© Hans-Georg Beckmann 2005

Inhalt

Einleitung.....1
Definition.....2
Transitionsgraph / Automatentafel 4
Simulation in Java 6
Aufgaben 9
Ein größeres Beispiel, Aufgaben 11
Lösungen 13
Simulation 2.....14
Eine Kopiermaschine.....19
Elementarmaschinen.....21
Zusammengesetzte Maschinen 22
Aufgaben 23
Rekursive Funktionen.....25
Aufgaben 33
Das Halteproblem 34
Chursche These, Berechenbarkeit 35
Fleißige Biber 36

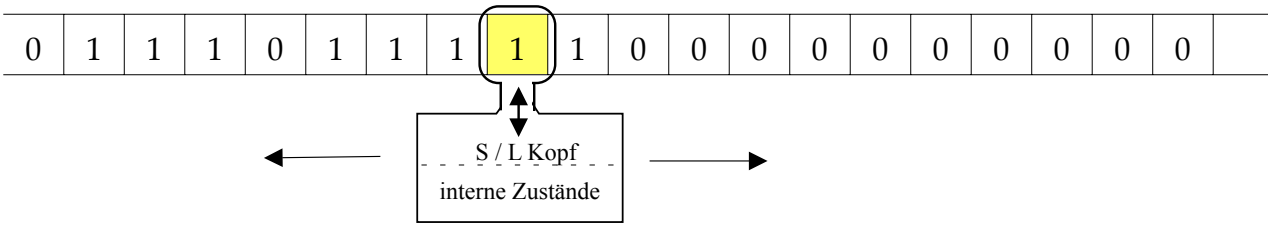
© Hans-Georg Beckmann 2005

Ein ganz besonderer Automat wurde von dem englischen Mathematiker Alan Turing 1936 entworfen.[1] Er entwickelte ein *"universelles Automatenmodell zum Studium algorithmischer Problemstellungen"*.

Die Speicherkapazität des Automaten soll unbegrenzt sein. Der Speicher selbst besteht aus einem unendlich langen Band, auf dem Speicherzellen / Speicherplätze angeordnet sind.

Dieses Band dient der Ein - und Ausgabe der Daten. Ein Schreib-Lesekopf kann auf dem Band schrittweise nach links oder rechts bewegt werden. (Man könnte auch festlegen, dass das Band sich nach recht / links bewegt und der Schreib - Lesekopf still steht.)

Weiterhin hat der Automat interne Zustände.



Es sollen folgende Festlegungen gelten:

Das Eingabealphabet und das Ausgabealphabet sind gleich. Es gibt nur die Zeichen "0" und "1".

$$E = A = \{ e_1, e_2 \} = \{ 0,1\}$$

Die Menge der internen Zustände sei $S = \{ s_0, s_1, s_2, \ldots\ldots, s_e\}$

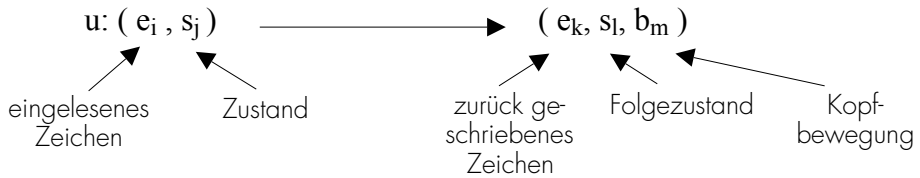
Die Menge der möglichen Kopfbewegungen des Schreib-Lesekopfes sei

$$B = \{b_1, b_2, b_3 \} = \{ L, H, R \}$$

Dabei bedeutet L= links bewegen, H = halten und R = rechts bewegen

Steht der Kopf an einer bestimmten Position des Bandes, wird das dort stehende Zeichen eingelesen, ein Zeichen an die gleiche Position zurückgeschrieben und dann der Kopf bewegt (wobei als Kopfbewegung auch H = halten auftreten kann).

Damit lässt sich eine Überführungsfunktion **u** angeben, die den Automaten beschreibt:



mit $i \in \{1,2\}$, $k \in \{ 1,2\}$, $j \in \{0,1,\ldots\ldots,e\}$, $l \in \{0,1,\ldots\ldots,e\}$, $m \in \{1,2,3\}$

[1] Alan Turing, 1912 - 1954

© Hans-Georg Beckmann 2005

Einige Aufmerksamkeit muss man nun der Codierung von Zeichen und Zahlen schenken, die man auf dem Band darstellen möchte. Da nur die Zeichen 0 und 1 zur Verfügung stehen, wird nun eine sehr willkürliche Festlegung getroffen:

Alle Zahlen / Ziffern n werden durch $n+1$ Einsen auf dem Band dargestellt. Damit kann also auch die Zahl Null durch genau eine 1 angegeben werden. Das Zeichen "0", das ja auch auf dem Band vorkommen kann, hat damit nicht die Bedeutung einer Ziffer. Die Null wird benutzt, um zwei Einsergruppen zu trennen. Außerdem beginne das Band links außen mit einer Null und nach der letzten Einsergruppe rechts stehen mindestens zwei Nullen auf dem Band. Die Standardstartposition sei dadurch festgelegt, dass der Scheib-Lesekopf auf der letzten "1" der am weitesten rechts stehenden Einsergruppe steht.

Anmerkung:

Die hier getroffene Festlegung der Zeichen ist durchaus willkürlich. Man findet in der Literatur sehr viele andere Angaben, die nicht besser und nicht schlechter sind , als die hier getroffenen Vereinbarungen.

Oft wird als Eingabealphabet eine Menge { 0, 1, # } benutzt wobei "#" für das Leezeichen steht, was dann die Rolle einnimmt, die in unserer Festlegung die Null hat. Damit kann man Zahlen in ihrer Dualdarstellung auf das Band schreiben.

In noch allgemeineren Darstellungen wird eine Einschränkung des Eingabealphabeths nur dahin gehend vorgenommen, dass die Zeichen für die Kopfbewegungen reserviert bleiben, aber alle anderen Zeichen zulässig sind.

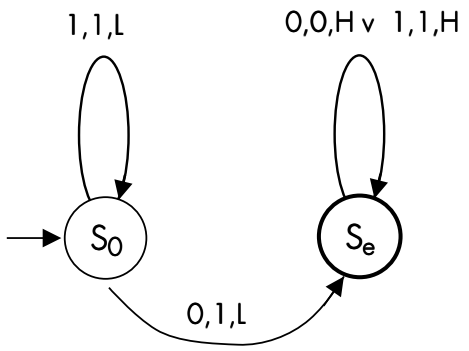
Wir werden weiter unten noch einmal auf einige Beispiele eingehen, in denen die Menge E in dieser angegebenen Weise erweitert ist.

Um die Funktion der Maschine zu beschreiben, gibt es zwei Wege, die wir schon von anderen endlichen Automaten kennen. Man kann einen Transitionsgraphen angeben, oder eine Tabelle, die die Überführungsfunktion darstellt.

Dazu ein einfaches Beispiel:

Die Maschine hat zwei Zustände. Sie startet in S_0.

An den Kanten des Graphen werden Tripel angegeben. Das erste Zeichen ist das vom Band gelesene, das zweite das zurückgeschriebene Zeichen und dann folgt noch die Kopfbewegung. Die Maschinenanweisung für (1,1,L) bedeutet also:



© Hans-Georg Beckmann 2005

" Wenn du eine "1" auf dem Band findest, dann schreibe eine "1" auf das Band zurück und bewege dich nach links."

Was leistet diese einfache Maschine ?

Das ist offensichtlich vom Zustand des Bandes zu Beginn der Arbeit abhängig.

Das Band sehe wie folgt aus, und der Kopf steht auf der "1" rechts außen.

0	1	1	1	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	

Die Maschine befindet sich im Zustand S_0 und findet eine "1" auf dem Band vor. Sie schreibt die "1" auf das Band zurück und geht um einen Schritt nach links. Da sie im Zustand S_0 bleibt, wiederholt sich das, bis die "0" gefunden wird. Jetzt wird eine "1" geschrieben, der Kopf bewegt sich wieder nach links und die Maschine befindet sich imZustand S_e, in dem sie in jedem Falle auf "Halten" steht, egal, welches Zeichen auf dem Band steht. Die obere Schleife im Transitionsgraphen könnte auch entfallen, denn wenn keine Angaben gemacht sind, dann gehen wir davon aus, dass der Endzustand (Halten) erreicht ist.

0	1	1	1	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	

Die Funktion der Maschine kann auch in Form einer Tabelle (Automatentafel) dargestellt werden:

Zustand	Eingabe-zeichen	Ausgabe-zeichen	Folge-zustand	Kopf-bewegung
S0	0	1	Se	L
S0	1	1	S0	L
Se	0	0	Se	H
Se	1	1	Se	H

Die Maschine findet die erste Null links von ihrer Startposition, überschreibt diese mit einer "1" , geht noch einen Schritt nach links und bleibt stehen.

Das war nun bestimmt nicht aufregend und man könnte auf den Gedanken kommen, dass Turing zu Unrecht mit seiner Maschine berühmt geworden ist. Deshalb wird es jetzt schon ein wenig gemeiner. Das unangenehme an den Truingmaschinen ist, dass sie leicht zu verstehen sind, wenn aber eine Maschine mit bestimmten Eigenschaften gebaut werden soll, wird es schnell unfreundlich. Beginnen wir mit einem übersichtlichen Problem:

© Hans-Georg Beckmann 2005

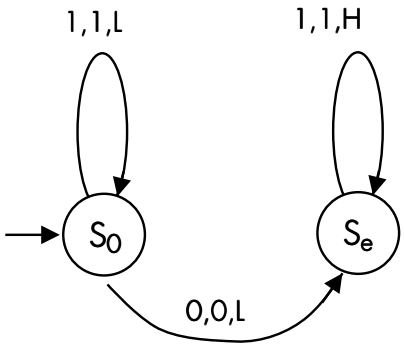
Aufgabe:

"Eine Maschine soll von der Startposition nach links laufen, bis sie die erste 1 *nach* einer Null findet und dort stehen bleiben. Stelle die Maschinentafel auf und gib auch den Transitionsgraphen an."

Lösung:

Zustand	Eingabe-zeichen	Ausgabe-zeichen	Folge-zustand	Kopf-bewegung
S0	1	1	S0	L
S0	0	0	Se	L
(Se	0	0	Se	L)
Se	1	1	Se	H

Der Aufgabenstellung entsprechend, kann die geklammerte Zeile nicht vorkommen. Solche Zeilen sollten weggelassen werden. Auch im Transitionsgraphen zeichnet man die entsprechenden Kanten nicht ein.



Start

0	1	1	1	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	

Ende

0	1	1	1	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	

Wie auch schon bei anderen Automaten, bietet es sich an, eine Turingmaschine durch ein (Java)-Programm zu simulieren. Dabei muss die Darstellung am Bildschirm nicht im Vordergrund stehen, obwohl sie am Ende doch die meiste Arbeit verursacht.

Die Startbeschriftung des Bandes möge auf eine möglichst einfache Weise bewerkstelligt werden. Das Band kann durch einen String repräsentiert werden, der in einem Textfeld ausgegeben wird oder durch eine Reihe von Textfeldern, die den einzelnen Feldern des Turingbandes entsprechen.

© Hans-Georg Beckmann 2005

Das eigentliche Programm lässt sich in einem Struktogramm übersichtlich darstellen[2]:

Maschine in den Anfangszustand bringen	
Wdh	Lese ein Zeichen vom Band
	Zeichen gemäß der Funktion *u* neu schreiben
	Kopfbewegung gemäß *u* ausführen
	Folgezustand einnehmen
bis Endzustand erreicht ist	

Zuerst die globalen Variablen und Klassen für das Programm:

```
//  Turing2java
//  Hans-Georg Beckmann on  Sun May 04 2005.
//  Eine Turingmaschine in der einfachsten Art

import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Turing2 extends Applet implements ActionListener
{
    Button step     = new Button("Step");           // Schrittweise abarbeiten
    Button reset    = new Button("Reset");          // ..der Name sagt alles
    Button run      = new Button("Run");            // .. auch hier ..
    Button start    = new Button("Startposition");  // S/L- Kopf in Starposition bringen
    char     Zustand,Folgezustand;           // Zustand und Folgezustand als Char vereinbart
    char     Lesezeichen,Schreibezeichen;    // eingelesenes Zeichen, ausgegebenes Zeichen
    char     Kopfbewegung;                   // kann L oder R oder H sein
    TextField[] TBand=new TextField[40];     //  das Band für die Darstellung am Bildschirm
    int      pos;                            // die Position des S/L-Kopfes
    Label    Nachricht=new Label("");
    String   Zustandsstring;                 // für die Ausgabe auf dem Bildschirm
```

In der Methode *init* kann man dann die Buttons und die Textfelder für das Turingband am Bildschirm plazieren. Die *paint*-Methode kann man benutzen, um die Position des S/L-Kopfes graphisch darzustellen. Im folgenden Ausschnitt des Javaprogramms ist die Überführungsfunktion *u* dargestellt, die den gesamten inneren Teil der im Struktogramm angegebenen Wiederholungsanweisung ausführt.

```
//*************************************************
// Die Überführungsfunktion erledigt alles
//*************************************************
    public void u()
    {
    switch (Zustand)              // Zustand "switchen"
    {
    case '0':
       switch (Lesezeichen)       // Eingabezeichen   "switchen "
        {
         case '0':
                   Folgezustand='e';                      // Folgezustand
                   Schreibezeichen='1';                   // Ausgabezeichen
                   TBand[pos].setText(""+Schreibezeichen); //Ausgabezeichen schreiben
                   Kopfbewegung='H';                       //Kopfbewegung
```

2 aus "Automaten, Schaltwerke, Sprachen" von E. Modrow, Dümmler 1987 Seite 183 f

© Hans-Georg Beckmann 2005

```
                    Zustandsstring="Se";         // zur Information der Zustand
                    break;
                case '1':
                    Folgezustand='1';
                    Schreibezeichen='0';
                    TBand[pos].setText(""+Schreibezeichen);
                    Kopfbewegung='L';
                    pos=pos-1;
                    Zustandsstring="S1";
                    break;
                default:
                }                 // Ende von switch Lesezeichen
                break;            // Ende von case Zustand 0
// ************* Nun das Ganze für den Zustand Nummer 1 , also S1 *****************
     case '1':
                switch (Lesezeichen)
                {
                case '0':
                   .................
                   .................
                   u.s.w

    default:
    } // Ende von switch Zustand

    }// Ende der Methode u
```

Die Methode *u* muss für jede Turingmaschine neu geschrieben werden !

Eine Methode *Startposition* findet die Eins ganz weit rechts:

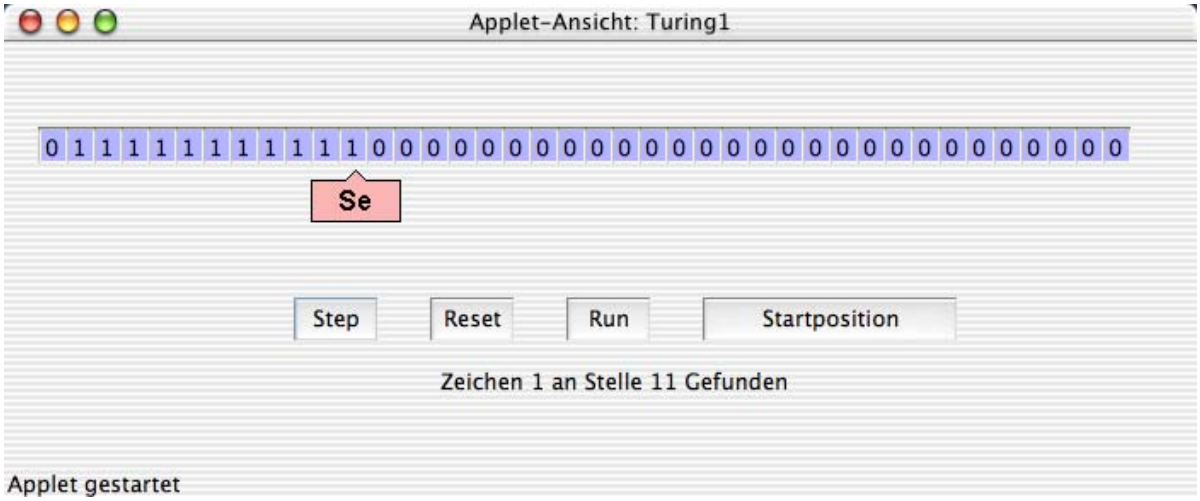
```
 //**********************************************
 //  Die Startposition einnehmen
 //  in dieser Maschine werden zwei Nullen gesucht
 //**********************************************
    public void Startposition_suchen()
    {
    int i=1;
    boolean gefunden=false;
    while (gefunden==false)
    {
    if((TBand[i].getText().charAt(0)=='0') && (TBand[i+1].getText().charAt(0)=='0'))
     gefunden=true;
     else
     i=i+1;
     }// Ende von while
    pos=i-1;
    Zustand='0';        // Anfangszustand
    Zustandsstring="S0";
    }
```

Weitere Methoden :

```
 //**********************************************
 // ein Reset ausführen
 //**********************************************
    public void myreset()
    {
    for (int i = 0;i<40;i++)
        {
        TBand[i].setText("#");
        }
        Zustand='0';          // Anfangszustand = Null
        Kopfbewegung='H';     // Anfangswert= Halt
        Nachricht.setText("Eine einfache Turingmaschine");
```

© Hans-Georg Beckmann 2005

```
        Zustandsstring="--";   // erst nach dem Finden der Startposition wird daraus ein S0
        repaint();
    }// Ende von myreset
//*****************************************************
// Die Buttons abfragen
//*****************************************************
    public void actionPerformed (ActionEvent e)
    {
    Object  welcher;
    welcher = e.getSource();
    //*******************************
    if(welcher==reset)
        {
          myreset();
        }
    //*******************************
    if(welcher==run)
        {
        while(Zustand != 'e')    // solange noch nicht im Endzustand ....
            {
            Lesezeichen=TBand[pos].getText().charAt(0);
            u();
            Zustand=Folgezustand;
            Nachricht.setText("Zeichen "+Lesezeichen +" an Stelle "+pos +"Gefunden");
            repaint();
            }// ende von while
        }// Ende von if...

    //*******************************
    if(welcher==step)
        {
        Lesezeichen=TBand[pos].getText().charAt(0);
        u();
        Nachricht.setText(" Zeichen "+Lesezeichen +" an Stelle "+pos +" gefunden und Zeichen "
        + Schreibezeichen+" geschrieben");
        Zustand=Folgezustand;
        repaint();
        }
    //*******************************
    if(welcher==start)
        {
        Startposition_suchen();
        Nachricht.setText("Die Startposition ist bei "+pos);
        repaint();
        }
    } // Ende von actionPerfomed
```



Aufgabe: "Bauen" sie sich eine virtuelle Turingmaschine in Java, mit der sie einfache Turingautomaten simulieren können.

© Hans-Georg Beckmann 2005

Weitere einfache Übungen

Aufgabe:

Auf einem Turingband ist eine Zahl *n* dargestellt. Eine Turingmaschine soll daraus *n+1* machen und dann anhalten.

Lösung:

Man hat nachstehendes Bild zu erwarten:

Start (n = 22)

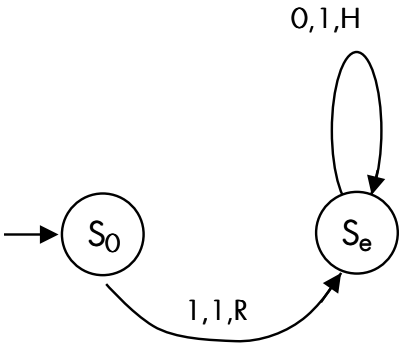
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0

Ende (n = 23)

0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

Der Transitionsgraph ist recht einfach. In der Startposition findet der Automat in jedem Fall eine "1" an der Kopfposition vor, daher kann man sich hier den Fall sparen, dass in Zustand S_0 eine "0" gelesen wird. Man könnte im Zustand S_e noch den Fall (1,1,H) angeben, der eintritt, nachdem auf dem Band die "0" durch eine "1" ersetzt worden ist.

Eine Automatentafel braucht man bei dieser einfachen Lösung nicht.



Aufgabe:

Auf einem Band stehen zwei Zahlen (n und m). Die weiter links stehende Zahl n soll um 1 erhöht werden, so dass am Ende n+1 und m auf dem Band stehen. Der Kopf der Turingmashine soll am Ende wieder auf der "1" am rechten Ende der Zahl m stehen.

Lösung:

Wieder ist es sinnvoll, das Band vor und nach der Arbeit der Maschine zu betrachten.

Start (n = 8 , m = 12)

0	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0

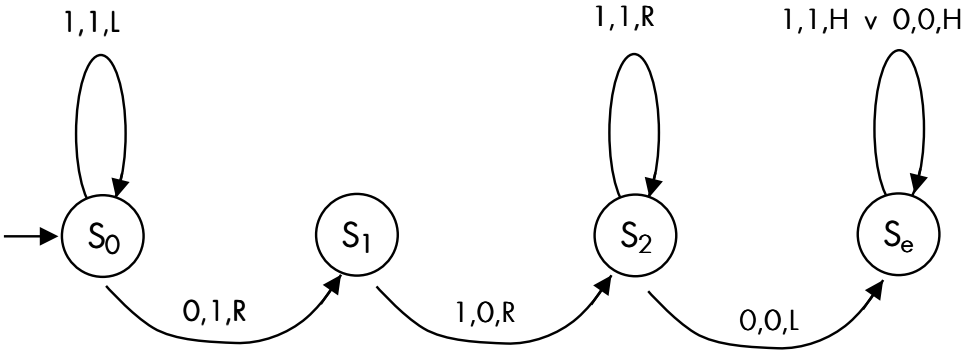
Ende (n = 9, m = 12)

0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0

© Hans-Georg Beckmann 2005

Die Lösung kann man zuerst in einfache Worte fassen: *"Gehe nach links bis zur Null. Ersetze die Null durch eine Eins. Gehe einen Schritt nach rechts und ersetze die Eins durch eine Null. Gehe ganz nach rechts auf die erste Null nach der Einsergruppe. Schreibe dort eine Eins und halte an.*"

Der Transitionsgraph:



Die Automatentafel:

Zustand	Eingabe-zeichen	Ausgabe-zeichen	Folge-zustand	Kopf-bewegung
S0	1	1	S0	L
S0	0	1	S1	R
S1	1	0	S2	R
S1	0	0	S2	R
S2	1	1	S2	R
S2	0	0	Se	L
Se	1	1	Se	H
Se	0	0	Se	H

nach Aufgabenstellung nicht möglich

Bei der letzten Aufgabe ist vielleicht schon deutlich geworden, wie nett unsere Art der Zahlendarstellung ist, da eine Zahl n durch n+1 Einsen angegeben wird.

Betrachten sie daher die folgende Aufgabe als eine kleine Gemeinheit.

Aufgabe (unfreundlich):

Eine Turingmaschine hat die zulässigen Zeichen { 0, 1, # }, wobei das Zeichen "#" für "blank" - also Leerzeichen steht. In dieser Maschine seien Zahlen als Dualzahlen kodiert. Zwei Zahlen auf dem Band werden durch ein "# " getrennt. In der Startposition steht der S/L-Kopf ganz rechts auf dem letzten Zeichen (0 oder 1) vor dem "#".

© Hans-Georg Beckmann 2005

*Baue eine Maschine, die eine Zahl **n** um 1 erhöht und dann wieder auf dem letzten Zeichen vor dem "#" stehen bleibt.*

Zur Veranschaulichung: Aus der Zahl 11 soll die Zahl 12 werden:

Start (n = 11)

#	1	0	1	1	#	#	#	#	#	#	#	

Ende (n = 12)

#	1	1	0	0	#	#	#	#	#	#	#	

Das sieht für diesen speziellen Fall doch einfach aus - oder ?

Hier müssen nur drei Zeichen verändert werden. Was , wenn es z.B. die Startzahl 8 gewesen ist ? Dann hat man nur eine Stelle zu verändern.

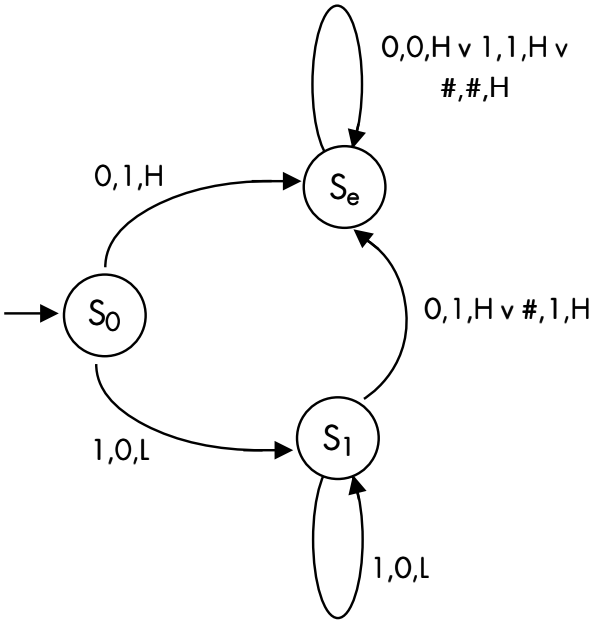
Was aber, wenn die Addition von 1 zu einer "Verlängerung" der Zahl führt , wie etwa bei n= 15 (1 1 1 1) und n+1 = 16 (1 0 0 0 0) ?

Für die Lösung muss man sich daran erinnern, dass hier eine Addition von Dualzahlen vorliegt. Diese Addition erfolgt Bitweise und kann mit und ohne Übetrag laufen (blättern sie weit, weit zurück zum Thema "Halbaddierer, Volladdierer"). Wir sollen nun also versuchen, eine solche Addition durch eine Turingmaschine ausführen zu lassen. Die wird nicht zu kompliziert sein, da wir nur eine "1" addieren müssen.

Zustand	Eingabe-zeichen	Ausgabe-zeichen	Folge-zustand	Kopf-bewegung
S0	1	0	S1	L
S0	0	1	Se	H
S1	1	0	S1	L
S1	0	1	Se	L
S1	#	1	Se	L
Se	1,0,#	1,0,#	Se	H

Neben einem Anfangszustand wird es einen Zustand geben, in dem eine Übetragseins gespeichert ist und einen Endzustand, der erreicht wird, wenn der letzte gespeicherte Übertrag zu einer Null addiert wurde.

Die Automatentafel kann auch wieder als Transitionsgraph dargestellt werden.



© Hans-Georg Beckmann 2005

Es bleibt bei dieser einfachen Lösung aber ein Problem, das deutlich wird , wenn man das Beispiel n= 15 betrachtet. Die Maschine macht dann aus einer vierstelligen Dualzahl eine 5-stellige Dualzahl und überschreibt das #-Zeichen, das links neben der Zahl n auf dem Band stand. Wenn weiter nach links weitere Zahlen auf dem Band plaziert sind, ist ein entscheidendes Trennzeichen verloren gegangen. Man muss in diesem Falle die neue Zahl (n+1) um eine Stelle nach rechts auf dem Band kopieren, wie es im Beispiel dargestellt ist.

Start

#	1	0	1	1	0	1	1	0	#	#	#	

Ende

#	1	0	1	#	1	0	1	1	0	#	#	

Auch dazu kann man eine Turingmaschine entwerfen, die das schafft. Aber auch hier ist zu beachten, dass zuerst der Fall betrachtet werden muss, bei dem die Zahl n+1 (die gerade durch Addition mit 1 entstanden ist) ganz rechts auf dem Band steht und nicht etwa noch weitere Zahlen rechts anschließen.

Folgeaufgabe:

Der S/L-Kopf einer Turingmaschine stehe an irgend einer Stelle mitten in der Dualdarstellung einer Zahl. Alles was von dieser Stelle an (inklusive der aktuellen Position) nach rechts hin steht, wird um einen Schritt nach rechts kopiert. An die Startposition wird ein #-Zeichen geschrieben.

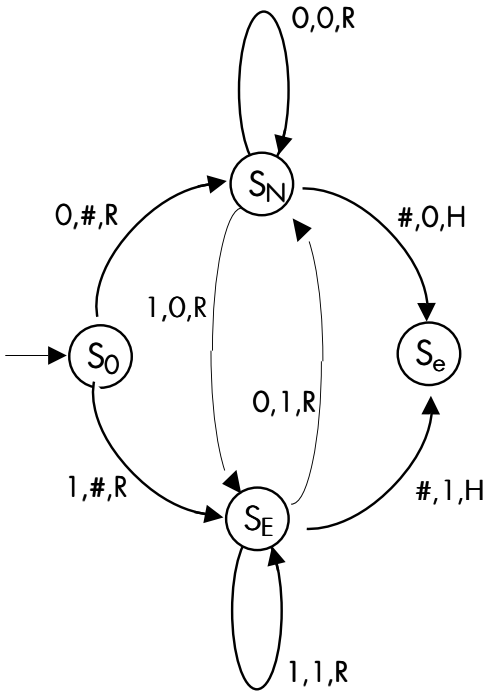
Lösung:

Man ahnt schon, dass es nun ein wenig aufwendiger wird, dieses Folgeproblem der einfachen Addition zu lösen.

Die Maschine wird gleich zu Beginn das #-Zeichen an die aktuelle Position schreiben, und sich merken müssen, ob dort eine "1" oder eine "0" stand. Dann muss der S/L-Kopf nach rechts wandern und von der rechten Seite her die Zeichen jeweils um eine Stelle nach rechts kopieren. Am Schluss muss das Zeichen, das zu Beginn mit " # " überschrieben wurde, eingesetzt werden.

Betrachten sie den folgenden Transitionsgraphen in Ruhe und stellen ihn als Maschinentafel dar.

© Hans-Georg Beckmann 2005



In diesem Transitionsgraphen sind beim Zustand Se Kombinationen aus Zustand und Eingabezeichen außer Acht gelassen worden, da sie nach Aufgabenstellung nicht vorkommen können.
Die Funktion des Automaten ist leicht geschildert. In einem Zustand S_N merkt er sich, ob er eine Null schreiben muss, die er gerade vom Band gelesen hat und die nun eine Stelle weiter rechts auf das Band geschrieben werden soll, und in einem Zustand S_E merkt er sich eine Eins, die zu schreiben ist.

Für Automten dieser Größe lohnt sich eine verbesserte Simulation durch ein Javaprogramm. Es sollte möglich sein, die Automatentafel komplett (aus einer Textdatei) einzugeben. Das Programm soll verschiedene Eingabealphabete verarbeiten können. Fehler in der Automatentafel sollten angezeigt werden. Das Programm soll einen *Step*modus haben, der eine schrittweise Kontrolle des Ablaufs sichert. Man soll ohne Neustart Änderungen an der Automtentafel und am Turingband durchführen können.
Das nachfolgende Javaprogramm erfüllt auf möglichst einfache Weise die Anforderungen und ist beliebig erweiterbar.
Der besondere Vorteil liegt darin, dass man Automatentafeln in einem beliebigen Texteditor eingeben , editieren und als Datei speichern kann. Dann wählt man den gesamten Text an, kopiert ihn in die Zwischenablage und setzt ihn im laufenden Applet in die Textarea ein, die für die Automatentafel vorgesehen ist.

© Hans-Georg Beckmann 2005

```
//  Turing4.java
//  Turing4
//  Hier nun die Version einer Textarea für die Maschinentafel
//  Es kommen einige Methoden zum Einsatz, die strings betreffen, wenn es darum geht,
//  aus dem Textarea die Stellen zu finden, die zum aktuellen Zustand und zum Eingabezeichen
//  der Turingmaschine passen
//  -------------------------------- Hinweise zur Automatentafel --------------------
// Die Automatentafel kann in  einem einfachen  Texteditor geschreiben und als Textdatei
// gesichert werden das erspart mühsame Tipperei.
// Da die Überführungsfunktion immer im Text nach dem Substring sucht, der aus
// Zustand + Komma + Eingabezeichen besteht und von der Fundstelle an weiter gezählt wird,
// kann man es sich nicht leisten, den Zustandsstring einmal zwei und ein andermal drei
// Zeichen lang zu machen. Daher : Immer dreistellig
// also S00 und S01 und S02 ...... und S12, S13 usw. bis Sen für den Endzustand.
// In den Zeilen der Automatentafel muss folgende Reihenfolge eingehalten werden:
// Zustand, Eingabezeichen, Ausgabezeichen, Folgezustand, Kopfbewegung
// Das ist wichtig, weil sonst ein Substring aus Zustand und Eingabezeichen einem anderen
// Substring aus  Folgezustand und Ausgabezeichen gleichen könnte, das wäre fehlerträchtig.

import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Turing4 extends Applet implements ActionListener
{
    Button   step    = new Button ("Step");
    Button   reset   = new Button ("Reset");
    Button   run     = new Button ("Run");
    Button   start   = new Button ("Startposition");   // S/L- Kopf in Starposition bringen
    String   Zustand,Folgezustand;          // Zustand und Folgezustand als String vereinbart
    String   Lesezeichen,Schreibezeichen;   // eingelesenes Zeichen, ausgegebenes Zeichen
    String   Kopfbewegung;                  // kann L oder R oder H sein
    TextField[] TBand=new TextField[40];    // für den Bildschirm das Turingband
    int      pos;                           // die Position des S/L-Kopfes
    int      Stelle;                        // Stelle in der Automatentafel
    Label    Nachricht   = new Label("");   // allerlei Mitteilungen am Bildschirm
    Label    Nachricht2  = new Label("");
    Label    Nachricht3  = new Label("");
    Label    Ueberschrift= new Label(" Eine einfache Turingmaschine ",Label.CENTER);

    // TextArea für die Automatentafel
    TextArea Automatentafel = new TextArea("- Die Automatentafel - \n
                                            \n",100,20,TextArea.SCROLLBARS_VERTICAL_ONLY);

    public void init()
    {
       setLayout (null);
       for (int i = 0;i<40;i++)
        {
        TBand[i]=new TextField();
        TBand[i].setBounds(20+i*16,70,18,15);
        TBand[i].setBackground(new Color(180,180,255));
        add(TBand[i]);
        TBand[i].setText("#");       // am Anfang mit Nullen oder mit # füllen
        }
       step.setBounds(50,150,50,26);
       add(step);
       step.addActionListener(this);

       reset.setBounds(110,150,50,26);
       add(reset);
       reset.addActionListener(this);

       run.setBounds(170,150,50,26);
       add(run);
```

```
        run.addActionListener(this);

        start.setBounds(230,150,150,26);
        add(start);
        start.addActionListener(this);

        Nachricht.setBounds(50,190,350,26);
        add(Nachricht);
        Nachricht2.setBounds(50,218,350,26);
        add(Nachricht2);
        Nachricht3.setBounds(50,246,350,26);
        add(Nachricht3);
        Ueberschrift.setBounds(20,20,700,35);
        Ueberschrift.setFont(new Font("Serif",1,20));
        add(Ueberschrift);

        Zustand="S00";            // Anfangszustand = Null
        Kopfbewegung="H";         // Anfangswert= Halt
        Nachricht.setText("Eine einfache Turingmaschine");
        Zustand="---";            // noch nicht festgelegt, wird dann aber S0 werden

        Automatentafel.setBounds(450,150,200,360);
        Automatentafel.setFont(new Font("Sansserif",1,14));
        add(Automatentafel);
        Stelle=0;
    }

    public void paint (Graphics g)
    {
    // malt ein Polygon an die passende Position und beschriftet es mit akt. Zustand
    int x=30+pos*16;
    int y=95;
    g.setFont(new Font("SansSerif",1,16));
    int[] arx ={x,x+6,x+26,x+26,x-26,x-26,x-6,x};
    int[] ary={y,y+6,y+6,y+30,y+30,y+6,y+6,y};
    g.setColor(new Color(250,180,180));
    g.fillPolygon(arx,ary,8);
    g.setColor(Color.black);
    g.drawPolygon(arx,ary,8);
    g.drawString(Zustand,x-13,y+24);
    }
```

In der Überführungsfunktion kommen nun einige Methoden zur Stringbehandlung zum Einsatz.

Da die Automatentafel eine TextArea ist, kann man deren gesamten Inhalt mit *Automatentafel.getText()* in einen String holen. Mit der Methode *indexOf(String str)* kann man einen String darauf untersuchen, ob er den Teilstring *str* enthält. Wenn ja, bekommt man die Stelle angegeben, an der *str* im String beginnt. Sucht man also nach Zustand S2 mit Eingabezeichen "1", dann erzeugt man den Suchstring *Zustand + "," + Lesezeichen = " S2,1 "* , wobei Lesezeichen, das vom Turingband eingelesene Zeichen sein soll. Danach wird durch *Stelle = automatenstring.indexOf(suchstring)* die passende Stelle gefunden. Sollte der *suchstring* nicht enthalten sein, wird als Ergebnis -1 ausgegeben, was wir als Fehler interpretieren. Hat man nun so die richtige Stelle (Zeile) der Automatentafel gefunden, muss man nun noch Ausgabezeichen (Schreibezeichen), Folgezustand und Kopfbewegung finden, die sich in der selben Zeile der Automatentafel befinden müssen. Das geschieht nun mit der Methode *substring(int start, int ende)*. Diese Methode kopiert aus einem String einen Teil heraus, der beim Index *start* beginnt und beim Index *ende* endet.

© Hans-Georg Beckmann 2005

Eine Zeile sieht z.B. wie folgt aus:

S02,1,0,Sen,H

Das Ausgabezeichen befindet sich beispielsweise 6 Stellen weiter, als das erste Zeichen der Zeile.

Der Folgezustand befindet sich von Stelle 8 bis Stelle 11 vom Anfang entfernt.

```
//**************************************************
// Die Überführungsfunktion erledigt alles
// Eingabezeichen lesen, Stelle suchen mit Zeichen
// und passendem Zustand dann in dieser Stelle
// Kopfbewegung, zrückgeschriebenes Zeichen und
// Folgezustand ermitteln
//**************************************************
  public void u()
  {
  String suchstring,automateninhalt;  // Automateninhalt ist der String der ganzen Textarea
  Lesezeichen=TBand[pos].getText();   // Eingabezeichen holen
  suchstring=Zustand+","+Lesezeichen; // Das ist Zustand + Komma + Eingabezeichen
  automateninhalt=Automatentafel.getText();    // Inhalt der Automatentafel holen
  Stelle=automateninhalt.indexOf(suchstring);  // .. wo beginnt der passende Eintrag ?
      // Hier eine Kontrollausgabe im Nachrichtenlabel
  Nachricht.setText("Suchstring : "+suchstring+" . Gefunden an Stelle "+Stelle);
      // Was ist , wenn es ein Fehler ist ?
  if(Stelle==-1) Nachricht.setText("FEHLER - Eintrag in der Automtentafel fehlt ");
  else
  {
      Schreibezeichen=automateninhalt.substring(Stelle+6,Stelle+7); // selbst nachzählen !
      Folgezustand=automateninhalt.substring(Stelle+8,Stelle+11);
      Kopfbewegung=automateninhalt.substring(Stelle+12,Stelle+13);
      TBand[pos].setText(Schreibezeichen);
      if (Kopfbewegung.equals("L")) pos=pos-1;
      if (Kopfbewegung.equals("R")) pos=pos+1;
      Zustand=Folgezustand;
      // Falls Kopfbewegung H sein sollte , ändert sich pos nicht
  }// Ende von else
   Nachricht2.setText("Folgezustand ist : "+Zustand+" . Ausgabezeichen
                                                    ist "+Schreibezeichen);
   Nachricht3.setText("Kopfbewegung ist "+ Kopfbewegung);
}// Ende von u

//**************************************************
// Die Startposition einnehmen
// in dieser Maschine wird die Null oder
//  die eins ganz rechts gesucht
//**************************************************
  public void Startposition_suchen()
  {
  int i=1;
  boolean gefunden=false;
  while ((gefunden==false)&&(i<39))
  {
  if((TBand[i].getText().equals("#")) && (TBand[i+1].getText().equals("#")))
   gefunden=true;
   else
   i=i+1;
   }// Ende von while
  pos=i-1;
  }

//**************************************************
// ein Reset ausführen
//**************************************************
  public void myreset()
  {
  for (int i = 0;i<40;i++)
      {
      TBand[i].setText("#");
```

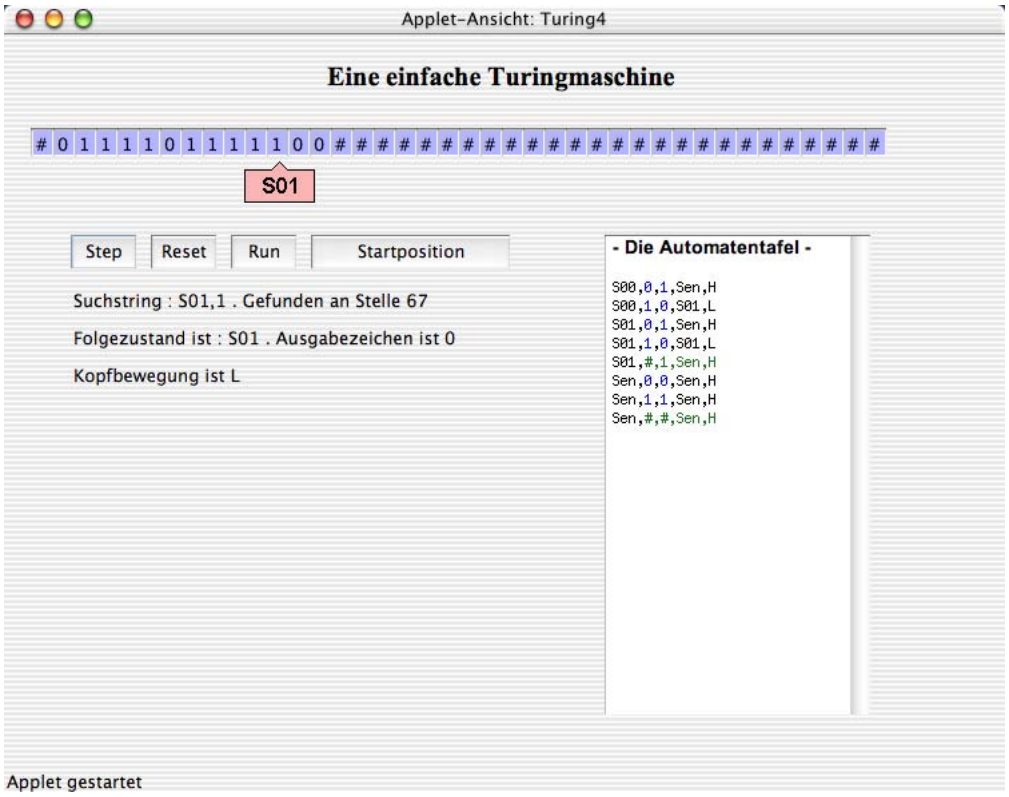
© Hans-Georg Beckmann 2005

```
        }
        Zustand="---";          // Anfangszustand kommt noch
        Kopfbewegung="H";       // Anfangswert= Halt
        Nachricht.setText("Eine einfache Turingmaschine");
        Stelle=0;
        repaint();
    }// Ende von myreset

  //******************************************************
  // Die Buttons abfragen
  //******************************************************
    public void actionPerformed (ActionEvent e)
    {
    Object   welcher;
    welcher=e.getSource();
    //*********************************
    if(welcher==reset)
        { myreset(); }
    //*********************************
    if(welcher==run)
        {
        while(Zustand.equals("Sen")==false)
         {
          u();
          repaint();
         }// ende von while
        }// ende von if...

    //*********************************
    if(welcher==step)
        {
        u();
        repaint();
        }
    //*********************************
    if(welcher==start)
        {
        Startposition_suchen();
        Zustand="S00";
        Nachricht.setText("Die Startposition ist bei "+pos);
        Stelle=0;
        repaint();
        }
    }
} // Ende von applet
```

Wenn dann alles geklappt hat, sieht das Ergebnis schon ganz beachtlich aus:

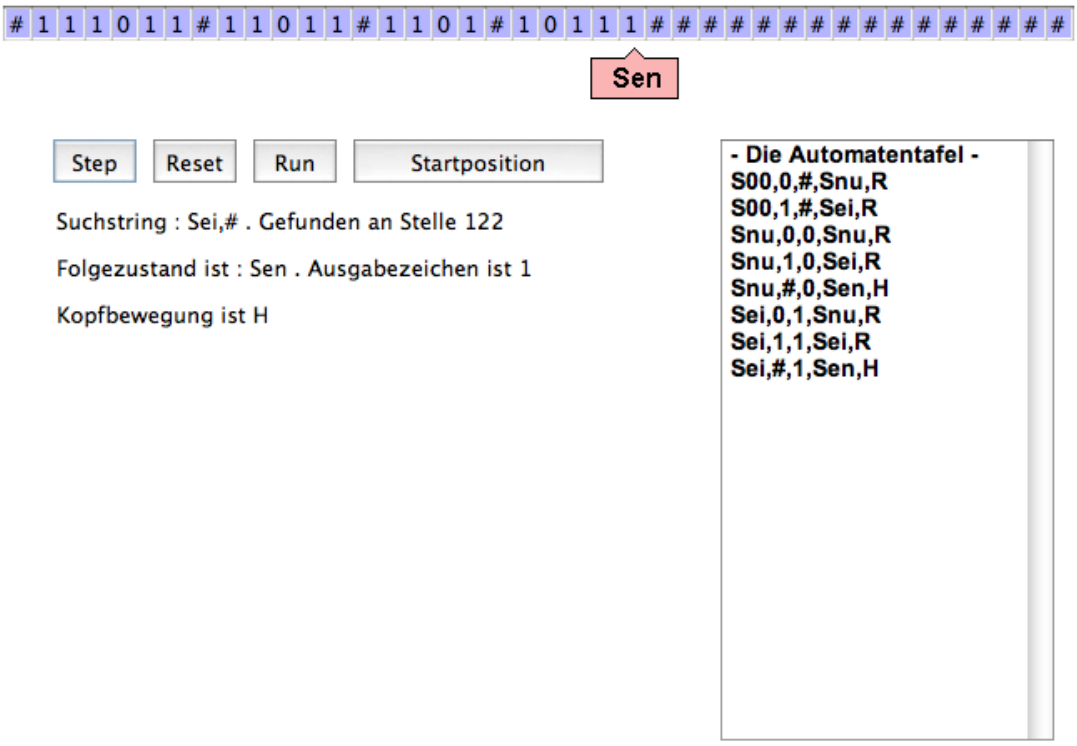
Nebenstehend ein schon beschriftetes Band jedoch noch ohne Automatentafel.

© Hans-Georg Beckmann 2005



Der oben geschilderte *Plus- Eins-Automat* während des Arbeitens

Mit diesem Programm kann man es nun auch wagen, den Transitionsgraphen des *"Verschiebe um eine Stelle nach rechts "* - Automaten (Seite 13) in eine Tabelle zu übertragen und laufen zu lassen.



© Hans-Georg Beckmann 2005

Nach diesem Ausflug in Maschinen, deren Bandbeschriftung z.B. mit Dualzahlen realisiert ist, kehren wir zu unserer ursprünglichen Codierung zurück, die zum Darstellen der Zahlen nur Ketten von Einsen kennt, und bei der das Trennzeichen die Null ist.

Beispiel: Eine Kopiermaschine **K**

Eine Turingmaschine K soll eine Gruppe von Einsen nach rechts kopieren.

Start

0	1	1	1	1	0	0	0	0	0	0	0	

Ende

0	1	1	1	1	0	1	1	1	1	0	0	

Die Maschine hat die übliche Startposition und soll am Ende auf der letzten Eins der kopierten Gruppe rechts außen stehen.

Die Arbeit der Maschine lässt sich in einem Struktogramm erklären:

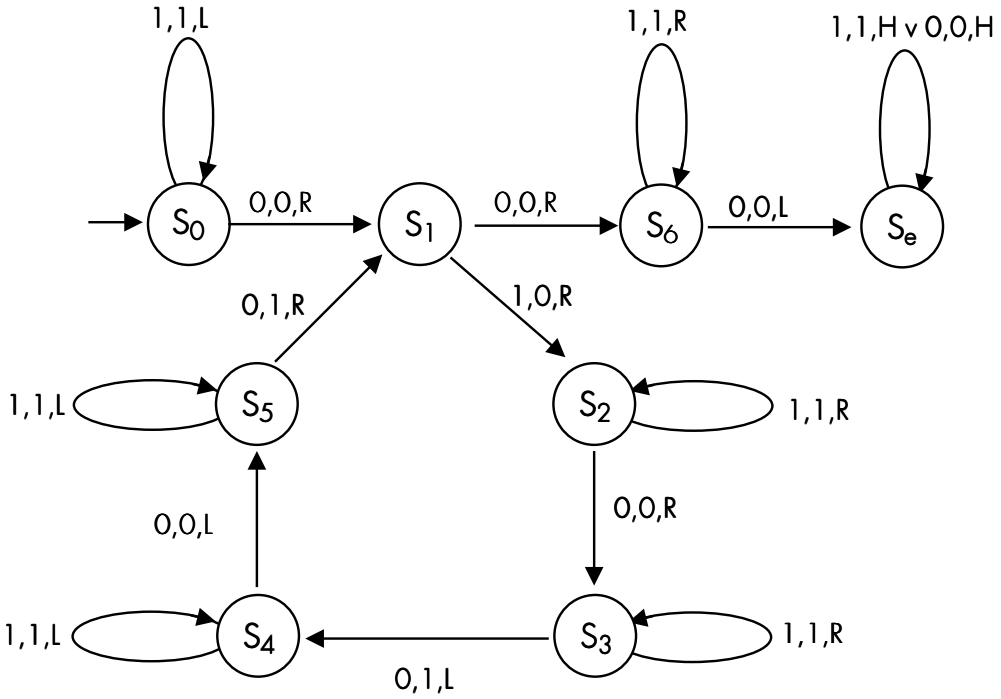
Gehe nach links bis zur ersten 0		
wdh	Gehe einen Schritt nach rechts	
	Steht dort eine 1 ?	
	Ja	Nein
	Markiere diese Stelle mit einer 0	Schreibe 0
	Gehe nach rechts bis zur 0	Gehe nach rechts
	Gehe noch mal nach rechts bis zur nächsten 0	Gehe nach rechts bis zur 0
	Schreibe eine 1 auf das Band	Gehe einen Schritt nach links
	Gehe nach links bis zur 0	Ende
	Gehe nach links bis zur nächsten 0	
	Schreibe eine 1	
bis	Ende	

Insgesamt arbeitet die Maschine etwa so,wie der Automat , der auf Seite 13 zum Verschieben einer Gruppe auf dem Band benutzt wurde.

Man muss die jeweils zu kopierende 1 mit einer 0 markieren, die Kopie an ihren Platz schreiben, zurückkehren, die Markierung wieder entfernen (indem man eine 1 zurückschreibt). Dann geht man um einen Schritt nach rechts und wiederholt den Vorgang solange, bis die ganze Gruppe abgearbeitet ist.

© Hans-Georg Beckmann 2005

In einem Transitionsgraphen wird das noch mal deutlich.[3]



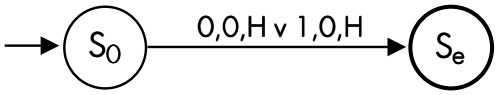
Aufgabe:

Stellen sie für diese Maschine die Automatentafel auf, und lassen sie die Maschine im Simulationsprogramm laufen.

Elementarmaschinen

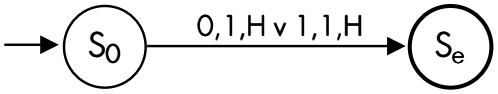
An den letzten Beispielen sieht man schon, dass die Entwicklung von Turingmaschinen recht unübersichtlich werden kann. Zur Vereinfachung "erfinden" wir jetzt Maschinen, aus denen andere, komplexere Maschinen zusammengesetzt werden können.

Die Nullmaschine **0**



- schreibt eine Null an der aktuellen Position auf das Band

Die Einsmaschine **1**



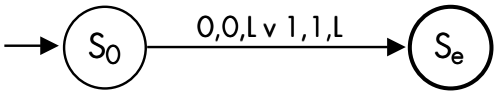
- schreibt eine Eins an der aktuellen Position aus das Band

[3] aus "Automaten, Schaltwerke, Sprachen" von E. Modrow, Dümmler 1987 Seite 184

© Hans-Georg Beckmann 2005

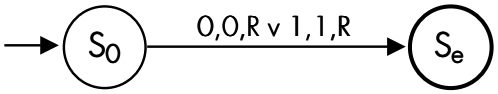
Die kleine Linksmaschine **l**

- bewegt den Kopf um eine Stelle nach links und verändert dabei die Bandbeschriftung nicht.



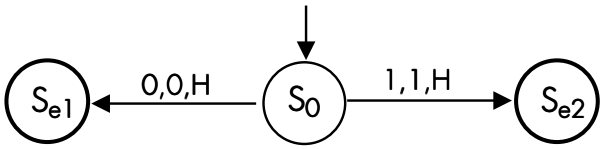
Die kleine Rechtsmaschine **r**

- bewegt den Kopf um eine Stelle nach rechts und verändert die Bandbeschriftung nicht.



Die Prüfmaschine **p**

- kann zwei Endzustände einnehmen, die vom Eingabezeichen abhängig sind.

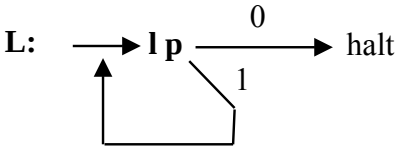


Aus diesen Elementarmaschinen lassen sich nun im nächsten Schritt weitere Maschinen bauen.

Einfache zusammengesetzte Turingmaschinen

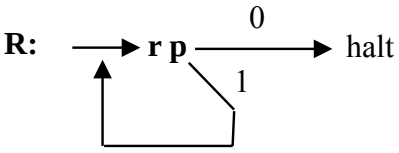
Die große Linksmaschine **L**

- geht auf die erste Null links von einer Gruppe von Einsen (aber immer mindestens eine Stelle nach links).



Die große Rechtsmaschine **R**

- geht auf die erste Null rechts von einer Gruppe von Einsen (aber immer mindestens eine Stelle nach rechts).

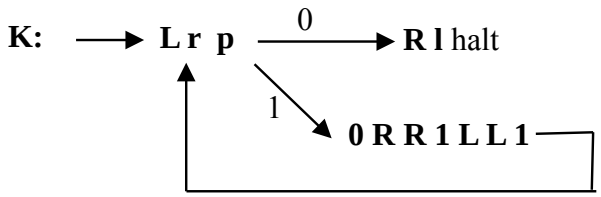


Hinweis: Hinter der Prüfmaschine stehen an den Pfeilen 0 und 1. Das sind die Eingabezeichen, die nichts mit der Nullmaschine oder der Einsmaschien zu tun haben.

© Hans-Georg Beckmann 2005

Die Kopiermaschine **K**

- kopiert eine Gruppe von Einsen nach rechts. Es wird eine trennende Null zwischen Original und Kopie gesetzt. Am Ende steht der Kopf auf der am weitesten rechts stehenden 1. Am Anfang muss rechts neben der zu kopierenden Gruppe Platz auf dem Band sein.



Aufgabe:

Konstruiere eine Addiermaschine **A**, die zwei Zahlen, die auf dem Band hintereinander stehen, addiert. Nach der Addition sind die Summanden verschwunden, es steht nur noch das Ergebnis auf dem Band.

Start (3 und 5)

0	1	1	1	1	0	1	1	1	1	1	1

Ende (8)

0	1	1	1	1	1	1	1	1	1	0	0	

Lösung:

Die Maschine muss die trennende Null durch eine Eins ersetzen und dann rechts außen zwei Einsen wegnehmen.

A : L 1 R l 0 l 0 l

Aufgabe:

Eine Kopiermaschine **K2** soll eine Gruppe von Einsen über eine andere Gruppe hinweg nach rechts auf das Band kopieren.

Aufgabe:

Eine Addiermaschine A2 soll zwei Zahlen addieren. Am Ende sollen die beiden Summanden und die Summe auf dem Band stehen.

Aufgabe:

Eine Löschmaschine X soll eine Gruppe von Einsen vom Band löschen.

© Hans-Georg Beckmann 2005

Aufgabe:

Eine Multipliziermaschine soll das Produkt zweier Zahlen a und b auf das Band schreiben. Dabei steht die Zahl a links von der Zahl b und am Ende sollen a und das Produkt a*b auf dem Band stehen.

Lösung:

a = 3 und b= 3)

0	1	1	1	1	0	1	1	1	1	0	0

0	1	1	0	1	0	1	1	1	1	0	1	1	1	1	0	0	0	0	

Die dritte 1 von a ist zur Markierung durch eine 0 ersetzt. Die Zahl b ist zum ersten Male mit einer Kopiermaschine kopiert worden.

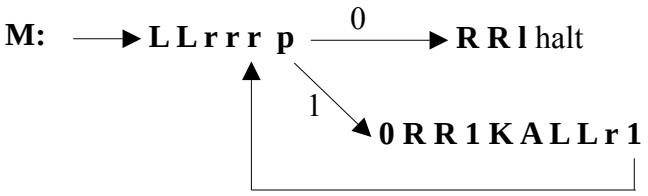
0	1	1	1	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	

0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	0	0	0	0	

0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	

Die Zahl b und die Kopie von b werden mit der Addiermaschine addiert. Dann wird die alte Markierung rückgängig gemacht. Nun folgt ein Schritt nach rechts. Findet sich dort noch eine 1, dann ist a noch nicht ganz abgearbeitet und es beginnt von vorn. Markieren , kopieren.
Ist a ganz abgearbeitet, findet die Maschine nur noch die trennende 0.
Nun muss nur noch mit einer großen Rechtsmaschine und einer kleinen Linksmaschine der Endzustand eingenommen werden.

In diesem Beispiel wird die Multiplikation auf eine Addition zurückgeführt. Ist z.B. die Rechnung (für a=3 und b=3) 3 • 3, dann entspricht das 3 + 3 + 3 . Da die Zahl b schon einmal auf dem Band steht, muss sie nur noch zweimal hinzu addiert werden. Da a durch (a+1) Einsen auf dem Band dargestellt ist, man aber nur noch (a +1) – 2 mal die Zahl b rechts hinzuaddieren muss, beginnt die Maschine mit dem Abarbeiten der Zahl a auf der dritten Eins der Zahl a.



Aufgabe:

Eine Multipliziermaschine **M2** soll zwei Zahlen a und b multiplizieren. Am Ende sollen a, b und das Produkt a • b auf dem Band stehen.

© Hans-Georg Beckmann 2005

Aufgabe:

Eine Löschmaschine **Z** soll von der aktuellen Position nach links hin alle Einsen bis zur nächsten Null hin löschen. Über dieser Null bleibt die Maschine stehen.

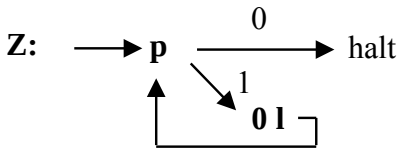
Start

0	1	1	1	1	0	1	1	1	1	1	1	

Ende

0	1	1	1	1	0	0	0	0	1	1	0	

Lösung:



Aufgabe:

Eine Linksschiebemaschine $\mathbf{V_L}$ soll eine Gruppe von Einsen, die durch mehrere Nullen von einer anderen Gruppe von Einsen getrennt ist, nach links an diese Gruppe heranschieben. Am Ende sollen beiden Einsergruppen eine zusammenhängende Einsergruppe sein. Zu Beginn stehe der Kopf auf der Eins ganz rechts in der Gruppe, die verschoben werden soll. Am Ende stehe der Kopf auf der Eins ganz rechts in der Gesamtgruppe.

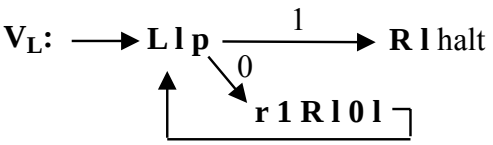
Start

0	1	1	1	0	0	0	0	1	1	1	1	

Ende

0	1	1	1	1	1	1	1	0	0	0	0	

Lösung:



Nun haben wir eine ganze Reihe von Maschinen zusammengestellt, die alle auf den elementaren Turingmaschinen basieren. Mithilfe dieser Maschinen werden nun komplexere Automaten gebaut, die z.B. in der Lage sind, rekursive Funktionen zu berechnen. Das ist mit den vorher behandelten endlichen Automaten nicht möglich gewesen.

© Hans-Georg Beckmann 2005

Primitiv rekursive Funktionen und Turingmaschinen

Im Abschnitt über Rekursionen wurden schon die "primitiv rekursiven Funktionen " kurz erläutert.
Wir brauche sie jetzt noch einmal. Bevor wir mit ihnen arbeiten, sei noch mal die *Churchsche These* zitiert:

" Jede berechenbare Funktion ist rekursiv."

Wir beschränken uns dabei auf die folgenden Grundfunktionen:

1. Die Nachfolgefunktion $N(x) = x + 1$
2. Die Konstantenfunktion $K_c = c$
3. Die Projektionsfunktion, die aus einer Anzahl n von Argumenten ein Argument i mit $1 \leq i \leq n$ als Funktionswert ergibt $P_i(x_1, \ldots\ldots x_n) = x_i$

Als einfaches Beispiel für eine rekursive Funktion hatten wir schon die Summe zweier Zahlen x und y behandelt. Es soll die Summenbildung auf die Nachfolgefunktion zurückgeführt werden.

$$sum(y, x) = N(sum(y-1, x)) = N(N(sum(y-2, x))) = \ldots\ldots \text{ u.s.w}$$

Das Verfahren muss aber auch abbrechen, wenn das erste Argument der Funktion *sum* bei Null angekommen ist. Es gilt also : $sum(0, x) = P2(0, x) = x$

Als Zahlenbeispiel:

$$sum(3,4) = N(sum(2,4)) = N(N(sum(1,4))) = N(N(N(sum(0,4)))) = N(N(N(4))) = N(N(5)) = N(6) = 7$$

$$\boxed{sum(0, 4) = P2(0, 4) = 4}$$

Allgemein gefasst:

Ein Rekursionsanfang (oder vielleicht besser eine Abbruchbedingung) ist

$$f(0, x_2, \ldots\ldots, x_n) = g(x_2, \ldots\ldots, x_n)$$

Das bedeutet, dass für den speziellen Fall, in dem das erste Argument der Funktion f Null ist, der berechenbare Funktionswert durch eine Funktion g aus den restlichen Argumenten ermittelt wird.

Die eigentliche Rekursion besteht dann darin, dass ein Funktionswert für f, bei dem das erste Argument y nicht Null ist, durch eine Funktion h bestimmt wird, deren erstes Argument wiederum auf dem Wert der Funktion f beruht, bei dem das erste Argument der Vorgänger von y ist.

$$f(y, x_2, \ldots\ldots, x_n) = h(f(y-1, x_2, \ldots\ldots, x_n), x_2, \ldots\ldots\ldots, x_n)$$

© Hans-Georg Beckmann 2005

Das soll noch mal am Beispiel der Multiplikation *prod* (y , x) klar gemacht werden:

1) Rekursionsanfang : *prod* (0, x) = K_0 = 0 (Konstantenfunktion)

2) *prod* (y , x) = sum (*prod* (y–1,x),x)

Hier wurde die Multiplikation durch eine rekursive Addition ersetzt.

Zahlenbeispiel:

prod (2,4) = *sum* (*prod* (1,4),4) = *sum*(*sum*(*prod*(0,4),4),4) = *sum* (*sum* (K_0,4),4) = *sum* (4,4) = 8

Es sollen nun Turingmaschinen konstruiert werden, die alle notwendigen Operationen für diese primitiv rekursiven Funktionen ausführen können.

Die Nachfolgemaschine N erzeugt zu einer Zahl, die durch eine Gruppe von Einsen dargestellt ist die Nachfolgezahl. Zu Beginn stehe der Kopf auf der "rechtesten" Eins.

N : **r 1**

Offensichtlich reichen eine kleine Rechtsmaschine und eine Einsmaschine hier aus.

Die Konstantenmaschine erzeugt eine feste Zahl als Ergebnis. Sie löscht entweder zwei Argumente, die noch auf dem Band stehen und ersetzt sie durch eine Zahl c (also c +1 Einsen) oder sie löscht nur ein Argument auf dem Band und schreibt dafür c +1 Einsen.

z.B. soll $K_{1,1}$ bedeuten, dass die Maschine *eine* Zahl vom Band löscht und dann die Zahl c = 1 (repräsentiert durch zwei Einsen) auf das Band schreibt.

$\mathbf{K_{1,1}}$: **Z r 1 r 1**

löschen — Zwei Einsen schreiben

Und $K_{3,2}$ löscht *zwei* Argumente vom Band und schreibt die Zahl 3 (also 4 Einsen) auf das Band.

$\mathbf{K_{3,2}}$: **Z Z r 1 r 1 r 1 r 1**

Und ähnlich die Maschine , die zwei Zahlen löscht und die 6 aufs Band schreibt:

$\mathbf{K_{6,2}}$: **Z Z r 1 r 1 r 1 r 1 r 1 r 1 r 1**

Damit bekommt jede Variante der Konstatenfunktion eine eigene Turingmaschine, was zuerst einmal etwas ungewöhnlich erscheint.

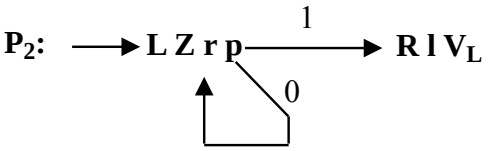
© Hans-Georg Beckmann 2005

Um die Beispiele nicht zu kompliziert zu machen, werden wir uns auf Funktionen mit nur zwei Argumenten beschränken. Damit kann man dann auch die Projektionsfunktion noch recht übersichtlich als Turingmaschinen angeben. Die Projektionsfunktion wird eine der beiden Zahlen (Argumente) vom Band löschen und die andere Zahl stehen lassen. Es könnte aber sein, dass die übrig gebliebene Zahl nach links verschoben werden muss.

Die Maschine $\mathbf{P_1}$ behält das erste Argument (weiter links) und löscht das zweite vom Band.

$$\mathbf{P_1 : Z\ l}$$

Die Maschine P_2 behält das zweite Argument und löscht das erste Argument vom Band.



Wir lassen bei dieser Maschine das Problem ausser Acht, das sich ergibt, wenn die Verschiebemaschine $\mathbf{V_L}$ in Richtung Bandanfang schieben sollte. Dort haben wir nach unserer Anfangsfestlegung eine Null als Start stehen. Die Maschine $\mathbf{V_L}$ ist aber so konstruiert, dass sie nach links an eine dort stehende Gruppe von Einsen heranschiebt. Es sind also möglicherweise Schwierigkeiten zu erwarten.

Wir werden an ein Beispielen untersuchen, wie unsere Maschinen arbeitet und ob Probleme auftreten werden oder nicht.

Beispiel 1: Eine Turingmaschine soll $\mathbf{P_2(4, N(2))}$ berechnen. Das heißt , die Maschine soll das zweite Argument halten, das in diesem Fall aus dem Nachfolger der Zahl 2 besteht. Die beiden Zahlen 4 und 2 stehen am Anfang auf dem Band:

Start

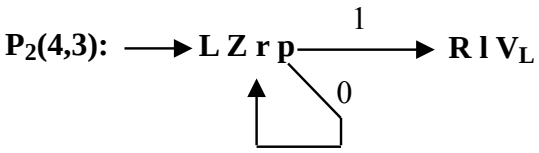
0	1	1	1	1	1	0	1	1	1	0	0

Zuerst die Nachfolgefunktion

N(2) : r 1

0	1	1	1	1	1	0	1	1	1	1	0

Dann kommt die Projektionsfunktion

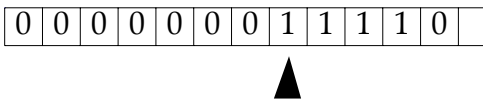


Die große Linksmaschine und die anschließende Löschmaschine erzeugen als Zwischenergebnis:

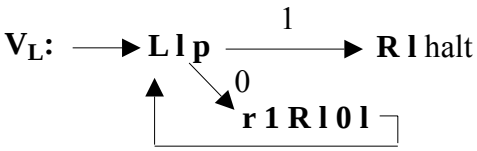
0	0	0	0	0	0	0	1	1	1	1	0

© Hans-Georg Beckmann 2005

Danach läuft eine Wiederholschleife, die den S/L-Kopf solange nach rechts führt, wie noch Nullen auf dem Band vorgefunden werden. Die Schleife wird verlassen, wenn die erste Eins des zweiten Arguments erreicht ist.



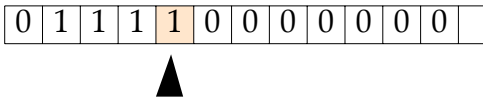
Die große Rechtsmaschine führt auf die Null rechts neben der Einsergruppe. Die kleine Linksmaschine setzt den S/L-Kopf auf die Eins ganz rechts. Nun arbeitet die Verschiebemaschine $\mathbf{V_L}$, die nun noch einmal genau betrachtet werden muss. Die Untermaschinen **L** und **l** führen auf die zweite Null links neben die Einsergruppe. Dort findet die Maschine eine Null vor und die Folge **r 1 R l 0 l** fügt nun eine Eins links an die Einsergruppe an und nimmt aus dieser Gruppe ganz rechts eine Eins weg.



Und hier wird deutlich, dass die Maschine $\mathbf{V_L}$ am Bandanfang nicht funktionieren kann, da dort nie eine Eins gefunden wird. Die würde man aber brauchen, damit die Folge **R 1 halt** erreicht wird und die Maschine anhält. Was ist zu tun ?

Die einfachste Möglichkeit besteht darin, unsere Anfangsverabredungen über die Startbeschriftung des Turingbandes zu ändern. Als Startsymbol ganz links außen nehmen wir eine "1", die nicht als Zahl Null interpretiert wird, und die auch nicht als Argument für Berechnungen herangezogen werden kann. Sie kann aber dazu dienen, bei Benutzung der $\mathbf{V_L}$ - Maschine als linksseitiger "Anschlag" zu gelten.

Es mag ein wenig pingelig erscheinen, wenn man sich mit solchen Kleinigkeiten herumplagt. Sollten sie aber versuchen, die Maschinen durch ein geeignetes Programm zu simulieren, dann kommt man an solchen Problemem nicht vorbei. Am Schluss sollte dann das Band wie nebenstehend aussehen.



Rekursiv war an diesem Beispiel eigentlich noch nichts. Das wird jetzt anders !

Nach der Definition von Seite 25 müssen wir zwei Maschinen bauen, die den beiden Rekursionsvorschriften entsprechen (hier für zwei Argumente): $f(0,x) = g(x)$ und $f(y,x) = h(f(y-1,x),x)$

Offensichtlich muss man **y** als Zähler auf dem Turingband benutzen, der angibt, wie oft eine Turingmaschine für die Funktion **h** arbeiten soll. Die zu **h** gehörende Bandbeschriftung ergibt sich aus f und x. Wenn dann der Zähler y bis auf Null abgearbeitet ist (was einer verbleibenden "1" auf dem Band entspricht), dann arbeitet eine weitere Turingmaschine für die Funktion g.

© Hans-Georg Beckmann 2005

Das wird nun schrittweise an einem einfachen Beispiel entwickelt.

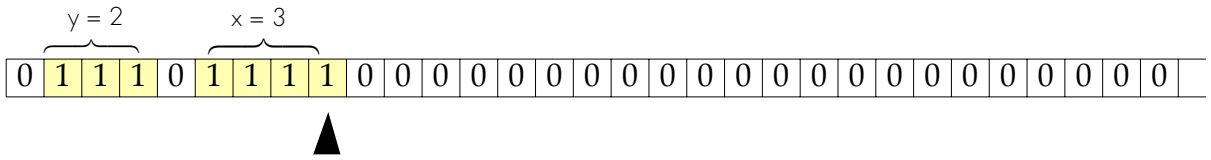
Es soll eine Turingmaschine gebaut werden, die sum(2,3) rekursiv abarbeiten kann. Dabei gilt:

$$\text{sum}(\, y, x\,) = N\,(\,\text{sum}\,(\,y-1,x)) \;\text{ und }\; \text{sum}\,(\,0,x\,) = P_2\,(\,0,x\,) = x$$

In der oben gewählten Darstellung entspräche damit die Projektionsfunktion P_2 (0 , x) der Funktion f(0,x) also g(x). Die Nachfolgefunktion, die zur Berechnung von f(y , x) benutzt wird , entspricht damit der Funktion h.

Die zu h gehörende Maschine wird dann so oft arbeiten, wie es y angibt.

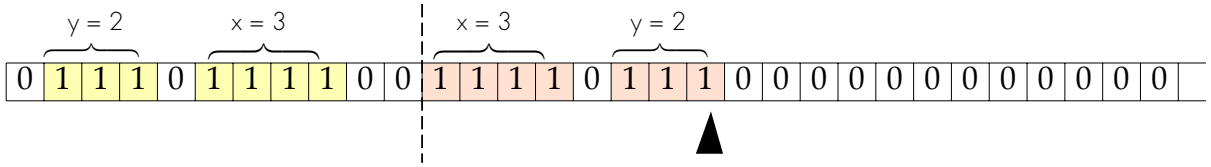
Startbeschriftung aus f(y , x) = sum (2 , 3)



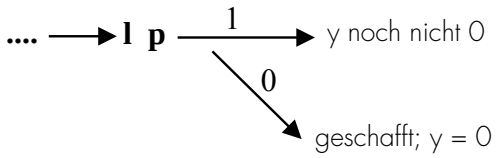
Zuerst stehen y und x auf dem Band. Eine Maschine, die K_{20} heißen soll, hat die Aufgabe die beiden Zahlen nach rechts hin auf das Band zu kopieren und dabei gleich die Reihenfolge der Argumente zu vertauschen. Der Zwischenraum, zwischen den beiden Zahlen links und den Kopien soll duch eine Doppelnull markiert sein. **K_{20} .....**

Wenn K_{20} (deren Funktion hier noch nicht genau angegeben werden muss) gearbeitet hat, sieht das Band wie folgt aus:

K_{20} hat gearbeitet.



Nun wird geprüft, ob y = 0 ist. Wäre das der Fall, dann hätte man schon genügend Kopien von x auf dem Band. Wenn nicht, muss weiter kopiert werden. Man muss zu dieser Prüfung nur ein Feld nach links gehen. Ist man dann auf der trennenden Null rechts vom Argument x, dann hatte man für y offensichtlich nur noch eine 1 auf dem Band, was bedeutet, dass y = 0 gelten würde. Die Maschine setzt sich also fort:

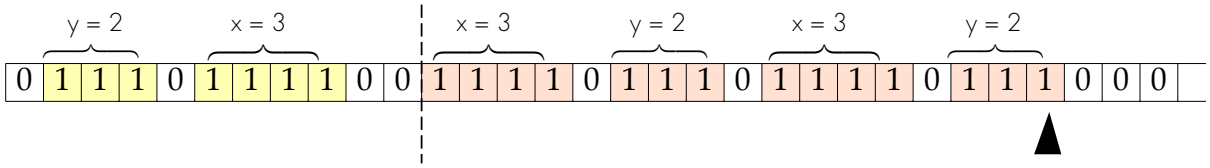


Ist y noch nicht Null, dann wieder einen Schritt nach rechts und x und y mit einer Kopiermaschine **K_2** rechts auf das Band kopieren:

© Hans-Georg Beckmann 2005

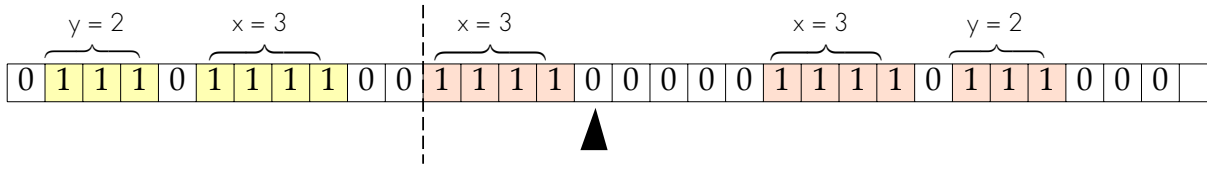
..... r K_2 ....

K_2 hat gearbeitet.



Nun stört das y zwischen den beiden x. Also soll es dort gelöscht werden. Das kann man u.a. mit zwei großen Linksmaschinen und einer Löschmaschine Z erreichen:

y herausgelöscht

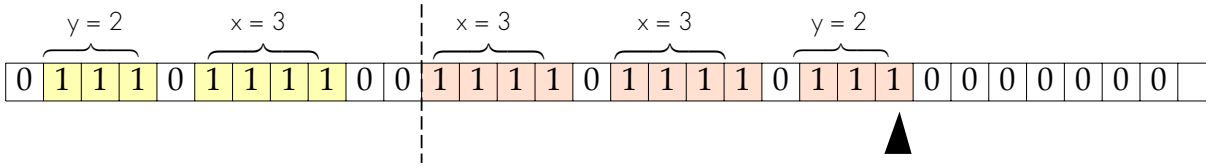


... L L Z ....

Nun werden die beiden Einsergruppen, die rechts stehen, gesucht und an die erste Kopie von x nach links herankopiert. Dazu brauchen wir eine große Rechtsmaschine (speziell) , die eine Anzahl von Nullen übergeht und dann auf der ersten Eins der Einsergruppe stehen bleibt. Sie soll $\mathbf{R_0}$ heißen. Eine weitere (normale) große Rechtsmaschine **R** führt auf die rechte Eins in der Einsergruppe. Die Verschiebemaschine $\mathbf{V_L}$ schiebt dann diese Einsergruppe nach links. Zuerst wird die Gruppe für x=3 verschoben, danach die Gruppe für y=2.

... $R_0RV_LR_0RV_L$ ....

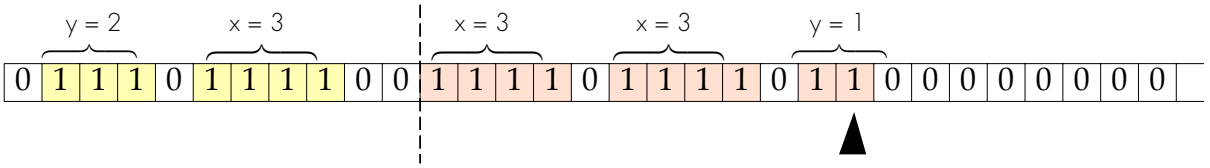
geschoben



Nun kann der Zähler y um 1 vermindert werden:

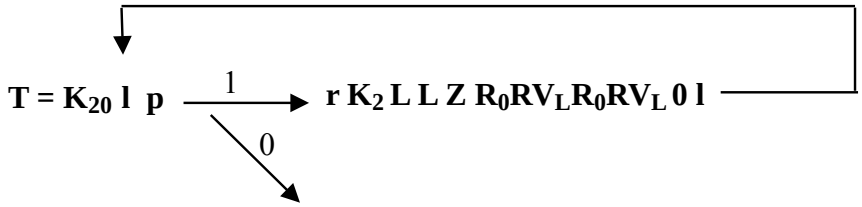
.... 0 l ....

y um 1 vermindert

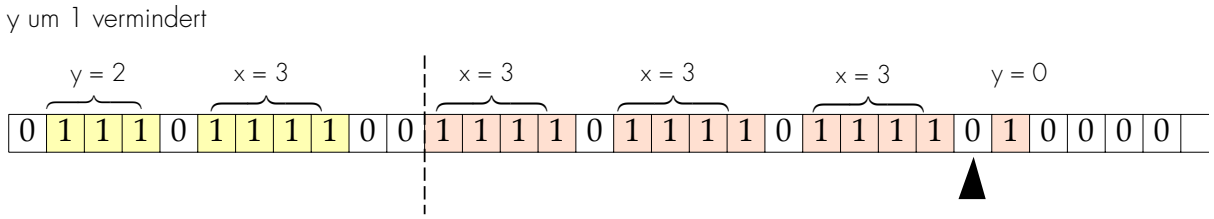


© Hans-Georg Beckmann 2005

Dieser Teil der Turingmaschine muss nun solange wiederholt werden, bis y = 0 ist:



Wenn y = 0 erreicht ist, sieht das Band so aus:



Die kleine Linksmaschine vor der Prüfmaschine führt nun auf die trennende Null und damit wird der rekursive Teil der Turingmaschine verlassen.

Zuerst soll nun y = 0 vom Band gelöscht werden:

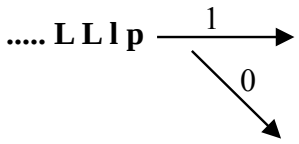
... r 0 l l .....

Das war nicht kompliziert.

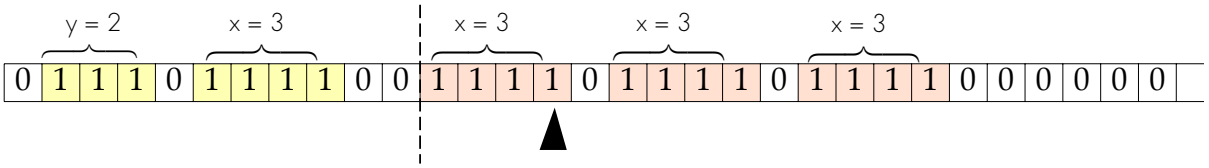
Nun kann mit den Argumenten x, die auf dem Band stehen gearbeitet werden, da ja y = 0 ist. Das tut die noch näher zu bestimmende Maschine $\mathbf{T_g}$, die für die Funktion g steht.

... $\mathbf{T_g}$ ....

Bevor die Maschine weiter arbeitet, muss erst einmal nachgeprüft werden, ob schon alle Kopien von x auf dem Band abgearbeitet sind. Dazu wird geprüft, ob links schon die Doppelnull neben der aktuellen Einsergruppe steht. Wäre das der Fall, dann wären wir fertig !



Der Kopf der Maschine steht jetzt :



Noch keine Doppelnull- also arbeiten und daher zurück nach rechts:

... r R R l ....

© Hans-Georg Beckmann 2005

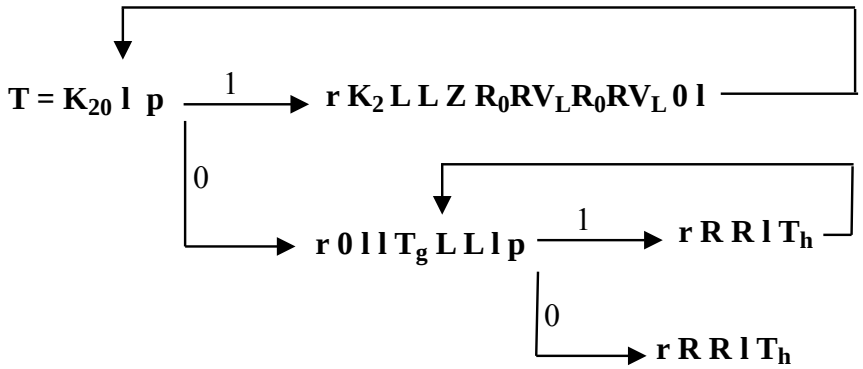
Jetzt soll die Maschine $\mathbf{T_h}$ arbeiten, die für die Funktion **h** steht, die in unserem Fall die Nachfolgefunktion ist.

Diese Maschine $\mathbf{T_h}$ wird die aktuelle Einsergruppe rechts um eine "1" erweitern und dann die Einsergruppe links neben der verlängerten Gruppe löschen. Damit ist dann eine Kopie des Arguments x abgearbeitet. Nun muss die Gruppe (x +1) nur noch nach links verschoben werden , was eine Maschine $\mathbf{V_1}$ erledigt, die eine Einsergruppe verschiebt.

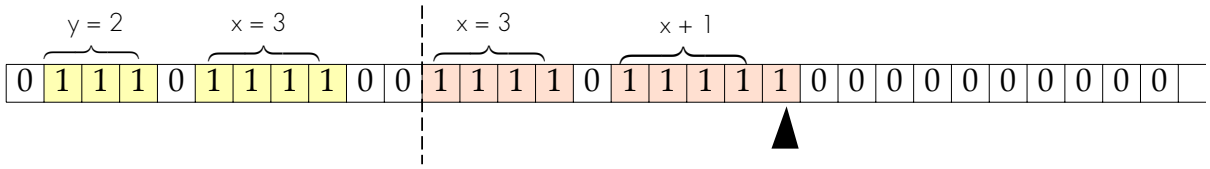
$$\mathbf{T_h : r\ 1\ L\ l\ Z\ R\ R\ l\ V_1}$$

Die schon erwähnte Maschine **Tg** ist die Projektionsfunktion, die in unserem Fall nicht weiter tut, als das Argument x auf dem Band stehen zu lassen.

Das Gesamtbild der Maschine T für die rekursive Summenbildung:

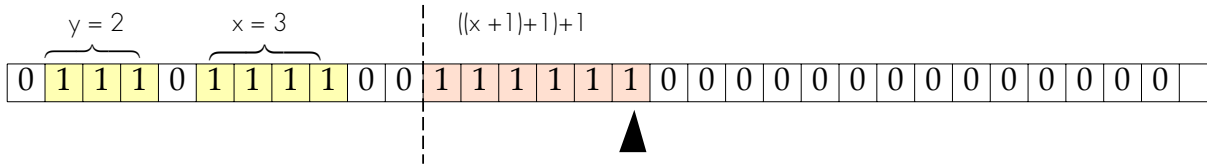


T_h hat zum ersten Mal gearbeitet:



Am Ende stehen die beiden Anfangsargumente und das Ergebnis auf dem Turingband.

T_h hat zum zweiten Mal gearbeitet und ist fertig



Auch wenn nun noch einige Teilmaschinen (V_1 oder R_0) zu erarbeiten wären, ist doch die Maschine insgesamt betriebsbereit. Wir können annehmen, dass Turingmaschinen in der Lage sind, rekursive Funktionen zu berechnen.

© Hans-Georg Beckmann 2005

Aufgabe:

Entwickeln sie die fehlenden Teilmaschinen für das vorangegangene Beispiel.

Wenn wir nach diesem hinreichend komplizierten Beispiel davon überzeugt sind, dass Turingmaschinen rekursive Funktionen berechnen können, dann sind damit diese Maschinen leistungsfähiger als alle anderen bisher behandelten endlichen Automaten. Im Prinzip können auch moderne Computer nicht mehr leisten als eine Turingmaschine.
"Außerdem kann man mit Turingmaschinen die Grenzen der algorithmischen Lösungsmethode erforschen. Da jedem Algorithmus eine Funktion zugeordnet werden kann, die sich durch eine Turingmaschine berechnen lässt, kann umgekehrt auch die Existenz von nicht lösbaren Problemen gezeigt werden, wenn eine entsprechende Turingmaschine nicht existiert." [4]

Die universelle Turingmaschine

Man kann Turingmaschinen entwickeln, die andere Turingmaschinen simulieren. Das sind dann universelle Turingmaschinen, bei denen man sich einige Gedanken über die Beschriftung des Bandes machen muss. Wenn **T** die Maschine ist, die simuliert werden soll, dann lässt sich **T** in Form einer Automatentafel darstellen. Die Tafel ist aber nichts anderes als eine Wertetabelle für die Überführungsfunktion u. Diese Wertetabelle kann codiert und als eine Folge von Zahlen dargestellt werden. Damit ist sie auf einerm Turingband darstellbar. Das Band der Maschine **T** kann direkt auf das Band der "großen" Simulationsmaschine geschrieben werden. Auf dem Band ist dann auch noch genug Platz für die Abarbeitung der Automatentafel und die Werte und Zwischenergebnisse, die auftreten werden.

Das Halteproblem[5]

Allen Computerbenutzerinnen und Computerbenutzern ist sicher die Situation bekannt, in der man vor dem Gerät sitzt und sich fragt: *"Arbeitet er noch, oder hängt er ?"* Glücklicherweise wird man meistens nach einiger Wartezeit aus der Ungewissheit erlöst, wenn es heißt " Ein schwerer Systemfehler ist aufgetreten..... wenden sie sich an der Hersteller."

[4] E.Modrow, Automaten, Schaltwerke, Sprachen Seite 196, Dümmler 1987
[5] Der Abschnitt basiert auf einem Script, das man unter http://www8.informatik.uni-erlangen.de/IMMD8/Lectures/ALGO-I/Turing findet

© Hans-Georg Beckmann 2005

Genau das tun wir natürlich sofort.

Es kann aber leicht passieren, dass nichts passiert, weil ein Prozess nicht zum Ende kommt. Bei selbst geschriebenen Programmen ist das dann oft eine unendliche Schleife, weil eine Bedingung zur Beendigung der Schleife nicht erfüllt werden kann:

```
a = 2;
while ( a != 1)   { b = 2 * a +1; }
```

Um solche "Kleinigkeiten " geht es aber nicht.

Das Problem kann allgemeiner gefasst werden: Hat man einen Algorithmus gestartet, dann könnte man eine Turingmaschine entwickeln oder angeben, die der Funktion entspricht , die zum Algorithmus gehört. Nur wenn diese Turingmaschine auch anhält, kann man sagen, dass ein Ergebnis berechnet wurde. Hält eine Turingmaschine nicht an, dann war das Verfahren, das man entwickelt hat, kein Algorithmus. (Ein Algorithmus zeichnet sich ja dadurch aus, dass Ergebnisse erzeugt werden).

Das Problem ist also:

Für eine beliebige Turingmaschine mit eine Bandbeschriftung aus Eingabedaten soll entschieden werden, ob diese Maschine mit diesen Daten nach endlicher Zeit stoppt oder nicht.

Nun könnte man eine Firma gründen, 20 Programmierer einstellen um ein Programm schreiben zu lassen, das dieses Problem löst. Man müsste jedoch feststellen, dass die Firmengründung ein Fehler war, weil sich diese Frage prinzipiell nicht beantworten lässt.

Man kann sogar beweisen, dass es kein Programm gibt, welches obige Frage beantwortet.

Eine ähnlich grundlegende Frage wurde von dem Göttinger Mathematiker David Hilbert im Jahre 1900 auf einem Kongress in Paris gestellt.

Gibt es ein Verfahren, das für jede ausreichend formalisierte Aussage der Mathematik entscheidet, ob diese wahr oder falsch ist?

Dieses sogenannte **Hilbertsche Entscheidungsproblem** blieb zuerst ungelöst.

Eine positive Lösung des Problems hätte aus heutiger Sicht zur Folge gehabt, dass Computer mathematische Aussagen erzeugen und auch gleich noch deren Wahrheit oder Falschheit beweisen könnten.

Erst im Jahr 1936 konnten sowohl A. Church als auch A. Turing beweisen, dass das oben genannte Entscheidungsproblem nicht lösbar ist.

© Hans-Georg Beckmann 2005

Warum es so lange gedauert hat, bis dieses Problem gelöst wurde ? Das liegt daran, dass es sehr schwierig ist, zu beweisen, dass etwas nicht existiert.

Wenn man beweisen will, dass ein Verfahren zur Lösung eines Problems *existiert*, dann muss man das Verfahren nur angeben.

Wenn man aber beweisen will, dass ein Verfahren zur Lösung eines Problems *nicht existiert*, dann muss man zuerst festlegen, was ein Verfahren überhaupt ist !

Genau diese Überlegungen haben Alan Turing 1936 zu der Idee der Turingmaschine geführt. Bis heute ist kein berechenbares Problem bekannt, das nicht bereits mit der Turingmaschine lösbar wäre. Daher gilt die Churchsche These mittlerweile als akzeptiert:

Alles was überhaupt (intuitiv) berechenbar ist, ist schon mit der Turingmaschine berechenbar.

(Auf Seite 25 hatten wir die These schon einmal in einer anderen Formulierung)

Diese These ist aber nicht bewiesen !

Man kann sie auch noch einmal anders formulieren :

Alles was mit der Turingmaschine nicht berechenbar ist, lässt sich überhaupt nicht berechnen.

Die zweite Formulierung wird verwendet, wenn bewiesen werden soll, dass etwas nicht berechenbar ist. Es reicht dann nämlich, zu zeigen, dass es mit der Turingmaschine nicht zu berechnen ist.

Dass nicht jede Funktion berechenbar ist, weiß man schon länger. Es gibt nämlich in der Mathematik mehr Funktionen als Berechnungsverfahren. Es war allerdings lange Zeit keine konkrete Funktion bekannt, die nicht berechenbar ist.

Im Jahre 1962 konnte eine nicht berechenbare Funktion angegeben werden und auch dabei spielte die Turingmaschine wieder eine entscheidende Rolle.

Zuerst die Kurzfassung des Problems:

Die Fleißige-Biber-Funktion ordnet einer Anzahl von Zuständen die maximale Anzahl von Einsen zu, die eine Turingmaschine mit dieser Anzahl von Zuständen auf das leere Band schreiben kann.

Beachte: Die Turingmaschine muss irgendwann stoppen!

Diese Funktion / Problemstellung stammt vom ungarischen Mathematiker Tibor Rado.

© Hans-Georg Beckmann 2005

Fleißige Biber

Auf eher humorvolle Weise näherte sich A.K. Dewdney den kleinen Tieren.[6]

Von Bienen einmal abgesehen, wird man gerne glauben , dass Biber die fleißigsten Tiere sind. In andauernden Geschäftigkeit verbringen sie ihre Zeit mit dem Bau von Dämmen und dem Sammeln der dazu notwendigen Materialien.

Der Fleiß dieser Tiere mag Tibor Rado bei der Namensgebung für sein Problem inspiriert haben.

Die Frage, die er sich stellte (weil Mathematiker ja sonst auch nichts besseres zu tun haben) lautet:

" Wieviele Einsen kann eine Turingmaschine mit n Zuständen maximal auf ein leeres (also nur mit Nullen gefülltes) Band schreiben, bevor die anhält ?

Für n= 1, n= 2 , n= 3 und n=4 ist die Frage beantwortet. Jedoch nicht für n= 5 oder höhere Werte von n.

Das kann man auf den ersten Blick nicht glauben.

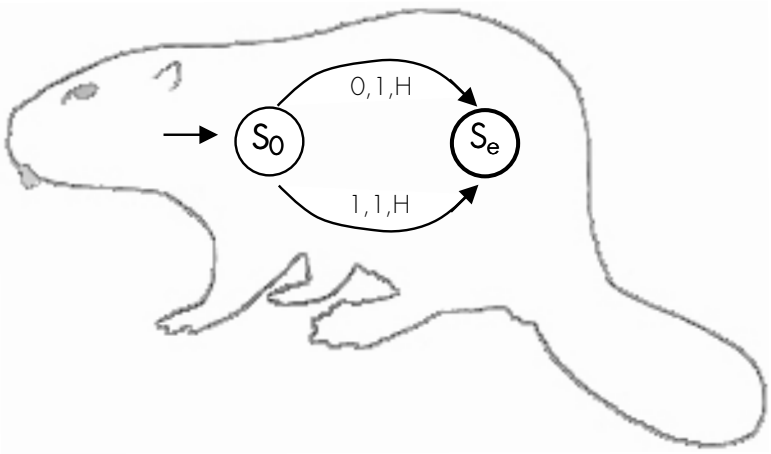
Schauen wir uns einige Biber mal näher an.

Wie gewohnt sollen die Turingmaschinen in einem Zustand S_0 starten. Die zulässigen Eingabezeichen vom Band sind "0" und "1". Die Maschine hält an, wenn sie in einem Endzustand S_e ankommt, den wir nicht als Zustand mitzählen werden.

Auch werden wir in S_e keine Eingabezeichen mehr verarbeiten. So können wir die Biber mit dem weiter oben angegebenen Simulationsprogramm testen, da sich die Automatentafeln leicht angeben lassen.

Der kleinste Biber **"Castor minimalis"**

Er hat nur einen Zustand und sein Transitionsgraph sieht recht einfach aus.



Man sieht schnell, dass dieser Biber genau eine "1" auf das Band schreibt und dann anhält.

Der Biber mit 2 Zuständen **"Castor dualis"**

Da wird es schon ein wenig schwieriger.
Haben sie schon Mut, es selbst zu probieren ? Soviel sei verraten, er schafft vier Einsen und hält dann an.
Wenn sie es schaffen, ihn zu konstruieren, dann sollte er wie auf der nächsten Seite aussehen.

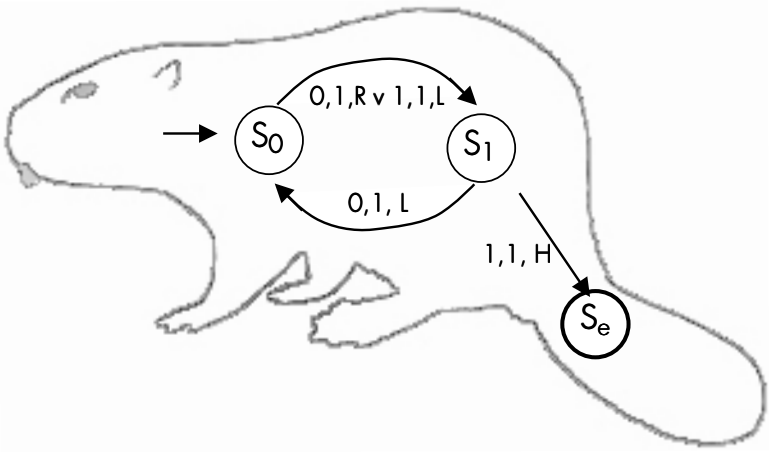
[6] A.K. Dewdney "Fleißige Biber " in Spektrum der Wissenschaften, Sonderheft Computer Kurzweil II, 1988

© Hans-Georg Beckmann 2005

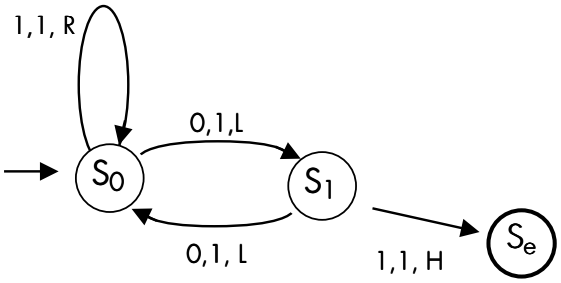
Beim Versuch diesen Biber zu entwicklen, werden sie wahrscheinlich schon auf Maschinen stoßen, die nicht anhalten.

Weil das dann keine Biber ist, muss das Bild des Transitionsgraphen ein wenig anders aussehen.

Testen sie diese Maschine !



Man kann bei einer vorgegebenen Anzahl von Zuständen die möglichen Turingmaschinen systematisch durchprobieren. Man beginnt in einem Zustand S_0. Von dem müssen zwei Kanten des Transitionsgraphen ausgehen. Die Zielzustände S_i und S_j können unterschiedlich sein , es kann aber auch einer davon wieder S_0 sein. Nun können an allen Kanten die Eingabezeichen 0 oder 1 stehen, und es können Kopfbewegungen L und R auftreten, da die Kopfbewegung H für die Kante reserviert ist, die zum Endzustand S_e führt. Fleißige Menschen haben ein sehr frustrierendes Ergebnis gefunden. Bereits für n=5 Zustände gibt es 63403380965376 mögliche Turingmaschinen, die man alle testen müsste, um herauszufinden , ob es fleißige Biber sind, ob es vielleicht der fleißigste Biber ist oder ob die Maschine leider nicht anhält.

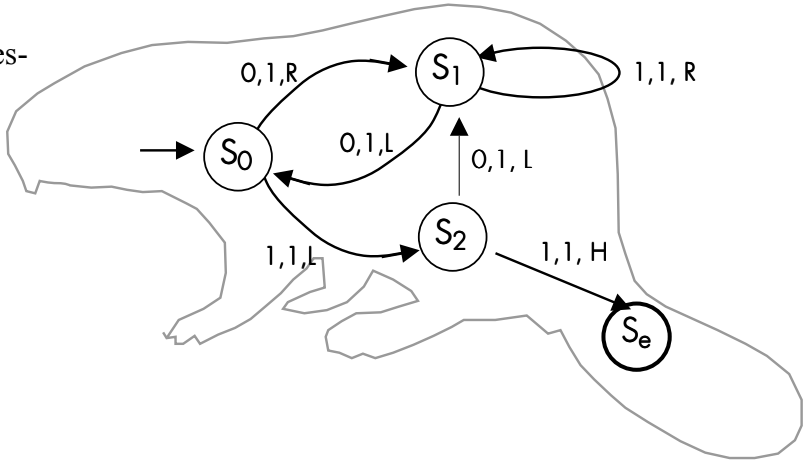


Ein Biber, der keiner ist, weil er nicht stoppt.

Doch schauen wir noch einen weiteren interessante Kandidaten an, den Biber mit drei Zuständen **"Castor trio"**

Seine Automatentafel ist schnell aufgestellt.

Probieren sie aus, wieviele Einsen er schafft.



Zustand	Eingabe-zeichen	Ausgabe-zeichen	Folge-zustand	Kopf-bewegung
S0	0	1	S1	R
S0	1	1	S2	L
S1	0	1	S0	L
S1	1	1	S1	R
S2	0	1	S1	L
S2	1	1	Se	H

Hinweis für die Benutzung des Simulationsprogramms: Wenn sie das Programm so benutzen, wie ab Seite 14 beschreiben, dann findet es hier keine Startposition, da ja keine "1" auf dem Band steht. Schreiben sie irgendwo eine "1" auf das Band. Klicken sie "Startposition".Der S/L-Kopf geht nun an die richtige Stelle. Jetzt können sie die 1 wieder vom Band löschen und dennoch arbeitet die Maschine danach korrekt, da die Startposition unverändert bleibt. Sie können aber das Programm auch der Bibererforschung nach Belieben anpassen.

© Hans-Georg Beckmann 2005

Wie fängt man einen Biber ?

Im Jahr 1983 wurde ein Wettbewerb organisiert, um den besten Biber mit 5 Zuständen zu finden.

Er fand in Dortmund statt und aus 133 Turingmaschinen, die an den Start gingen, ergab sich die Maschine von Uwe Schult als Sieger. Dieser Biber schaffte 501 Einsen bevor er tatsächlich anhielt. Ist damit aber der fleißigste Biber in die Falle gegangen ? Die meisten Informatiker meinten - ja !

Kann man beweisen, dass es keinen besseren geben kann ?

Wenn man die maximale Anzahl von Einsen in Abhängigkeit von n als $\Sigma(n)$ bezeichnet, dann sind die Werte $\Sigma(1)$ bis $\Sigma(4)$ bekannt, nicht aber $\Sigma(5)$. Die Funktion $\Sigma(n)$ hat ein sehr unangenehme Eigenschaft, sie ist nicht berechenbar. Das hatte Tibor Rado selbst schon 1962 beweisen.

Sie wächst außerdem schneller als eine Exponentialfunktion. Der oben erwähnte Uwe Schult hat eine Maschine gefunden, die mit 6 Zuständen 2075 Einsen erzeugt und eine Maschine mit 12 Zuständen die es auf eine schier undarstellbare Zahl bringt, die Schult in nebenstehender Form angibt.

Es ist 6 * 4096 hoch 4096 hoch 4096 .....hoch 4096 hoch 4.

In der Mitte, die hier nur durch Punkte angegeben ist, taucht die Zahl 4096 insgesamt 166 mal im Exponenten auf, was hier durch Punkte angegeben ist.

$$6 * 4096^{4096^{4096^{4096^{\cdot^{\cdot^{\cdot^{4096^{4}}}}}}}}$$

Auch eine Funktion s(n), die die Anzahl der Arbeitschritte angibt, die eine Bibermaschine bis zum Anhalten braucht , hat sich als nicht berechenbar herausgestellt.

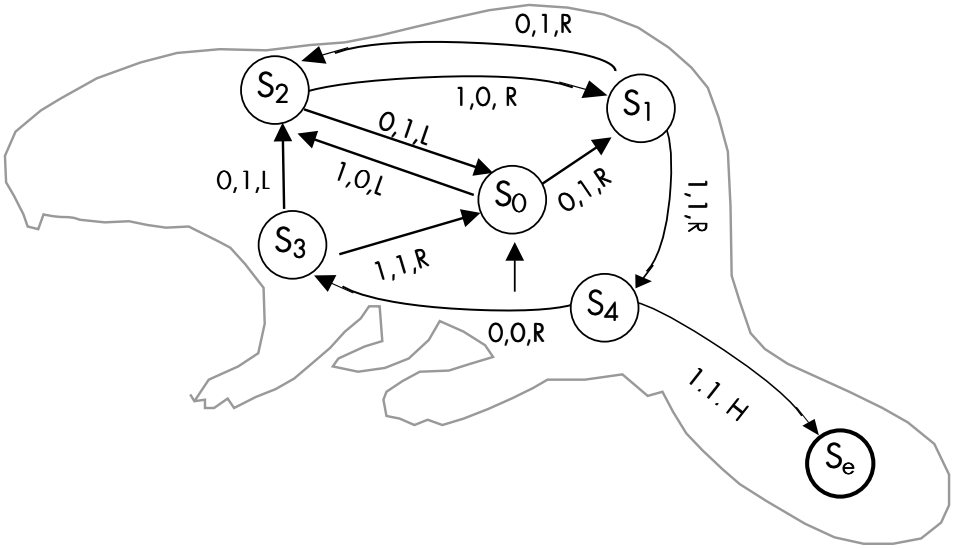
Nach Uwe Schult fand man einen Biber mit 5 Zuständen und 1915 Einsen und im Jahr 1989 wurde in Deutschland einer mit 4098 Einsen gesichtet. Dieses bisher letzte Exemplar wurde nach einer 3-tägigen Jagd mit Hilfe eines Hochgeschwindigkeitscomputers gefunden.

Unterdessen ist die Suche nach fleißigen Bibern nicht zu Ende und der jetzige Stand der Dinge lässt sich als Tabelle zusammenfassen:

Anzahl Zustände	Max. Anzahl Einsen auf dem Band
1	1
2	4
3	6
4	13
5	mindestens 4098
6	mindestens 95.524.079

© Hans-Georg Beckmann 2005

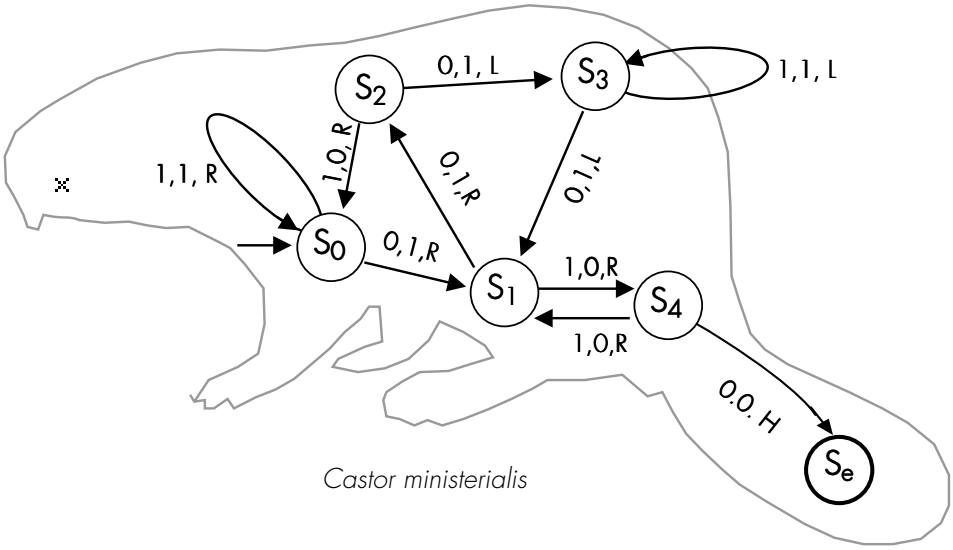
Der Biber mit 5 Zustanden, der 501 Nullen schafft.



Zum Schluss noch einige Biberexemplare, die bei der Suche nach dem fleißigsten Tierchen nebenbei gefangen wurden und die man näher betrachten muss, um seine Freude an ihnen zu haben.

Castor ministerialis

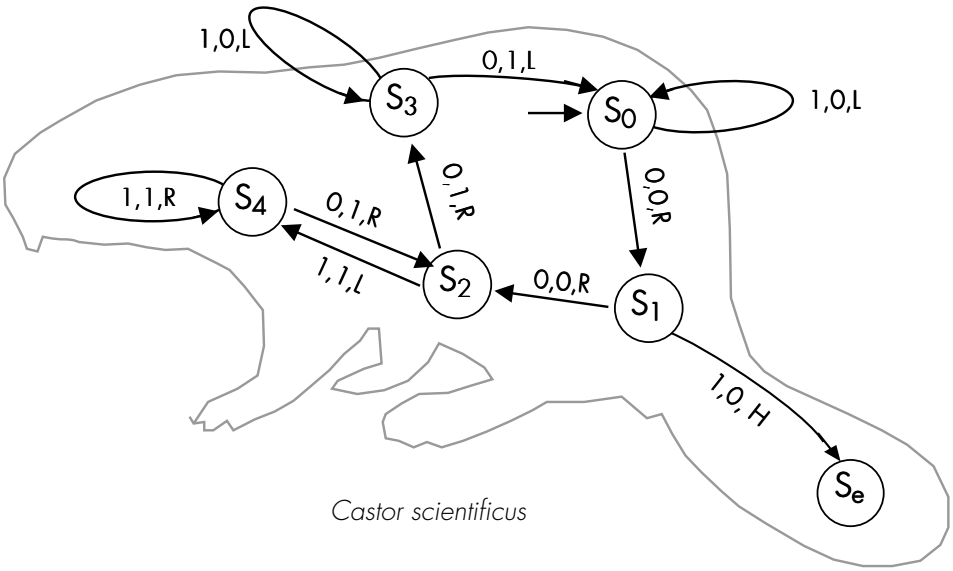
(Beamtenbiber) Diese besondere Art versucht möglichst weit voran zu kommen , ohne dabei etwas zu leisten. Er erzeugt nur eine einzige 1, kommt aber immerhin 11 Stellen vorwärts (also von A5 auf A16).



Castor ministerialis

Castor scientificus

(Wissenschaftsbiber) Diese Sorte arbeitet sich ganz furchtbar ab , ohne etwas zu produzieren. Dieses Exemplar macht 187 Schritte, kommt aber nur 8 Felder voran und erzeugt nicht eine einzige 1 auf dem Band.



Castor scientificus