PROGRAMMIERUNG

ÜBUNG 10: C_0 UND ABSTRAKTE MASCHINE AM_0

Eric Kunze

eric.kunze@mailbox.tu-dresden.de

INHALT

Implementierung von C_0 und abstrakte Maschine AM_0

C_0 UND AM_0

- **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 zugelassende 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

$$\begin{aligned} & (m, d, h, inp, out) \\ = \; & (\text{Befehlszähler}, \text{Datenkeller}, \text{Hauptspeicher}, \text{Input}, \text{Output}) \end{aligned}$$

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

$$\begin{aligned} & \mathcal{C}[\![\mathtt{SUB}]\!](m, d, h, inp, out) := \\ & \qquad \text{if } d = d.1 : d.2 : \cdots : d.n \\ & \qquad \text{then } (m + 1, (d.2 - d.1) : d.3 : \cdots : d.n, inp, out) \end{aligned}$$

SEMANTIK DER BEFEHLE

$(AM_0 \to AM_0)$ gibt zu jedem Befehl der AM_0 an, welche Zustandstransformation der AM_0 dieser Befehl bewirkt. Dabei schreiben wir statt $\mathcal{C}[\![.]\!](\gamma)$ nun $\mathcal{C}[\![\gamma]\!]$.

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

  if $d = d.1 : d.2 : d.3 : \ldots : d.n$ mit $n \geq 2$ then $(m+1, (d.2 + d.1) : d.3 : \ldots : d.n, h, inp, out)$

  für MUL analog

  $\mathcal{C}[\![\mathrm{SUB}]\!](m, d, h, inp, out) :=$

  if $d = d.1 : d.2 : d.3 : \ldots : d.n$ mit $n \geq 2$ then $(m+1, (d.2 - d.1) : d.3 : \ldots : d.n, h, inp, out)$

  für DIV und MOD analog

  $\mathcal{C}[\![\mathrm{LT}]\!](m, d, h, inp, out) :=$

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

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

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

  für EQ, NE, GT, LE und GE analog

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

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

  $\mathcal{C}[\![\mathrm{LIT}\ z]\!](m, d, h, inp, out) := (m+1, z : d, h, inp, out)$

  $\mathcal{C}[\![\mathrm{STORE}\ n]\!](m, d, h, inp, out) :=$

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

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

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

  $\mathcal{C}[\![\mathrm{JMC}\ e]\!](m, d, h, inp, out) :=$

  if $d = 0 : d.2 : \ldots : d.n$ mit $n \geq 1$ then $(e, d.2 : \ldots : d.n, h, inp, out)$

  if $d = 1 : d.2 : \ldots : d.n$ mit $n \geq 1$ then $(m+1, d.2 : \ldots : d.n, h, inp, out)$

  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.

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

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

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

ÜBERSETZUNG VON IF - THEN - ELSE

$$
\begin{aligned}
\mathit{sttrans}(\texttt{if }(\mathit{exp})\ \mathit{stat}_1\ \texttt{else}\ \mathit{stat}_2, \mathit{tab}, a) := \\
& \mathit{boolexptrans}(\mathit{exp}, \mathit{tab}) \\
& \texttt{JMC}\ a.1; \\
& \mathit{sttrans}(\mathit{stat}_1, \mathit{tab}, a.2) \\
& \texttt{JMP}\ a.3; \\
a.1:\ & \mathit{sttrans}(\mathit{stat}_2, \mathit{tab}, a.4) \\
a.3:\ &
\end{aligned}
$$

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

AUFGABE 1

Wir betrachten das C_0-Programm *Max*:

```
#include <stdio.h>

int main( ) {
  int a, b, max;
  scanf("%i", &a);
  scanf("%i", &a);
  if (a > b)
    max = a;
  else max = b;
  printf("%d", max);
  return 0;
}
```

(a) Berechnen Sie schrittweise das baumstrukturierte Programm $bMax_0 = \mathrm{trans}(Max)$ 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;
```

Linearisierte Adressen:

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

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	)

$$\mathcal{P}[\![Max_0]\!](5:7) = proj_5^{(5)}\Big(\mathcal{I}[\![Max_0]\!](1, \varepsilon, [], 5:7, \varepsilon)\Big) = 7$$

AUFGABE 2 – TEIL (A)

```c
#include <stdio.h>

int main() {
  int x1, x2;
  scanf("%i", &x1);
  scanf("%i", &x2);
  while (x1 > 0){
    x1 = x2 - x1;
    if (x2 > x1)
      x2 = x2 / 2;
  }
  printf("%d", x1);
  return 0;
}
```

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 (A)

```
READ 1;
READ 2;
LOAD 1;
LIT 0;
GT;
JMC 20;
LOAD 2;
LOAD 1;
SUB;
STORE 1;
LOAD 2;
LOAD 1;
GT;
JMC 19;
LOAD 2;
LIT 2;
DIV;
STORE 2;
JMP 3;
WRITE 1;
```

AUFGABE 2 – TEIL (B)

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

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

AUFGABE 2 – TEIL (B)

Ablauf der abstrakten Maschine:

	BZ	DK	HS	Inp	Out	
(	7	ε	[1/3, 2/1]	ε	ε	)
(	8	3	[1/3, 2/1]	ε	ε	)
(	9	1:3	[1/3, 2/1]	ε	ε	)
(	10	2:1:3	[1/3, 2/1]	ε	ε	)
(	11	2:3	[1/3, 2/1]	ε	ε	)
(	12	5	[1/3, 2/1]	ε	ε	)
(	13	ε	[1/3, 2/5]	ε	ε	)
(	3	ε	[1/3, 2/5]	ε	ε	)
(	4	5	[1/3, 2/5]	ε	ε	)
(	5	5:5	[1/3, 2/5]	ε	ε	)
(	6	0	[1/3, 2/5]	ε	ε	)
(	14	ε	[1/3, 2/5]	ε	ε	)
(	15	ε	[1/3, 2/5]	ε	3	)