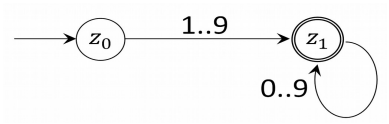
Informatik Abitur Bayern 2015 / IV - Beispiellösung

Autor:
Wörnle

1a 2

```
<zahl>   →   <zahl> <ziffer>
         →   <zahl> <ziffer> <ziffer1>
         →   <ziffer> 0 3
         →   <ziffer1> 0 3
         →   2 0 3
```

1b 3



1c 7

Eingabealphabet:

```
Σ = {"1", "2", "3", "4", "5", "6", "7", "8", "9", "0",
     "load", "store", "add", "sub", "jmp", "jeq", "hold", ";" }
```

Nichtterminalsymbole:

```
N = {programm, zeile, zahl, befehl,
     befehl1, ziffer, ziffer1}
```

Startsymbol:

```
S = programm
```

Produktionsregeln:

```
P={
programm    =   zeile programm | zeile;
zeile       =   zahl befehl ";";
befehl      =   "hold" | befehl1 zahl;
befehl1     =   "load" | "store" | "add" | "sub" | "jmp" | "jeq";
zahl        =   zahl ziffer | ziffer1;
ziffer      =   "0" | ziffer1;
ziffer1     =   "1"| "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
}
```

2a 4

Für Algorithmus 1 gilt:
Für a=5 und n=4 wird der Algorithmus 5 mal durchlaufen. 5 · 5 · 5 · 5 · 1
Für a=5 und n=8 wird der Algorithmus entsprechend 9 mal durchlaufen.

Für Algorithmus 2 gilt:
Für a=5 und n=4 wird der Algorithmus 4 mal durchlaufen. Der Ablauf ist dabei:

```
potenz2(5,4) =
quadrat(potenz2(5,2)) =
quadrat(quadrat(potenz2(5,1))) =
quadrat(quadrat(5 · potenz2(5,0))) =
quadrat(quadrat(5 · 1))
```

Für a=5 und n=8 wird der Algorithmus nur einmal öfter, also 5 mal durchlaufen, da nur
`potenz2 (5,8) = quadrat(potenz2(5,4))=...`
hinzu kommt.

(Verallgemeinerung: Für a=5 und n=2^m wird der Algorithmus 2+m mal durchlaufen)

2b 3

Bei Algorithmus 1 werden immer n+1 Durchläufe benötigt. Daher wird die Anzahl der Durchläufe beim Verdoppeln von n auf (2·n) +1 erhöht.
Bei Algorithmus 2 ist bei Verdoppelung von n nur ein zusätzlicher Durchlauf nötig.

2c 4

Bei Verwendung von Algorithmus 1 sind 1.000.001 Methodenaufrufe nötig. Bei 10^{-6} s pro Durchlauf wären daher 1.000.001 · 10^{-6} s = 1,000001 Sekunden nötig um ein Ergebnis zu erhalten.
Bei Verwendung von Algorithmus 2 gilt folgende Abschätzung nach oben (vgl. Verallgemeinerung bei 2a):

$2^m \geqslant 1.000.000$ => $m \geqslant ld(1.000.000) = 19{,}93$, also m = 20.

Nach der Verallgemeinerung in 2a) sind somit höchstens 20+2 = 22 Aufrufe notwendig, dies entspricht 0,000022 s.
(Auch andere Abschätzungen sind möglich. Der exakte Wert ist 21.)

3a 4

z1	z2	erg
13	5	0
6	10	5
3	20	5
1	40	20+5=25
0	80	25+40=65

3b In der Von-Neumann Architektur ist nur ein einziges Speicherwerk vorgesehen, bestehend aus Speicherzellen gleicher Größe. Dabei werden alle Programme und Daten auf dem selben Speicher abgelegt. 3
Vorteile sind z. B.:

- Programme und deren benötigte Daten können physisch näher und flexibler abgelegt werden.
- Da nur ein Speicher verbaut ist werden Kosten gespart.
- Der Programmablauf und Speicherzugriff ist streng deterministisch, Fehler werden vermieden.

Nachteile sind z.B.:

- Da sowohl Daten als auch Befehle über den selben Bus in die ALU und ins Steuerwerk befördert werden müssen, können sich Datenstaus ergeben (Von-Neumann-Flaschenhals)
- Da eine Speicherverwaltung fehlt, kann der Speicher an beliebigen Stellen unbeabsichtigt überschrieben werden.

3c Enthält die Speicherzelle 101 eine ungerade Zahl, wie z.B. 9, so wird diese mit shr ohne Rest geteilt. Im Akkumulator steht also 4. Im Anschluss wird die Zahl 4 mit 2 multipliziert, sodass danach 8 im Akkumulator steht. Nach Subtraktion der Ausgangszahl 9 folgt ein negatives Ergebnis, nämlich -1. 3

Enthält die Speicherzelle 101 hingegen eine gerade Zahl, wie z.B. 10, steht nach dem Teilen und Multiplizieren wieder 10 im Akkumulator. Bei Subtraktion von der Ausgangszahl folgt die 0 als Ergebnis. Mit jeq x ist es nun möglich, diese Erkenntnis zu nutzen, da nur bei 0 gesprungen wird.

3d 7

```
 1: load 101
 2: jeq 17       // Falls z1=0 wird das Verfahren sofort beendet.
                 // Überprüfung ob z1 ungerade ist.
 3: shr
 4: shl
 5: sub 101
 6: jeq 10
                 // 1. Fall: z1 ist ungerade
 7: load 103     // 103=erg mit dem Startwert 0
 8: add 102
 9: store 103

10: load 101
11: shr
12: store 101
13: load 102
14: shl
15: store 102
16: jump 1       // Wiederhole bis Speicherzelle 101=0

17: end
```