

AuD: 21.07. (2.DS)

Prog: 27.07.

PROGRAMMIERUNG

ÜBUNG 10: C₀ UND ABSTRAKTE MASCHINE AM₀

Eric Kunze

eric.kunze@mailbox.tu-dresden.de

INHALT

1. Funktionale Programmierung
 - 1.1 Einführung in Haskell: Listen
 - 1.2 Algebraische Datentypen
 - 1.3 Funktionen höherer Ordnung
 - 1.4 Typpolymorphie & Unifikation
 - 1.5 Beweis von Programmeigenschaften
 - 1.6 λ -Kalkül
2. Logikprogrammierung
3. Implementierung einer imperativen Programmiersprache
 - 3.1 **Implementierung von C_0**
 - 3.2 Implementierung von C_1
4. Verifikation von Programmeigenschaften
5. H_0 – ein einfacher Kern von Haskell

Implementierung von C_0 und abstrakte Maschine AM_0

- **Ziel:** Implementierung einer einfachen Programmiersprache $C_1 \subset C$

- ▶ **Ziel:** Implementierung einer einfachen Programmiersprache $C_1 \subset C$
- ▶ **Hier:** zunächst Einschränkung auf $C_0 \subset C_1$
 - ▶ genau eine main-Funktion
 - ▶ Zugriff auf stdio durch #include
 - ▶ einzig zugelassene Datenstruktur: int, Konstanten
 - ▶ Kontrollstrukturen: Ein-/Ausgabebefehle, Zuweisungen, Sequenzen, Verzweigungen, bedingte Schleifen

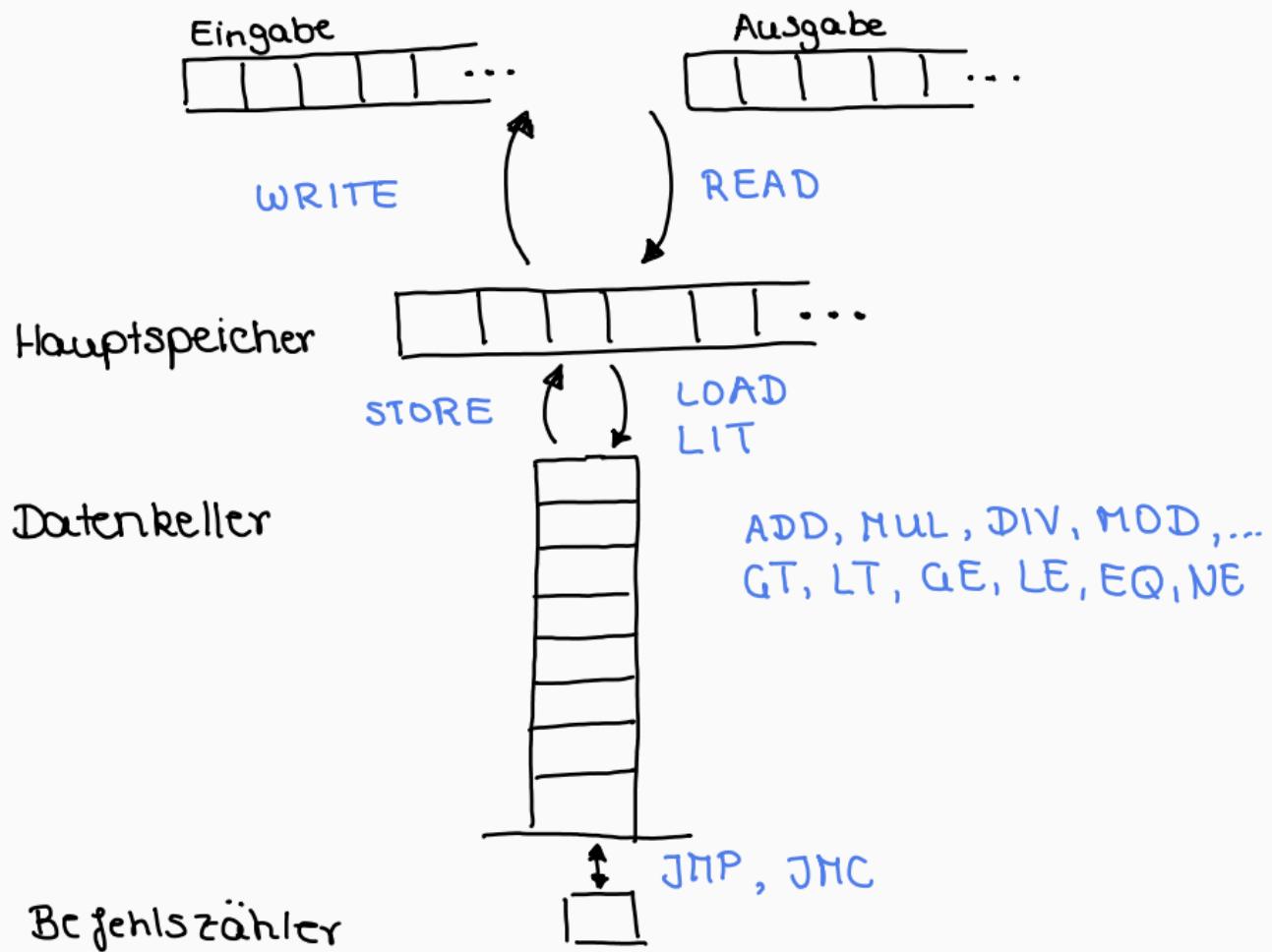
- ▶ **Ziel:** Implementierung einer einfachen Programmiersprache $C_1 \subset C$
- ▶ **Hier:** zunächst Einschränkung auf $C_0 \subset C_1$
 - ▶ genau eine main-Funktion
 - ▶ Zugriff auf stdio durch #include
 - ▶ einzig zugelassene Datenstruktur: int, Konstanten
 - ▶ Kontrollstrukturen: Ein-/Ausgabebefehle, Zuweisungen, Sequenzen, Verzweigungen, bedingte Schleifen
- ▶ **Implementierung** durch
 - ▶ Syntax von C_0
 - ▶ Befehle und Semantik einer abstrakten Maschine AM_0
 - ▶ Übersetzer $C_0 \leftrightarrow AM_0$

BEFEHLE UND SEMANTIK DER AM_0

Wir bauen eine abstrakte Maschine AM_0 , die unsere Berechnungen ausführen kann. Wir benötigen dafür:

- ▶ ein Ein- und Ausgabeband,
- ▶ einen Datenkeller,
- ▶ einen Hauptspeicher und
- ▶ einen Befehlszähler

Nun müssen aber auch Aktionen ausgeführt werden, wie zum Beispiel das Einlesen vom Eingabeband in den Hauptspeicher. Dafür gibt es folgende Befehle:



SEMANTIK DER BEFEHLE

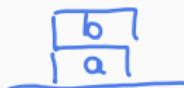
Den Zustand der abstrakten Maschine beschreiben wir durch die Zustände der 5 Komponenten, also als 5-Tupel

$$(m, d, h, \underline{inp}, \underline{out})$$

= (Befehlszähler, Datenkeller, Hauptspeicher, Input, Output)

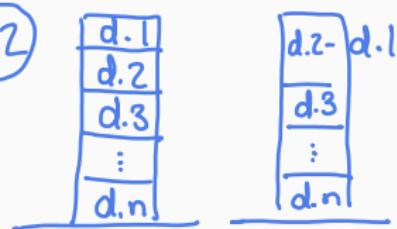
Jeder Befehl verändert den Zustand der Maschine – er verändert also die Einträge in diesem Tupel.

$a - b$



$d.1 - d.2$

$b - a$



$\underline{C}[\text{SUB}](m, d, h, \underline{inp}, \underline{out}) :=$

if $d = d.1 : d.2 : \dots : d.n$

then $(m + 1, \underline{(d.2 - d.1)} : d.3 : \dots : d.n, \underline{inp}, \underline{out})$

h

SEMANTIK DER BEFEHLE

formation der AM_0 dieser Befehl bewirkt. Dabei schreiben wir statt $C[\cdot](\gamma)$ nun $C[\gamma]$.

- $C[\text{ADD}](m, d, h, \text{inp}, \text{out}) :=$

if $d = d.1 : d.2 : d.3 : \dots : d.n$ mit $n \geq 2$ then $(m + 1, \underline{(d.2 + d.1)} : d.3 : \dots : d.n, h, \text{inp}, \text{out})$

für MUL analog

- $C[\text{SUB}](m, d, h, \text{inp}, \text{out}) :=$

if $d = d.1 : d.2 : d.3 : \dots : d.n$ mit $n \geq 2$ then $(m + 1, \underline{(d.2 - d.1)} : d.3 : \dots : d.n, h, \text{inp}, \text{out})$

für DIV und MOD analog

- $C[\text{LT}](m, d, h, \text{inp}, \text{out}) :=$

if $d = d.1 : d.2 : d.3 : \dots : d.n$ mit $n \geq 2$ then $(m + 1, b : d.3 : \dots : d.n, h, \text{inp}, \text{out})$

wobei $b = 1$, falls $\underline{d.2 < d.1}$, und $b = 0$, falls $\underline{d.2 \geq d.1}$,

d. h. für den Wert *true* (bzw. *false*) wird 1 (bzw. 0) abgelegt

für EQ, NE, GT, LE und GE analog

- $C[\text{LOAD } n](m, d, h, \text{inp}, \text{out}) :=$

if $h(n) \in \mathbb{Z}$ then $(m + 1, \underline{h(n)} : d, h, \text{inp}, \text{out})$

- $C[\text{LIT } z](m, d, h, \text{inp}, \text{out}) := (m + 1, \underline{z} : d, h, \text{inp}, \text{out})$

- $C[\text{STORE } n](m, d, h, \text{inp}, \text{out}) :=$

if $d = d.1 : d'$ then $(m + 1, d', h[n/d.1], \text{inp}, \text{out})$

wobei $h[n/d.1](k) = \begin{cases} d.1 & \text{falls } k = n \\ h(k) & \text{sonst} \end{cases}$

- $C[\text{JMP } e](m, d, h, \text{inp}, \text{out}) := (\underline{e}, d, h, \text{inp}, \text{out})$

- $C[\text{JMC } e](m, d, h, \text{inp}, \text{out}) :=$

if $d = 0 : d.2 : \dots : d.n$ mit $n \geq 1$ then $(\underline{e}, d.2 : \dots : d.n, h, \text{inp}, \text{out})$

if $d = \underline{1} : d.2 : \dots : d.n$ mit $n \geq 1$ then $(m + 1, d.2 : \dots : d.n, h, \text{inp}, \text{out})$

Es wird also zum Befehl mit der Nummer e gesprungen, wenn das oberste Kellerelement gleich 0 ist; die 0 repräsentiert den Wert *false*. Wenn das oberste Kellerelement gleich 1 ist (und damit den Wert *true* repräsentiert), dann wird der Befehlszähler um 1 inkrementiert.

- $C[\text{READ } n](m, d, h, \text{inp}, \text{out}) :=$

if $\text{inp} = \underline{\text{first}(\text{inp}).\text{rest}(\text{inp})}$ then $(m + 1, d, \underline{h[n/\text{first}(\text{inp})]}, \text{rest}(\text{inp}), \text{out})$

wobei für jedes $n \in \mathbb{Z}$ und $w \in \mathbb{Z}^*$ gilt: $\text{first}(n : w) = n$ und $\text{rest}(n : w) = w$



ÜBERSETZUNG VON IF - THEN - ELSE

$\text{sttrans}(\text{if } \langle \text{exp} \rangle \text{ stat}_1 \text{ else stat}_2, \text{tab}, a)$:=
→ $\text{boolexprtrans}(\text{exp}, \text{tab})$

JMC a.1;
 $\text{sttrans}(\text{stat}_1, \text{tab}, a.2)$ then - Zweig
JMP a.3;
a.1 : $\text{sttrans}(\text{stat}_2, \text{tab}, a.4)$ else - Zweig
a.3 :

für alle $\text{exp} \in W(\langle \text{BoolExpression} \rangle)$, $\text{stat}_1, \text{stat}_2 \in W(\langle \text{Statement} \rangle)$,
 $\text{tab} \in \text{Tab}$ und $a \in \mathbb{N}^*$.

AUFGABE 1

Wir betrachten das C_0 -Programm *Max*:

```
1 #include <stdio.h>          7     if (a > b)
2                               8         max = a;
3 int main( ) {                9     else max = b;
4     int a, b, max;           10    printf("%d", max);
5     scanf("%i", &a);          11    return 0;
6     scanf("%i", &b);          12 }
```

- (a) Berechnen Sie schrittweise das baumstrukturierte Programm $bMax_0 = \text{trans}(Max)$ mit Hilfe der in der Vorlesung angegebenen Übersetzungsfunktionen. Dokumentieren Sie dabei jeden rekursiven Funktionsaufruf.

AUFGABE 1 – TEIL (A)

Baumstrukturierte Adressen:

```
READ 1;  
READ 2;  
LOAD 1;  
LOAD 2;  
GT;  
JMC 1.3.1;  
LOAD 1;  
STORE 3;  
JMP 1.3.3;  
1.3.1  LOAD 2;  
      STORE 3;  
1.3.3  WRITE 3;
```

AUFGABE 1 – TEIL (A)

Baumstrukturierte Adressen: Linearisierte Adressen:

READ 1;	
READ 2;	1 READ 1;
LOAD 1;	2 READ 2;
LOAD 2;	3 LOAD 1;
GT;	4 LOAD 2;
JMC 1.3.1;	5 GT;
LOAD 1;	6 JMC 10;
STORE 3;	7 LOAD 1;
JMP 1.3.3;	8 STORE 3;
1.3.1 LOAD 2;	9 JMP 12;
STORE 3;	10 LOAD 2;
1.3.3 WRITE 3;	11 STORE 3;
	12 WRITE 3;

AUFGABE 1 – TEIL (B)

Ablauf der abstrakten Maschine:

BZ	,	DK	,	HS	,	Inp	,	Out	
(1	,	ε	,	[]	,	5:7	,	ε)
(2	,	ε	,	[1/5]	,	7	,	ε)
(3	,	ε	,	[1/5, 2/7]	,	ε	,	ε)
(4	,	5	,	[1/5, 2/7]	,	ε	,	ε)
(5	,	7:5	,	[1/5, 2/7]	,	ε	,	ε)
(6	,	0	,	[1/5, 2/7]	,	ε	,	ε)
(10	,	ε	,	[1/5, 2/7]	,	ε	,	ε)
(11	,	7	,	[1/5, 2/7]	,	ε	,	ε)
(12	,	ε	,	[1/5, 2/7, 3/7]	,	ε	,	ε)
(13	,	ε	,	[1/5, 2/7, 3/7]	,	ε	,	7)

AUFGABE 1 – TEIL (B)

Ablauf der abstrakten Maschine:

BZ	,	DK	,	HS	,	Inp	,	Out	
(1	,	ε	,	[]	,	5:7	,	ε)
(2	,	ε	,	[1/5]	,	7	,	ε)
(3	,	ε	,	[1/5, 2/7]	,	ε	,	ε)
(4	,	5	,	[1/5, 2/7]	,	ε	,	ε)
(5	,	7:5	,	[1/5, 2/7]	,	ε	,	ε)
(6	,	0	,	[1/5, 2/7]	,	ε	,	ε)
(10	,	ε	,	[1/5, 2/7]	,	ε	,	ε)
(11	,	7	,	[1/5, 2/7]	,	ε	,	ε)
(12	,	ε	,	[1/5, 2/7, 3/7]	,	ε	,	ε)
(13	,	ε	,	[1/5, 2/7, 3/7]	,	ε	,	7)

$$\mathcal{P}[\![\text{Max}_0]\!](5 : 7) = \text{proj}_5^{(5)} \left(\mathcal{I}[\![\text{Max}_0]\!](1, \varepsilon, [], 5 : 7, \varepsilon) \right) = 7$$

AUFGABE 2 – TEIL (A)

```
1 #include <stdio.h>          8      x1 = x2 - x1;
2                               9      if (x2 > x1)
3 int main() {                  10     x2 = x2 / 2;
4     x1, x2;                 11 }
5     scanf("%i", &x1);        12     printf("%d", x1);
6     scanf("%i", &x2);        13     return 0;
7     while (x1 > 0){         14 }
```

Übersetzen Sie das Programm mittels trans in AM_0 -Code mit linearen Adressen. Geben Sie nur das Endergebnis der Übersetzung (keine Zwischenschritte) an!

AUFGABE 2 – TEIL (B)

3 LOAD 2;	6 JMC 14;	9 LIT 2;	12 STORE 2;
4 LIT 5;	7 LOAD 1;	10 MUL;	13 JMP 3;
5 LT;	8 LOAD 2;	11 ADD;	14 WRITE 1;

Erstellen Sie ein Ablaufprotokoll für dieses
Programmfragment, bis die AM_0 terminiert. Die
Startkonfiguration ist $(7, \varepsilon, [1/3, 2/1], \varepsilon, \varepsilon)$.