Matthias Jauernig
Kurs: 03In1

16.01.04
linux-related.de

13. Aufgabenserie zu den Grundlagen der Informatik

Abgabetermin: Mi, 21.01.04

Zu 37.) *Automat mit Ausgabe (Mealy-Automat)*

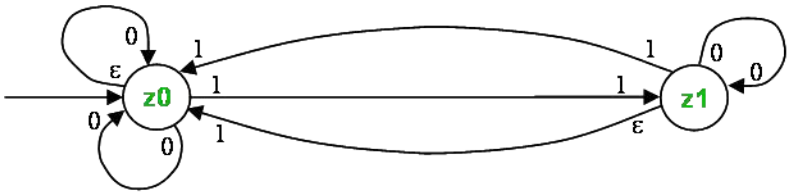
- A:=(E, A, Z, f, g, z_0);
  Automat soll das Anhängen eines Paritätsbits p an eine Zeichenkette w ∈ {0,1}* erledigen, wobei gilt: p=1, wenn Anzahl der Einsen in w gerade; p=0, wenn Anzahl der Einsen in w ungerade
- E:={0, 1, ε }
- A:={0,1}
- Z:={z_0, z_1}
- z_0 ∈ Z => Startzustand

- Tabelle der Überführungsfunktion f und der Ausgabefunktion g:

f, g	**0**	**1**	ε
z_0	z_0, 0	z_1, 1	z_0, 0
z_1	z_1, 0	z_0, 1	z_0, 1

  Dabei sei ε als das Zeichen definiert, welches das Ende der Zeichenkette anzeigt (leeres Zeichen).

- Graph:

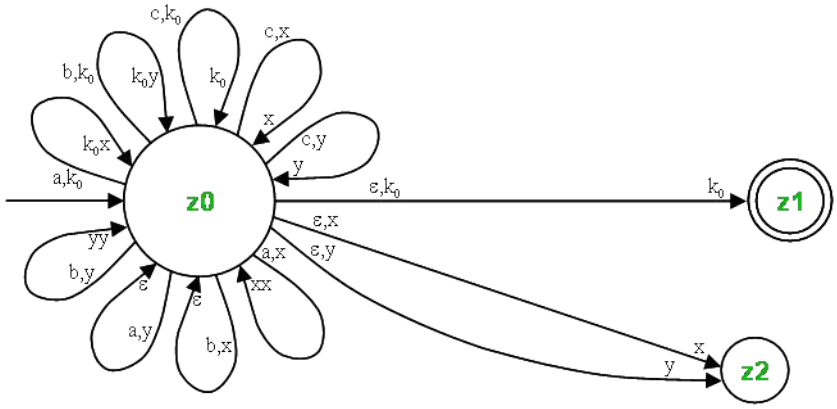


Zu 38.) *Kellerautomat*

- A:=(E, Z, k, f, g, z_0, k_0, F);
  Automat soll eine Zeichenkette genau dann akzeptieren, wenn $a^n=b^n$
- E:={a, b, c, ε}
- Z:={z_0, z_1, z_2}
- K:={k_0, x, y}
- F:={z_1}
- z_0 ∈ Z => Startzustand
- k_0 ∈ K => Kellerstartsymbol
- ε ..leeres Zeichen; zeigt das Ende der Zeichenkette an

- Definition der Überführungsfunktion f:
  $f(z_0, a, k_0) = (z_0, k_0x)$
  $f(z_0, b, k_0) = (z_0, k_0y)$
  $f(z_0, c, k_0) = (z_0, k_0)$
  $f(z_0, c, x) = (z_0, x)$
  $f(z_0, c, y) = (z_0, y)$
  $f(z_0, a, x) = (z_0, xx)$
  $f(z_0, a, y) = (z_0, \varepsilon)$
  $f(z_0, b, x) = (z_0, \varepsilon)$
  $f(z_0, b, y) = (z_0, yy)$
  $f(z_0, \varepsilon, k_0) = (z_1, k_0)$
  $f(z_0, \varepsilon, x) = (z_2, x)$
  $f(z_0, \varepsilon, y) = (z_2, y)$

- Graph:



Zu 39.) ***Turingmaschine***

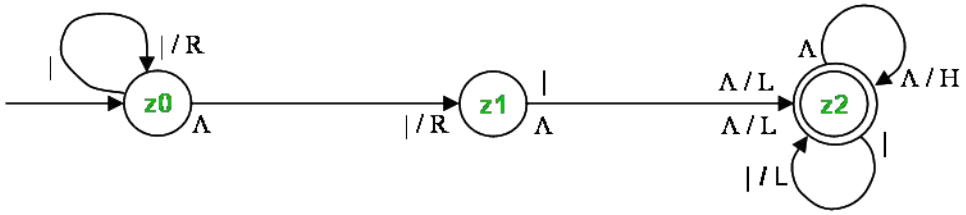
- T:=(Z, Σ, f, z_0, F);
  Die Turingmaschine soll zu einer vorgegebenen Zahl n ∈ den Nachfolger berechnen.
- Σ:={Λ, |}
- Z:={z_0, z_1, z_2}
- $z_0 \in Z$ (Startzustand)
- F:={z_2}

- Tabelle der Überführungsfunktion f:

f	Λ	|
z_0	z_1 | R	z_0 | R
z_1	z_2 ΛL	z_2 ΛL
z_2	z_2 ΛH	z_2 | L

Dabei werde angenommen, dass vor und nach der Zeichenkette, bestehend aus |, mit der die Zahl codiert ist, jeweils ein Λ steht und sich der LS-Kopf anfangs direkt links von der Strichfolge befindet. Allerdings gilt es zu berücksichtigen, dass ja ein | an das Ende der Zeichenkette angehangen wird und somit das Zeichen nach dem Λ am Ende noch geprüft werden und ggf. auf Λ gesetzt werden muss, um wieder eine abgeschlossene Zeichenkette zu erhalten. Hierfür ist z_1 notwendig.

- Graph:



Zu 40.) ***rekursive Primzahlbestimmung***

C-Programm, welches die rekursive Funktion prim() implementiert:

```
/*      prim_rek.c -- Matthias Jauernig, 14.01.04                       */
/*      Programm implementiert die rekursiv definierte Funktion prim()  */

#include <stdio.h>
#define MAX 500
int z=0;

char prim(int n){
        z++;
        if(n==2) return 1;
        if(n>2){
                int k;
                for(k=2; k*k<=n; k++)
                        if(prim(k) && !(n%k))
                                return 0;
                return 1;
        }
        return 0;
}

int main(void){
        int i;
        printf( "\nPrimzahlen im Bereich 1...%d:\n"
                "--------------------------------\n", MAX);
        for(i=1; i<=MAX; i++)
                if(prim(i))
                        printf("%d, ", i);
        printf("\b\b \n=> %d Aufrufe von prim() wurden dafür benötigt!\n\n", z);

        return 0;
}
/* Ausgabe des Programms mit MAX=500

Primzahlen im Bereich 1...500:
--------------------------------
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79,
83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173,
179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269,
271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373,
379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467,
479, 487, 491, 499
=> 4494 Aufrufe von prim() wurden dafür benötigt!
*/
```

An der Beispielausgabe sieht man schon ein entscheidendes Problem, welches für alle rekursiv definierten Algorithmen charakteristisch ist: für die Prüfung von 500 Zahlen auf Primalität wurde die Funktion prim() fast 4500 mal aufgerufen, rekursive Aufrufe natürlich mit eingeschlossen. Die rekursiven Aufrufe nehmen natürlich zusätzlichen Speicherplatz in Anspruch, da z.B. Rücksprungadressen und Variablen bei jedem Aufruf im Speicher abgelegt werden müssen. Hier mag es noch gehen, da prim() vom Typ char ist und höchstens 2 int-Variablen pro Aufruf reserviert werden, in Hinblick auf größere Projekte, wo viele Variablen gespeichert werden müssen, sind rekursive Funktionen allerdings als „Speicherfresser“ zu bezeichnen. Trotzdem haben sie ihre Daseinsberechtigung: auch wenn sich jede Rekursion mit Schleifen simulieren lässt, so lassen sich damit doch viele Algorithmen elegant beschreiben, womit dies ein wichtiger Zweig der Informatik ist…

Das Ende?

Zum Schluss des letzten Beleges in GdInf noch ein Wunsch: ich möchte nicht, dass es mir einmal so ergeht (ob der Wunsch in Erfüllung geht, muss sich am Ende erst noch zeigen):