laut Voraussetzung gilt diese Aussage.



Partielle Korrektheit des Algorithmus `mod`

In Zeile 12 gelten

- $A \geq 0$ und $B > 0$ (Vorbedingungen aus der Aufgabenstellung)
- die Invariante η (gerade gezeigt)
- $0 \leq x < B$, klar: sonst wäre der rote Pfad nicht durchlaufen worden

Zu zeigen ist

$$0 \leq x < B \ \wedge \ x = A - \left\lfloor \frac{A-x}{B} \right\rfloor \cdot B \ \Rightarrow \ x = A \bmod B = A - \left\lfloor \frac{A}{B} \right\rfloor \cdot B$$

im Wesentlichen ist also zu zeigen, dass

$$0 \leq x < B \ \Rightarrow \left\lfloor \frac{A-x}{B} \right\rfloor = \left\lfloor \frac{A}{B} \right\rfloor$$

gilt, was aber trivial ist, denn

$$\left\lfloor \frac{A-x}{B} \right\rfloor = \left\lfloor \frac{A}{B} - \frac{x}{B} \right\rfloor \underset{*}{=} \left\lfloor \left\lfloor \frac{A}{B} \right\rfloor - \left\lfloor \frac{x}{B} \right\rfloor \right\rfloor \underset{0 \leq x < B}{=} \left\lfloor \left\lfloor \frac{A}{B} \right\rfloor - 0 \right\rfloor = \left\lfloor \frac{A}{B} \right\rfloor$$

Partielle Korrektheit?

Definition 1.10 (Partielle Korrektheit)

Seien P eine Vorbedingung und Q eine Nachbedingung. Ein Algorithmus heißt **partiell korrekt**, wenn er für Eingaben, unter denen P erfüllt ist, nur Ausgaben liefert, welche Q erfüllen.

Nicht verlangt: Algorithmus **terminiert** für jede gültige Eingabe!

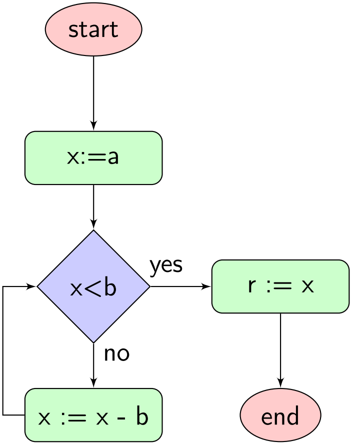
Definition 1.11 (Totale Korrektheit)

Seien P eine Vorbedingung und Q eine Nachbedingung. Ein Algorithmus heißt **total korrekt**, falls er partiell korrekt ist und auf jeder Eingabe, die P erfüllt, terminiert.

Totale Korrektheit des Algorithmus `mod`

In unserem Beispiel trivial:

- Eingangs:
  - $a \geq 0$ und $b > 0$ - Werte bleiben konstant
  - x wird mit a initialisiert $\Rightarrow x \geq 0$.
- Zwei Möglichkeiten:
  - $x < b$: Schleife wird nicht durchlaufen - der Algorithmus terminiert
  - $x \geq b$: Schleife wird durchlaufen, x wird um b echt verkleinert



Fazit: nach $\left\lfloor \frac{A}{B} \right\rfloor$ Schleifendurchläufen terminiert der Algorithmus

Schleifenvariante

Allgemein zum Nachweis der Termination: **Schleifenvariante**

- Ausdruck über Programmvariablen (wie Schleifen**in**variante)
- liefert Zahl aus $\mathbb{N}_0$ (Schleifen**in**variante: aus $\mathbb{B}$)
  - nach unten durch 0 beschränkt
  - muss in jeder Iteration verringert werden

In unserem Beispiel ist x geeignete **Schleifenvariante**.

Speicherplatzverbrauch von `mod`

Speicherplatzverbrauch M wird in Abhängigkeit zur Größe der Eingabe angegeben.

Unsere RAM: Speicherzelle nimmt beliebig große Zahl auf

- Speicherbedarf von `mod`: $M(A, B) = 3$ (Platz für a,b und x)

Realistischer: logarithmisches Kostenmaß (z.B. Anzahl der Bits)

- $M(A, B) = 2 \cdot \lceil \log_2 A \rceil + \lceil \log_2 B \rceil$

Laufzeit von `mod` - 1

Laufzeit T ist abhängig von Größe der Eingabe und entspricht Anzahl der abgearbeiteten RAM-Kommandos:

$$T(A,B) = 4 + 3 + \underbrace{\left\lfloor \frac{A}{B} \right\rfloor \cdot 7}_{\text{Schleife}} + 3$$

Dabei:

- 4 Instruktionen zur Initialisierung
- 3 Instruktionen für das Abbruchkriterium
- 7 Instruktionen im Rumpf der Schleife
- 3 Instruktionen nach Verlassen der Schleife

```
 1: READ 1      a ≅ σ(1)
 2: READ 2      b ≅ σ(2)
 3: LOAD *1     a→Akku
 4: STORE 3     x ≅ σ(3) = a
 5: LOAD *2     b→Akku
 6: SUB *3      b-x→Akku
 7: JGTZ 12     (b-x)>0?
 8: LOAD *3     x→Akku
 9: SUB *2      x-b→Akku
10: STORE 3     x=x-b
11: GOTO 5      LOOP
12: LOAD *3     x->Akku
13: WRITE 0     x auf Band
14: HALT        Done.
```

Laufzeit von `mod` - 2

Realistischer: logarithmisches Kostenmaß für arithmetische Operationen (vereinfachend hier nur am Beispiel der Subtraktion `x-b` in Zeile 9).

Wichtig: $T_{\mathrm{SUB}}(A, B)$ beschreibt den Aufwand für **alle** Subtraktionen in Zeile 9!

- Aufwand Subtraktion: Anzahl Bits des größeren Operanden
- Dann:
  $$T(A, B) = 7 + \left\lfloor \frac{A}{B} \right\rfloor \cdot 6 + \underbrace{T_{\mathrm{SUB}}(A, B)}_{\text{Alle Subtraktionen}} + 3$$
  mit

$$T_{\mathrm{SUB}}(x, y) = \begin{cases} 0 & x < y \\ T_{\mathrm{SUB}}(x - y, y) + \max\left(\lceil \log_2 x \rceil, \lceil \log_2 y \rceil\right) & \text{sonst} \end{cases}$$

```
 1: READ 1     a ≅ σ(1)
 2: READ 2     b ≅ σ(2)
 3: LOAD *1    a→Akku
 4: STORE 3    x ≅ σ(3) = a
 5: LOAD *2    b→Akku
 6: SUB *3     b-x→Akku
 7: JGTZ 12    (b-x)>0?
 8: LOAD *3    x→Akku
 9: SUB *2     x-b→Akku
10: STORE 3    x=x-b
11: GOTO 5     LOOP
12: LOAD *3    x->Akku
13: WRITE 0    x auf Band
14: HALT       Done.
```

Zusammenfassung und Ausblick 1

- RAM ist formales Modell einer einfachen Registermaschine
- Erlaubt systematische Analyse von Algorithmen

  Korrektheit, Laufzeit, Speicherplatzverbrauch

- **Kritik**:
  - Maschinensprache „unhandlich"

    insbesondere: "unstrukturiertes `GOTO`"
    - besser: einfache "`while`"-Sprachen mit ggf. Übersetzungsfunktion zur RAM
    - dann: Verifikation z.B. mit Hoare Kalkül (hier nicht behandelt)
  - Praxisfern

    z.B. Speicheraufbau, Befehlssatz

    Wie viel **Vereinfachung** lassen wir zu?

- Weitere Modelle
  - RASP - (Random-Access Stored-Program Machine) - Programm kann manipuliert werden

    Pendant zum von Neumann-Modell
  - PRAM - (Parallel-RAM) - Analyse von parallelen Algorithmen (SIMD)

Zusammenfassung und Ausblick 2

Laufzeit / Speicherbedarf oft als rekursive Funktionen

- Wie löst man diese?
  Erzeugende Funktionen, Master-Theorem

Modelle entsprechen nicht der Realität!

- Genauere Modelle oder Abschätzung?
  O-Notation

Zuvor beantworten wir zwei entscheidende Fragen:

Berechenbarkeit

Gibt es Problemstellungen, zu denen man **keinen** Algorithmus finden kann?

Komplexität

Gibt es besonders schwere bzw. einfache Problemstellungen?