Informatik Ↄ: – Blatt 8

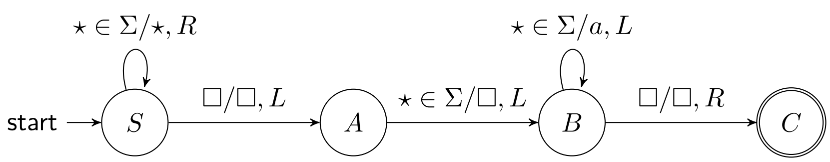
Rasmus Diederichsen

26. Juli 2014

Aufgabe 8.1

Den abgerundeten Logarithmus zu berechnen, ist dasselbe, wie den Index (nullbasiert) des höchsten gesetzten Bits zu bestimmen. Da wir annehmen, dass α keine führenden Nullen hat, ist dies gleichzeitig das Bit am linken Ende der Eingabe. Die Anzahl an zu schreibenden as ist also gleich dem Index des höchstwertigsten Bits. Die Turingmaschine hierzu ist (z.B.) folgende:

Sei $\Sigma = \{0, 1\}$



Hierbei ist die Kante zwischen A und B dafür zuständig, eine Stelle der Eingabe abzuschneiden, da man ein a weniger braucht, als die Eingabe lang ist.

Aufgabe 8.2

a)

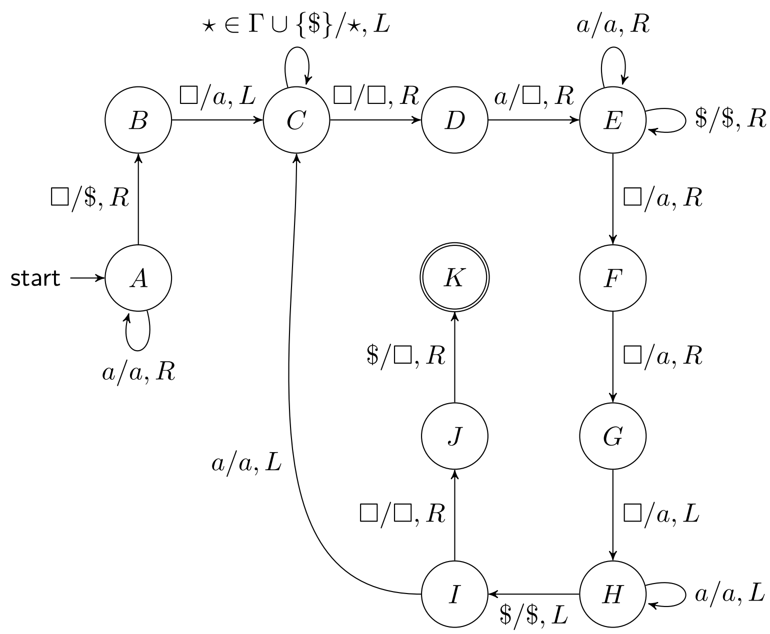
Man geht wie folgt vor

```
Laufe nach ganz R
Ersetze Blank durch $, gehe R
Ersetze Blank durch a, gehe L // +1
Gehe nach ganz L // stehe auf Blank
Gehe R
Falls $ gelesen: // passiert nicht in 1. Iteration
   ersetze durch Blank, gehe R
   terminiere
sonst: // a gelesen
   Ersetze a durch Blank // löschen
   Gehe R, bis $ gelesen
```

```
Gehe R, solange a gelesen
Ersetze Blank durch a, gehe R // 3 as schreiben
Ersetze Blank durch a, gehe R
Ersetze Blank durch a, gehe L
Goto Stelle 4
```

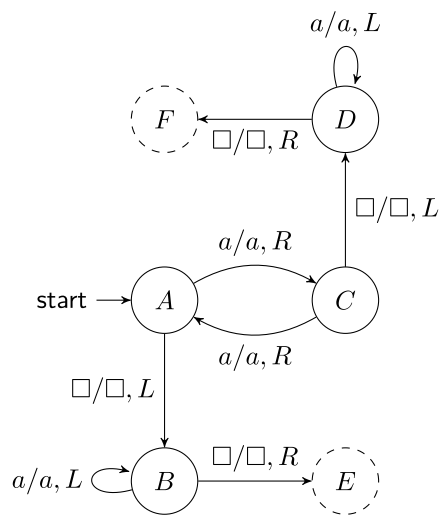
Die Turingmaschine hierzu lautet:

Sei $\Gamma = \{a\}$



b)

Zunächst wird mittels folgendem Automaten bestimmt, ob die Eingabe gerade oder ungerade Länge hat:



Hierbei folgt an Knoten F der Automat, der ungerade Eingaben verarbeitet, und an E der für gerade Eingaben. Im Weiteren geht man dann wie folgt vor:

```
1  Bestimme, ob Eingabe gerade oder ungerade
2  Falls ungerade
3     gehe vor gemäß a)
4  sonst
5     Laufe nach ganz R // stehe auf Blank
6     Ersetze Blank durch $ // markiere Output-Buffer
7     Laufe nach ganz L // stehe auf Blank
8     Gehe R
9     Falls $ gelesen, ersetze durch Blank, gehe R
10       terminiere
11    sonst // stehe auf a
12       Ersetze a durch Blank, gehe R
13       Ersetze a durch Blank, gehe R // min 2 as, da gerade und > 0
14       Laufe ganz R bis $ gelesen // bis zum Output-Buffer
15       Laufe R solange a gelesen // zum Ende des Buffers
16       Ersetze Blank durch a, gehe L // zwei a gelöscht, eins geschrieben
17       Goto Stelle 7
```

Aufgabe 8.3

Eine Turingmaschine $\mathcal{M}$, die nur n Felder des Bandes beschreibt, kann eine solche simulieren, die $c \cdot n$ Felder beschreibt und das Alphabet Γ besitzt, wenn sie ein Alphabet $\Gamma_{\mathcal{M}} = \{k \mid k \in \Gamma^c\}$ besitzt, ihr Alphabet also eine Menge von c-Tupeln aus Γ ist. Dies ist effektiv das selbe wie eine mehrbändige Turingmaschine mit jeweile endlichem Speicher, da in einem Übergang einzelne Einträge des Tupels geändert werden können. Der Automat wird dafür mehr Zustände benötigen.

Aufgabe 8.4

Goto-Programm

Das `Goto`-Programm ist relativ simpel. Man speichert sich die Werte der beiden Zahlen, prüft, ob die erste ungleich null ist, wenn nein, gibt man sie zurück. Wenn ja, prüft man, ob die zweite Zahl ungleich null ist. Wenn nein, gibt man die zweite Zahl zurück (kleiner als 0 kann das Minimum nicht sein). Wenn ja, so dekrementiert man beide Zahlen (bzw. ihre Kopien) und beginnt wieder oben.

```
     x_4 := x_1
     x_5 := x_2
L_1: if(x_4 ≠0) goto L_2
     x_3 := x_1
     halt
L_2: if(x_5 ≠0) goto L_3
     x_3 := x_2
     halt
L_3: goto L_4
L_4: x_4 := x_4 - 1
     x_5 := x_5 - 1
     goto L_1
```

While-Programm

Aus dem obigen Programm wird gemäß dem Vorgehen aus den Slides ein `While`-Programm generiert. Dies wird verkompliziert dadurch, dass Prüfungen der Form `if(x =` c`)` nicht erlaubt sind. Daher wird eine Variable immer dekrementiert, und geprüft, ob sie 0 ist. So stellt man fest, welchen Wert der Programmzeiger (x_{pos}) hat. Ärgerlicherweise kann man nur auf Ungleichheit prüfen, sodass die Anweisungen für den Fall, dass der Zeiger einen bestimmten Wert hatte, in die `else`-Klauseln am Ende wandern. Das Ganze muss geschatelt werden und es kommt folgender Programmsalat dabei heraus:

```
x_0 := 1; x_4 := x_1; x_5 := x_2
while(x_0 ≠ 0) {
   x_6 := x_0 - 1 // test if pos is 1
   if(x_6 ≠ 0) {
      x_6 := x_6 - 1 // test if pos is 2
      if(x_6 ≠ 0) {
         x_6 := x_6 - 1 // test if pos is 3
         if(x_6 ≠ 0) {
            x_0 := 1 // jump to L_1
         } else { // pos is 4
            x_4 := x_4 - 1
            x_5 := x_5 - 1
            x_0 := 1 // start at top
         }
      } else { // pos was 2
         if(x_5 ≠ 0) { x_0 := 3 } // jump to L_3
         else {
            x_3 := x_2 // x_5 is 0, so the first one is at least as large
            x_0 := 0 // terminate
```

```
        }
    }
} else { // pos was 1
    if(x_4 ≠ 0) { x_0 := 2 }
    else {
        x_3 := x_1; // if x_4 is 0, the other one is at least as large
        x_0 := 0 // terminate
    }
}
}
```

Per Hand lässt sich eine einfachere Möglichkeit finden (Dank an Hendrik).

```
x_f := 1;
t_1 := x_1;
t_2 := x_2;
while(x_f != 0) {
    if(t_1 != 0) {
        if(t_2 != 0){
            t_1 := t_1 - 1;
            t_2 := t_2 - 1;
        } else {
            x_3 := x_2;
            x_f := 0;
        }
    } else {
        x_3 := x_1;
        x_f := 0;
    }
}
```

Aufgabe 8.5

Aufgabe 8.6

a						c		a				
c						b		c				
b					a	a	c	a	c			
a	c	b	c	b	c	a	a	c	a	c	a	b