Faltunkskodierung und katastrophale Eigenschaften

Dirk Schulze und Stefan Mätzler

16. März 2009

Inhaltsverzeichnis

1 Die Anwendung

1.1 Systemvoraussetzungen

Die Anwendung ist System- und Plattformunabhängig und lässt sich auf jedem beliebigen Gerät ausführen. Grundvoraussetzung ist ein JavaScript fähiger Browser mit eingeschaltetem JavaScript. Für den vollen Funktionsumfang (Ergebnis mit grafischer Auswertung), empfiehlt es sich, die jeweils aktuelle Version Ihres Browsers zu nutzen.

Um eine zügige Verarbeitung zu gewährleisten, werden ausdrücklich folgende Browser empfohlen: Safari ab Version 3.0, Firefox ab Version 3.0, Opera ab Version 9 sowie Google Chrome.

1.2 Starten der Anwendung

Starten Sie zunächst Ihren Browser. Geben Sie folgende Adresse (URL) in die Adresszeile Ihres Browsers ein und bestätigen Sie anschließend mit Enter:

`http://www1.inf.tu-dresden.de/~s7468461/kanalkodierung.html`

Nach dem Laden der Anwendung, sehen Sie folgenden Fensterinhalt:

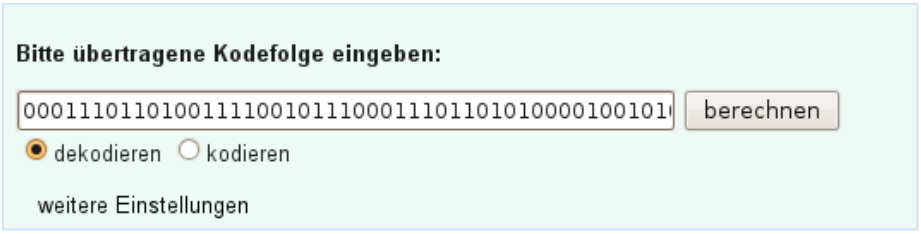

Abbildung 1: Gestartete Anwendung

1.3 Anwendungsaufbau

In der Mitte des Fensterinhaltes sehen Sie einen blauen Rahmen mit einem Eingabefeld für das Kodewort, einen Bestätigungsbutton `berechnen`, sowie zwei Auswahlmöglichkeiten `dekodieren` und `kodieren` zur Dekodierung oder Kodierung.
Mit dem Aufruf `weitere Einstellungen` können Sie weitere Einstellungen in der Anwendung vornehmen (siehe Weitere Einstellungen, Kapitel 3).

Es ist ein Faltungskode mit der Generatormatrix $G = (13_8, 15_8, 17_8)$ voreingestellt.

2 Kodieren und Dekodieren

2.1 Starten der Berechnungen

Zum Kodieren oder Dekodieren eines beliebigen Kodewortes, fügen Sie zunächst das in binärer Form vorliegende Kodewort in das dafür vorgesehene Eingabefeld ein. Dazu markieren Sie das Eingabefeld und löschen die vormals eingebene Kodefolge. Geben Sie das Kodewort manuell, oder per rechten Mausklick und dem Menüpunkt `Einfügen` ein. Wählen Sie `kodieren` oder `dekodieren` mit einem Klick auf Ihre linke Maustaste aus. Zum Starten der Berechnungen klicken Sie auf `berechnen`. Nach Vollendung der Berechnung erscheint ein zweiter, rötlicher Rahmen direkt unter dem Blauen. Dieser Rahmen beinhaltet sämtliche Auswertungen der Berechnung, wenn nicht schon vorher ein Fehler während der Berechnung aufgetreten ist (siehe Warnungen und Fehlermeldungen, Kapitel 4).

Beispiel 1:

Wir wollen folgendes binäres Kodewort mit dem unter Anwendungsaufbau beschriebenen Faltungskode kodieren:

```
010110000101110000010110001011011110010001001001011001010101011100
111000
```

Wir fügen das Kodewort in das Eingabefeld ein und drücken anschließend den Button `berechnen`. Wir erhalten folgenden Fensterinhalt:

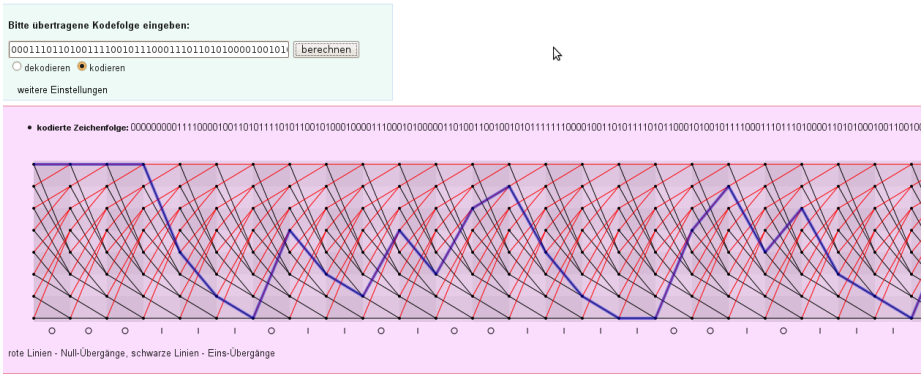

Abbildung 2: Grafische Ausgabe von Beispiel 1

Beispiel 2:

Wir wollen folgendes binäres Kodewort mit dem unter Anwendungsaufbau beschriebenen Faltungskode dekodieren:

```
000111011010011110010111000111011010100001001010111000000111110010
011110010111111011110011110101011001110001010000011101111111011010000
001110100001101001111001000010001010010000001010001001100100101011
111100001001010111
```

Dazu fügen wir das Kodewort in das Eingabefeld ein, drücken anschließend den Button `berechnen` und erhalten nun folgenden Fensterinhalt:

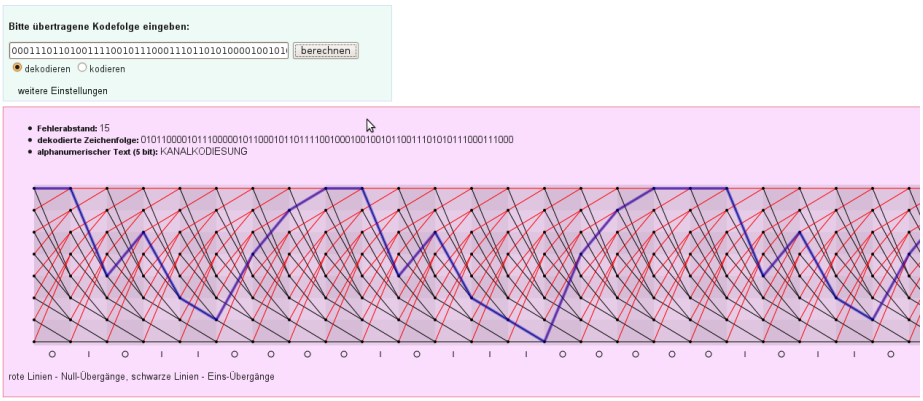

Abbildung 3: Grafische Ausgabe von Beispiel 2

2.2 Auswertung der Berechnungen

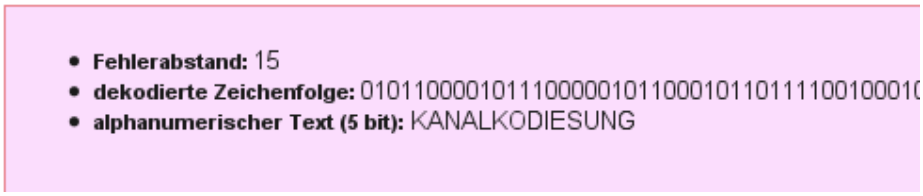

Abbildung 4: Ergebnis der Berechnung

Sämtliche Ergebnisse der Berechnungen werden in einem rötlichen Rahmen unterhalb des blauen, für die Eingabe zuständigen, Rahmen angezeigt. Ist eine grafische Auswertung gewünscht (Standardeinstellung), so wird zusätzlich ein Trellis-Diagramm angezeigt (siehe Grafische Auswertung, Kapitel 3.5). Die Auswertung umfasst folgende Eigenschaften:

- **Fehlerabstand** - Fehlerabstand zu einer Fehlerfreien Übertragung
- **dekodierte Zeichenfolge** - binäre Ergebnis nach der Dekodierung
- **kodierte Zeichenfolge** - binäre Ergebnis nach der Kodierung
- **alphanumerischer Text (5 bit)** - 5 Zeichen des Kodewortes werden zu einem alphanumerischen Zeichen `A-Z` gewandelt.

3 Weitere Einstellungen

3.1 Aufbau

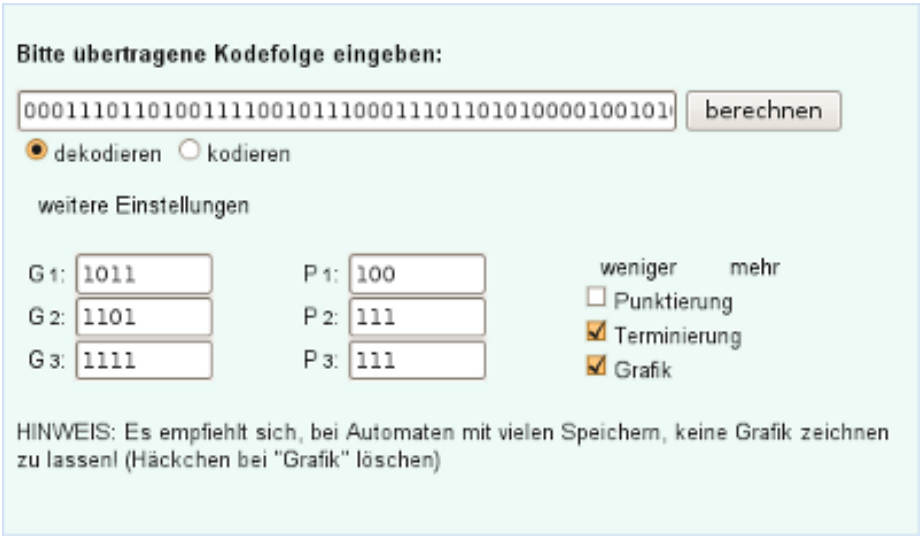

Abbildung 5: Weitere Einstellungen

3.2 Generatormatrix

3.2.1 Allgemeines zur Generatormatrix

Ein Faltungskodierer ist ein LTI-System (linear time-invariant system) und beruht auf den Grundlagen der mathematischen Faltung, daher auch Faltungskode. Damit lässt sich jeder Ausgang mit einer Übertragungsfunktion beschreiben. Diese Anwendung unterstützt keine Rekursionen. Dies entspricht einem Automaten ohne Rückkopplung. Eine Impulsantwort wird mittels einer Z-Transformation berechnet. So haben die Übertragungsfuntkionen die Form:

$$G_k(z) = a_{k0} + a_{k1}z^{-1} + \ldots + a_{kn}z^{-n}$$

$a_{kn} = 0$ wenn Speicher keinen Einfluss auf das Ergebnis des Ausgangs k hat, 1 sonst. k ist der jeweilige Ausgang, n der jeweilige Speicher.

Die Kombination aller Übertragungsfunktionen lassen sich als Matrix darstellen, indem die Übertragungsfunktionen untereinander gesetzt werden (jeder Koeffizient entsprechend zu der Stelle seines Speichers).

3.2.2 Anpassen der Generatormatrix

Um die Matrixhöhe zu vergrößern oder zu verkleinern (Zeilen hinzufügen, löschen), drücken Sie `mehr` zum Vergrößern oder `weniger` zum Verkleinern. Die Eingabe erfolgt in das jeweilige, dafür vorgesehene Feld und muss binär sein.

Dabei steht das erste Feld für die Übertragungsfunktion des ersten Ausgangs, das n-te Feld für die Übertragungsfunktion des n-ten Ausgangs. Die Breite der Generatormatrix lässt sich durch die Eingabe entsprechend vieler Nullen und Einsen erweitern bzw. vermindern. Die Länge jeder Übertragungsfunktion muss gleich und nicht leer sein.

3.3 Punktierung

3.3.1 Allgemeines zur Punktierung

Mit Punktierung ist das auslöschen von Informationsstellen an vordefinierten Stellen gemeint. Diese Stellen werden durch eine Matrix, der Punktierungsmatrix, angegeben. Bei einer bestimmten Güte der Übertragung, sowie einer leistungsfähigen Generatormatrix, können die gesendeten Informationen dennoch rekonstruiert werden.

3.3.2 Nutzung der Punktierung

Um Punktierung nutzen zu können, setzen Sie das Häkchen bei `Punktierung` mit einem linken Mausklick und zum deaktivieren, das Häkchen wieder löschen.

3.3.3 Punktierungsmatrix

Die Punktierungsmatrix hat die selbe Höhe wie die Generatormatrix. Jede Zeile muss eine binäre Zeichenfolge, gleicher Länge enthalten, andernfalls wird ein Fehler ausgegeben (siehe Warnungen und Fehlermeldungen, Kapitel 4). Um die Matrixhöhe zu vergrößern oder zu verkleinern (Zeilen hinzufügen, löschen), drücken Sie `mehr` zum Vergrößern oder `weniger` zum Verkleinern.

3.4 Terminierung

Sind an einer empfangenen Kodefolge keine Terminierungsbits angehängt, besteht die Möglichkeit die geforderte Terminierung abzuschalten, so dass die Speicherglieder nicht am Ende der Dekodierung mit Nullen gefüllt werden. Dazu löschen Sie bitte das Häkchen bei `Terminierung`. Das gilt auch, wenn Sie eine Zeichenfolge ohne Terminierung kodierende wollen.

3.5 Grafische Auswertung

Die grafische Auswertung erfolgt mittels eines Trellis-Diagramms.

Sollten Sie komplexe Faltungskodes mit vielen Speichergliedern berechnen wollen empfiehlt es sich die Grafikausgabe abzuschalten, da diese enorm viel Rechenzeit in Anspruch nimmt. Löschen Sie dazu das Häkchen bei `Grafik`.

4 Warnungen und Fehlermeldungen

4.1 Warnungen

- **Die Generatormatrix hat katastrophale Eigenschaften verursacht durch einen gemeinsamen Teiler $\neq 1$ der Generatorpolynome!**
  Die Generatormatrix hat katastrophale Eigenschaften, da alle Generatorpolynome einen gemeinsamen Teiler $\neq 1$ besitzen. Das Kodewort lässt sich womöglich nicht mehr rekonstruieren. Wählen Sie eine andere Generatromatrix (siehe Katastrophale Eigenschaften von Generatormatrizen, Kapitel 5.2).

- **Die Generatormatrix hat katastrophale Eigenschaften verursacht durch einen Zyklus im Zustandsgraphen mit dem Gewicht Null!**
  Die Generatormatrix hat katastrophale Eigenschaften. Es gibt einen Zyklus über einen oder mehrere Knoten im Zustandsgraphen mit dem Gewicht Null. Das Kodewort lässt sich womöglich nicht mehr rekonstruieren. Wählen Sie eine andere Generatromatrix (siehe Katastrophale Eigenschaften von Generatormatrizen, Kapitel 5.2).

- **Diese Punktierungsmatrix verursacht katastrophale Eigenschaften!**
  Die von Ihnen angegebene Punktierungsmatrix verursacht katastrophale Eigenschaften. Das Kodewort lässt sich womöglich nicht mehr rekonstruieren. Wählen Sie eine andere Punktierungsmatrix (siehe Katastrophale Eigenschaften von Generatormatrizen, Kapitel 5.2).

4.2 Fehlermeldungen

- **Zeichenfolge nicht dekodierbar: Anzahl Zeichen inkorrekt!**
  Die von Ihnen angegebene Zeichenfolge ist inkompatibel mit der gegenwertigen Generatormatrix. Die Zeichenkettenlänge muss ein Vielfaches der Anzahl der Zeilen der Generatormatrix sein (auch Anzahl an Ausgängen eines Automaten). Bitte überprüfen Sie Ihre Eingabe.

  Beispiel: $G = (6_8, 1_8), b = 1$ Die Zeichenlänge beträgt 1, die Generatormatrix hat aber 2 Zeilen (2 Ausgänge). Da die Zeichenlänge kein Vielfaches der Anzahl der Zeilen der Generatormatrix ist, ist diese Folge nicht dekodierbar. Dies gilt ausdrücklich *nicht* für Kodierung.

- **Wählen Sie einen andere Punktierungsmatrix P! Weiter ohne Punktierung.**
  Die von Ihnen angegebene Punktierungsmatrix ist inkonsistent. Bitte überprüfen Sie die Eingabe Ihrer Punktierungsmatrix. Die Länge einer jeden Matrixzeile muss gleich und nicht leer sein. Es wird ohne Punktierung fortgefahren.

- **Bitte nur Binärfolgen nutzen!**
  Einer Ihrer Angaben ist keine Binärfolge. Jede eingegebene Zeichenkette darf nur aus `0` oder `1` bestehen und keine Buchstaben, Leerzeichen oder Sonderzeichen beinhalten. Die Überprüfung erfolgt mittels eines zeichenweisen Abgleichs auf Null oder Eins. Wird in den Eingaben ein anderes Zeichen entdeckt, wird die weitere Berechnung abgebrochen.

- **Wählen Sie einen anderen Faltungskodierer G!**
  Die von Ihnen angegebene Generatormatrix ist inkonsistent. Bitte überprüfen Sie die Eingabe Ihrer Generatormatrix. Die Länge einer jeden Matrixzeile muss gleich und nicht leer sein.

5 Funktionsweise und Ergebnisse

5.1 Allgemeine Funktionsweise der Anwendung

1. **Laden der Eingaben**
   Zunächst werden die Eingaben des Nutzers, wie kodierte/zu kodierende Zeichenfolge, Punktierungsmatrix, Generatormatrix, Terminierung und Punktierung, eingelesen.

2. **Verifizierung der Generatormatrix**
   Nach dem Einlesen, wird nun die Generatormatrix verifiziert. Dabei wird geprüft, ob jede Zustandsgleichung eine gleich lange, nicht leere Binärfolge ist und ob die Zeichenfolge stimmig zur Generatormatrix ist. Bei Bedarf wird der weitere Vorgang mit einer Fehlermeldung abgebrochen (siehe Fehlermeldungen, Kapitel 4.2)

3. **Erstellen des Trellis-Diagramms**
   Aus den Daten der Generatormatrix wird das Trellis-Diagramm erstellt. Dies wird für die Kodierung und Dekodierung als auch bei der grafischen Darstellung verwendet. Mit Hilfe des Trellis-Diagramms lassen sich auch katastrophale Eigenschaften des Faltungskodierers feststellen.

4. **Kodierung / Dekodierung**
   Abhängig von der Wahl des Nutzers wird nun entweder der Kodierungs- oder Dekodierungsprozess gestartet. Bei aktivierter Punktierung (siehe Punktierung, Kapitel 3.3) wird zunächst die eingegebene Zeichenfolge punktiert und anschließend erneut auf katastrophale Eigenschaften geprüft. Es folgt das berechnen des Resultats nach dem Prinzip von Viterbi.

5. **Anzeigen der Resultate**
   Der letzte Schritt ist die Aufbereitung und Anzeige der Resultate. Abhängig davon, ob `Grafik` aktiviert ist (siehe Grafische Auswertung, Kapitel 3.5) und zusätzlich der Browser HTML 5 Canvas oder VML (MS Internet Explorer) unterstützt, erfolgt die grafische Auswertung. Die grafische Auswertung ist ein Trellis-Diagramm, berechnet anhand der eingegebenen Generatormatrix und Punktierung (sofern aktiviert), in das der Pfad des resultierenden Kodes eingezeichnet wird.

5.2 Katastrophale Eigenschaften von Generatormatrizen

Ein Faltungskodierer mit katastrophalen Eigenschaften kann bei einer endlichen Kodefolge theoretisch unendlich viele Dekodierfehler erzeugen.
Katastrophale Eigenschaften von Generatormatrizen erkennt man an:

1. alle Generatorpolynome haben einen gemeinsamen Teiler $\neq 1$ und $\neq x^i$ mit $i > 0$

2. es gibt eine Schleife im Zustandsgraphen mit dem Gewicht 0 außer dem Nullzustand

3. alle Addierer haben eine gerade Anzahl von Verbindungen

4. Test auf Katastrophalität von Massey-Sain

Diese Anwendung nutzt die beiden ersten Prüfalgorithmen.

5.2.1 Gemeinsamer Teiler $\neq 1$ der Generatorpolynome

Besitzen alle Generatorpolynome ein gemeinsames Teilerpolynom $p(x) \neq 1$, so hat die Generatormatrix katastrophale Eigenschaften.

$G(x) = (p(x)\tilde{g}_1(x), \ldots, p(x)\tilde{g}_n(x))$ mit gemeinsamen Teiler $p(x) \neq 1$

Eine Ausnahme sind Polynome $p(x)$ mit Potenzen von x, x^k. Diese entsprechen einer Verzögerung der Ausgabe um k. Da sie keine weiteren Auswirkungen auf die Kodefolge haben, sind diese unnötig aber verursachen selbst keine katastrophalen Eigenschaften.

Systematische Faltungskodierer haben mindestens ein Generatorpolynom $g_k = 1$. Aus diesem Grund haben systematische Faltungskodes nie katastrophale Eigenschaften, da das größte gemeinsame Teilerpolynom $p(x)$ immer 1 ist.

5.2.2 Schleife mit dem Gewicht Null

Hat der Zustandsgraph einer Generatormatrix Schleifen (auch Zyklen genannt), mit einem Gewicht von Null, so besitzt diese Generatormatrix katastrophale Eigenschaften. Dabei lässt man die Schleife, mit dem Gewicht Null, im Nullzustand außer acht.
Solche Schleifen lassen sich leicht daran erkennen, dass ein Pfad von einem Zustand zu sich selbst existiert, der nur Null als Ausgabe hat.

5.2.3 Beispiel für einen katastrophalen Faltungskodierer

Ein Beispiel für einen Faltungskodierer mit katastrophalen Eigenschaften ist:

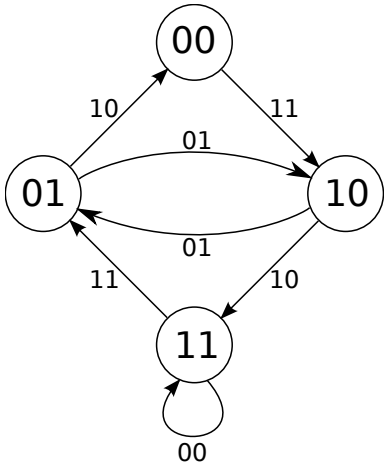

Abbildung 6: katastrophaler Faltungskodierer

$$G = (5_8, 3_8)$$

Finden der katastrophalen Eigenschaften über:

1. **Gemeinsames Teilerpolynom der Generatorpolynome**
   Die Generatorpolynome sind:

$$\begin{aligned} g_1 &= 1 + x^2 \\ g_2 &= x + x^2 \end{aligned}$$

   Diese beiden Polynome lassen sich in die folgenden irreduziblen Polynome zerlegen:

$$\begin{aligned} g_1 &= (1 + x)(1 - x) \\ g_2 &= (1 + x)x \end{aligned}$$

   Wie man nun unschwer erkennen kann, ist das größte gemeinsame Teilerpolynom $p(x) : (1 + x)$, mit $p(x) \neq 1$ und $p(x)$ keine Potenz von x.
   $\Rightarrow$ Die Generatormatrix besitzt katastrophale Eigenschaften.

2. **Schleife mit dem Gewicht Null**
   Wie man in der Abbildung 6 leicht erkennen kann, besitzt der Zustand 11 einen Pfad zu sich selbst mit dem Gewicht Null.
   $\Rightarrow$ Die Generatormatrix besitzt katastrophale Eigenschaften.

5.3 Katastrophale Eigenschaften durch Punktierung

5.3.1 Erkennungsalgorithmus für katastrophale Punktierungesmatrizen

Punktierungen können durch das Auslöschen von Informationsstellen katastrophale Eigenschaften verursachen, ohne das die Generatormatrix selbst katastrophale Eigenschaften besitzt.

Der in dieser Anwendung verwendete Algorithmus sucht nach solchen geschlossenen Schleifen im Zustandsgraphen. Dazu werden zunächst alle Knoten markiert. Anschließend werden alle abgehenden Pfade mit einer Punktierungsmatrix überlagert.

Nun werden von jedem markierten Knoten ausgehende Pfade, welche ergänzt wurden und deren Gewichte Null sind, gespeichert und deren letzten Knoten markiert. Dies geschieht so lange bis es keine markierten Knoten mehr gibt (Terminierung des Algorithmus mit nicht katastrophal) oder das Ende der Punktierungsmatrix erreicht ist (mit Ausnahme der Überprüfung der Generatormatrix) und ein Zyklus in einem gespeicherten Pfad gefunden wurde. Bei dieser Betrachtung wird der Pfad vom Nullzustand zu sich selber mit dem Gewicht Null vernachlässigt.

5.3.2 Beispiel für eine katastrophale Punktierungsmatrix

Ein Beispiel: $G = (5_8, 7_8)$ mit Punktierungsmatrix: $P = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$

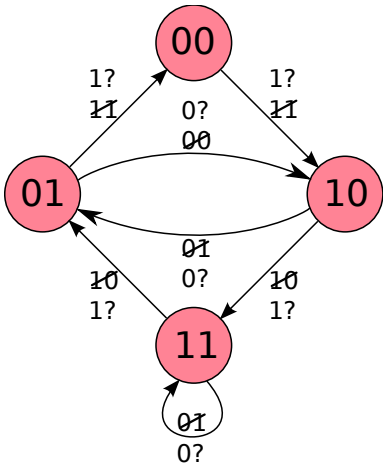

Abbildung 7: Schritt 1

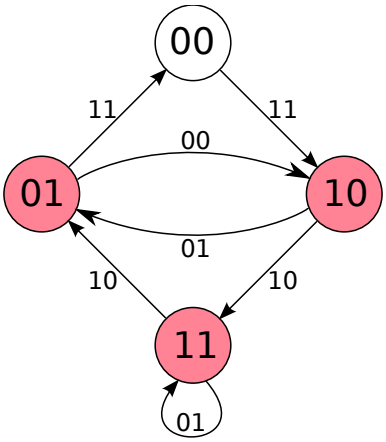

Abbildung 8: Schritt 2

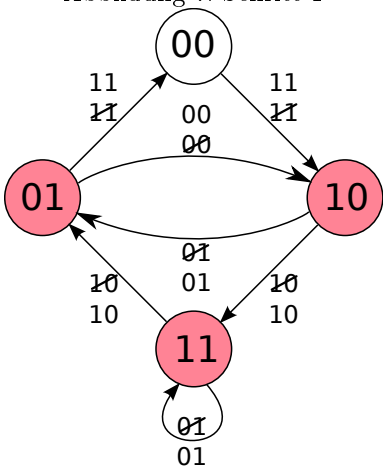

Abbildung 9: Schritt 3

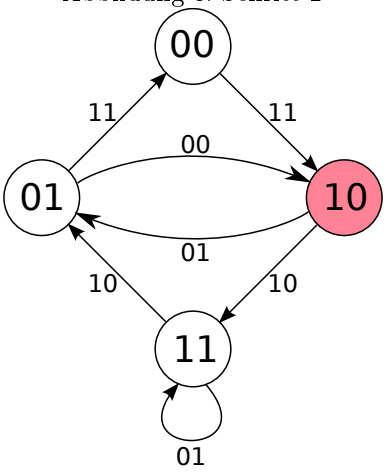

Abbildung 10: Schritt 4

Die Generatormatrix besitzt keine katastrophalen Eigenschaften.

Schritt 1:
Alle Knoten werden markiert und alle abgehenden Pfade mit der ersten Spalte der Punktierungsmatrix $(1\,0)^T$ überlagert. (Abbildung 7)

Schritt 2:
Es werden alle Knoten markiert, die über einen punktierten Weg mit dem Gewicht Null zu erreichen sind. Zusätzlich werden diese Wege gespeichert. (Abbildung 8)

Wege:

- Zustand 10 → Zustand 01
- Zustand 01 → Zustand 10
- Zustand 11 → Zustand 11

Schritt 3:
Da das Ende der Punktierungsmatrix noch nicht erreicht ist, kann der Zyklus (Zustand 11 → Zustand 11) noch nicht berücksichtigt werden.
Es werden wieder die abgehenden Pfade der markierten Knoten mit der aktuellen Spalte der Punktierungsmatrix $(1\,1)^T$ überlagert. (Abbildung 9)

Schritt 4:
Das Ende der Punktierungsmatrix ist erreicht und es wurde ein Zyklus mit dem Gewicht Null gefunden. (Abbildung 10)

Der Weg lautet: Zustand 10 → Zustand 01 → Zustand 10.

5.4 Ergebnisse der Aufgabe

Zu Aufgabe (E) lautet der dekodierte Text: KANALKODIESUNG und zu Aufgabe (H): FALTUNGSSUMME.

6 Quellenverzeichnis

6.1 Literaturverzeichnis

- Bernd Friedrichs, Peter Herbig, *Kanalcodierung: Grundlagen und Anwendungen in modernen Kommunikationssystemen*, Springer Verlag, 1996
- Herbert Klimant, Rudi Potraschke, Dagmar Schönfeld, *Informations- und Kodierungstheorie*, 4. Auflage, Teubner Verlag
- Martin Röder, *Effiziente Viterbi Decodierung und Anwendung auf die Bildübertragung in gestörten Kanälen*, Leipzig, 2001
- Schwarz, *Entwicklung geeigneter Verfahren zur sicheren und zuverlässigen Bestimmung zunächst unbekannter Übertragungsverfahren und Kodierung*, Dresden, 2007
- Dr. Ing. Volker Kühn, Vorlesungsskript Kanalcodierung I, Bremen
- Martin Bossert, *Kanalcodierung: Mit 36 Tabellen und 211 Beispielen*, Teubner Verlag, 1998
- Lehrmaterial der Lehrveranstaltung Kanalkodierung
- Wikipedia EN: Convolutional code

7 Quelltext

Listing 1: convolutionalCode.js

```
var canvas, ctx;
// this is the main-function
function calculateCode (b)
{
    if (!isValid(b))
        return;

    outputBox();
    // create the transfer function first.
    g = transferFunction();
    if (g == undefined) {
        addInformation("Fehler", "Waehlen Sie einen anderen Faltungskodierer H!");
        return;
    }

    var numOutputs = g.length;
    var numStates = Math.pow(2,g[0].length-1);
    // number of chars to code the states
    var stateCoding = Math.ceil(Math.log(numStates)/Math.log(2));

    // create Trellis diagram (state table)
    trellis = calculateTrellis(g, numStates, stateCoding);

    // catastrophic properties
    testVector = new Array();
    for (i=0;i<g.length;i++)
        testVector.push('1');
    if (!checkDivisor(g))
        addInformation("Warnung", "Die Generatormatrix hat katastrophale
                                   Eigenschaften verursacht durch einen gemeinsamen
                                   Teiler != 1 der Generatorpolynome!");

    if (!catastrophicProperties(trellis,testVector))
        addInformation("Warnung", "Die Generatormatrix hat katastrophale
                                   Eigenschaften verursacht durch einen Zyklus im
                                   Zustandsgrafen mit dem Gewicht Null!");

    // start decoding/encoding process
    result = new Array();
    if (document.formular.coding[1].checked)
        result = encodingCode(b, trellis, g);
    else
        result = decodingCode(b, trellis, g);

    if (result == undefined)
        return;

    // draw the trellis diagram with the result
    if (document.formProperties.graphics.checked)
        drawGraphics (result, trellis, stateCoding);

    if (document.formular.coding[0].checked)
        addInformation("alphanumerischer Text (5 bit)", binToText(result.join('')));
}

function encodingCode (b, trellis, g)
{
    var currentState = 0;
    result = new Array();
    erg = new Array();
    for (i=0;i<b.length;i++) {
        result.push(parseInt(b[i]));
        erg.push(trellis[currentState][0+parseInt(b[i])]);
        currentState = trellis[currentState][2+parseInt(b[i])];
    }
    var bCoded = erg.join('');
    if (document.formProperties.puncturing.checked)
        bCoded = puncturingCode(bCoded, 'encoding');
    addInformation("kodierte Zeichenfolge", bCoded);
```

```
70      return result;
71  }
72
73  function decodingCode (b, trellis, g)
74  {
75      var numOutputs = g.length;
76      var numStates = Math.pow(2,g[0].length-1);
77      var stateCoding = Math.ceil(Math.log(numStates)/Math.log(2));
78
79      if (document.formProperties.puncturing.checked)
80          b = puncturingCode(b, 'decoding');
81      if (b.length%numOutputs != 0) {
82          addInformation("Fehler", 'Zeichenfolge nicht dekodierbar: Anzahl Zeichen inkorrekt!');
83          return;
84      }
85
86      var codeLength = b.length/numOutputs;
87
88      // create arrays for each path (max. numStates*2)
89      paths = new Array(numStates);
90      for (m = 0; m < numStates; m++) {
91          paths[m] = new Array(codeLength);
92          // new state, result, length
93          for (n = 0; n < codeLength; n++)
94              paths[m][n] =  new Array('0',0,0);
95      }
96
97      // calculate the paths.
98      for(i = 0; i < codeLength; i++) {
99          // calculate every transition to every state and
100         // store it in a temporary array (char by char)
101         var rowHeight = Math.pow(2,i);
102         tempRow = new Array();
103         var maxRow = Math.min(numStates, rowHeight);
104         for(j = 0; j < maxRow; j++) {
105             var lastState = 0;
106             var distance = 0;
107             if (i>0) {
108                 var posY = j;
109                 if (rowHeight < numStates*2)
110                     posY = (numStates/rowHeight) * j;
111                 lastState = paths[posY][i-1][0];
112                 distance = paths[posY][i-1][2];
113             }
114             tempRow.push(new Array(trellis[lastState][2], 0,
115                                    binaryComparator(trellis[lastState][0],
116                                    b.substr(i*numOutputs, numOutputs)) + distance, j));
117             if (document.formProperties.termination.checked) {
118                 if (!(((i+g[0].length-1)*numOutputs)>=b.length))
119                     tempRow.push(new Array(trellis[lastState][3], 1,
120                                    binaryComparator(trellis[lastState][1],
121                                    b.substr(i*numOutputs, numOutputs)) + distance, j));
122             } else
123                 tempRow.push(new Array(trellis[lastState][3], 1,
124                                    binaryComparator(trellis[lastState][1],
125                                    b.substr(i*numOutputs, numOutputs)) + distance, j));
126         }
127         // reduce transitions according to VITERBI. That means
128         // reduce temporary array if possible and save the transitions
129         // to the right path.
130         if (tempRow.length <= numStates) {
131             // building of Trellis not yet released
132             var repeat = numStates/tempRow.length;
133             var dist = 0;
134             for(j = 0; j < tempRow.length; j++) {
135                 for(l = 0; l < repeat; l++) {
136                     paths[l+dist][i][0] = tempRow[j][0];
137                     paths[l+dist][i][1] = tempRow[j][1];
138                     paths[l+dist][i][2] = tempRow[j][2];
139                 }
140                 dist += repeat;
141             }
142         } else {
143             tempPaths = new Array();
```

```
144             for(j = 0; j < numStates; j++) {
145                 tempPath = new Array();
146                 tempState = new Array();
147                 for(l = 0; l < tempRow.length; l++) {
148                     if (tempRow[l][0] == j)
149                         tempState.push(tempRow[l]);
150                 }
151                 // only 2 paths go to one state, choose the shortest of both.
152                 // If the two paths have the same length, choose the earlier one.
153                 // FIXME: add termination
154                 if (tempState[0][2] >= tempState[1][2])
155                     tempState.shift();
156                 var numPath = tempState[0][3];
157                 tempPath = copy2DArray(paths[numPath]);
158                 tempPath[i][0] = tempState[0][0];
159                 tempPath[i][1] = tempState[0][1];
160                 tempPath[i][2] = tempState[0][2];
161                 tempPaths.push(tempPath);
162             }
163             paths = tempPaths;
164         }
165     }
166
167     // choose shortest Path
168     var shortest;
169     var pathLength = 1000;
170     for(i = 0; i < numStates; i++) {
171         if (paths[i][codeLength-1][2] < pathLength) {
172             shortest = i;
173             pathLength = paths[i][codeLength-1][2];
174         }
175     }
176     result = new Array();
177     for(i = 0; i < codeLength; i++) {
178         result.push(paths[shortest][i][1]);
179     }
180
181     addInformation("Fehlerabstand", paths[shortest][codeLength-1][2]);
182     addInformation("dekodierte Zeichenfolge", result.join(''));
183     return result;
184 }
185
186 function drawGraphics (paths, trellis, stateCoding) {
187     var canvas = document.createElement('canvas');
188     var output = document.getElementById("output");
189     output.appendChild(canvas);
190
191     if (/MSIE/.test(navigator.userAgent) && !window.opera) {
192         if ((!canvas.getContext) && (typeof G_vmlCanvasManager != "undefined"))
193             canvas = G_vmlCanvasManager.initElement(canvas);
194     } else {
195         if (!canvas.getContext) {
196             addInformation("Hinweis", "Fuer Opera, Safari, Chrome oder Firefox (-basierende)
197                                 Browser wird an dieser Stelle eine Grafik angezeigt!");
198             return;
199         }
200     }
201
202     var statesWidth = 50;
203     var statesHeight = 30;
204     canvas.width = Math.max(450,paths.length*statesWidth+45);
205     canvas.height = trellis.length*statesHeight+25;
206     output.style.width = canvas.width+'px';
207     ctx = canvas.getContext("2d");
208
209     ctx.save();
210     ctx.translate(35, 30);
211     //ctx.scale(450/(result.length*50+25),450/(result.length*50+40));
212
213     // draw complete diagram
214     for (i=0;i<paths.length;i++) {
215         ctx.fillStyle = "rgba("+100*(i%2)+","+100*(i%2)+","+100*(i%2)+",0.1)";
216         ctx.fillRect(i*statesWidth,-5,statesWidth,statesHeight*(trellis.length-1)+10);
217         for (j=0;j<trellis.length;j++) {
```

```
218             ctx.fillStyle = "rgba("+150*(j%2)+","+150*(j%2)+","+150*(j%2)+",0.05)";
219             if (j<trellis.length-1)
220                 ctx.fillRect(i*statesWidth,j*statesHeight,statesWidth,statesHeight);
221             for (l=0;l<2;l++) {
222                 ctx.strokeStyle = "rgb("+255*(1-l)+",0,0)";
223                 ctx.beginPath();
224                 ctx.moveTo(i*statesWidth,j*statesHeight);
225                 ctx.lineTo(i*statesWidth+statesWidth,trellis[j][2+l]*statesHeight);
226                 ctx.stroke();
227             }
228         }
229     }
230     for (i=0;i<paths.length+1;i++) {
231         for (j=0;j<trellis.length;j++) {
232             ctx.fillStyle = "rgb(0,0,0)";
233             ctx.beginPath();
234             ctx.arc(i*statesWidth, j*statesHeight, 2, 0, 2*Math.PI, false);
235             ctx.closePath();
236             ctx.fill();
237         }
238     }
239
240     // draw path in to diagram
241     ctx.strokeStyle = "rgba(0,0,200,0.5)";
242     ctx.lineWidth = 4;
243     var currentState = 0;
244     for (i=0;i<paths.length;i++) {
245         ctx.beginPath();
246         ctx.moveTo(i*statesWidth,currentState*statesHeight);
247         ctx.lineTo(i*statesWidth+statesWidth,trellis[currentState][2+result[i]]*statesHeight);
248         currentState = trellis[currentState][2+result[i]];
249         ctx.stroke();
250     }
251     ctx.restore();
252
253     // draw text (onlx for Firefox >=3.1, Safari >= 4 beta)
254     if (ctx.fillText) {
255         ctx.save();
256         ctx.translate(0,33);
257         ctx.fillStyle = "black";
258         for (i=0;i<trellis.length;i++)
259             ctx.fillText(decToBin(i, stateCoding), 0, statesHeight*i);
260         ctx.restore();
261
262         ctx.save();
263         ctx.translate(60,trellis.length*statesHeight+15);
264         for (i=0;i<paths.length;i++)
265             ctx.fillText(paths[i], i*statesWidth, 0);
266         ctx.restore();
267     } else {
268         ctx.save();
269         ctx.translate(60,trellis.length*statesHeight+13);
270         for (i=0;i<paths.length;i++) {
271             if (parseInt(paths[i])) {
272                 ctx.beginPath();
273                 ctx.moveTo(i*statesWidth,0);
274                 ctx.lineTo(i*statesWidth,8);
275                 ctx.stroke();
276             } else {
277                 ctx.save();
278                 ctx.beginPath();
279                 // not supported by excanvas
280                 //ctx.translate(i*statesWidth*0.35,0);
281                 //ctx.scale(0.65,1);
282                 ctx.arc(i*statesWidth, 4, 4, 0, 2*Math.PI, false);
283                 ctx.stroke();
284                 ctx.restore();
285             }
286         }
287         ctx.restore();
288     }
289
290     var text = document.createElement('p');
291     var info = document.createTextNode('rote Linien - Null-Uebergaenge,
```

```
292                                         schwarze Linien - Eins-Uebergaenge');
293     text.appendChild(info);
294     output.appendChild(text);
295 }
296
297 /********************
298  * helper functions *
299  ********************/
300
301 function checkDivisor(g)
302 {  // false -> catastrophic
303     gCopy = new Array();
304     for (n=0;n<g.length;n++) {
305         gCopy.push("");
306         for (o=0;o<g[0].length;o++) {
307             gCopy[n] = g[n].substr(o,1) + gCopy[n];
308         }
309         // systematic Coders don't have catastrophic properties
310         if (binToDec(gCopy[n]) == 1)
311             return true;
312     }
313
314     var anzahl = Math.pow(2,gCopy[0].length);
315     for (o=2;o<=anzahl;o++) {
316         if ((o==2) || ((o%2)!=0)) {
317             var poly=decToBin(o);
318             // a polynomial x^n is just a delay
319             if (countOne(poly) == 1)
320                 continue;
321             // count generator polynomial with same divisor != 1
322             var countPoly = 0;
323             for (j=0;j<gCopy.length;j++) {
324                 if (polyDiv(gCopy[j],poly))
325                     countPoly++;
326             }
327             if (countPoly == gCopy.length)
328                 return false;
329         }
330     }
331     return true;
332 }
333
334 function polyDiv (p1, p2)
335 {  // true-> is divisor
336     p1 = decToBin(binToDec(p1));
337     while (p1!=0) {
338         if (p2.length>p1.length)
339             return false;
340         var second = p2;
341         while (p1.length!=second.length)
342              second=second+"0";
343         var zsp="";
344         for (i=0;i<second.length;i++)
345             zsp=zsp+xor(p1.substr(i,1), second.substr(i,1));
346         p1 = decToBin(binToDec(zsp));
347     }
348     return true;
349 }
350
351 function countOne (string)
352 {
353     var count = 0;
354     for (k=0;k<string.length;k++) {
355         if (string.substr(k,1) == 1)
356             count++;
357     }
358     return count;
359 }
360
361 function xor (a, b)
362 {
363     if (a != b)
364         return "1";
365     else
```

```
366         return "0";
367 }
368
369 function transferFunction ()
370 {
371     g = new Array();
372     var numElements = (document.getElementsByTagName("p")[3].childNodes.length+1)/6;
373     for (i=0;i<numElements;i++) {
374         var trans = document.getElementsByName("h")[i];
375         if (i==0) {
376             if (trans.value.length!=0)
377                 g.push(trans.value);
378             else
379                 return;
380         } else {
381             if (trans.value.length!=0 && trans.value.length==g[0].length)
382                 g.push(trans.value);
383             else
384                return;
385         }
386     }
387     return g;
388 }
389
390 function calculateTrellis (g, numStates, stateCoding)
391 {
392     var numOutputs = g.length;
393     trellis = new Array();
394     for (i=0;i<numStates;i++) {
395         line = new Array();
396         var oldState = decToBin(i, stateCoding);
397         // output for input 0, output for input 1, z(t+1) for 0, z(t+1) for 1
398         for (j=0;j<2;j++) {
399             // output
400             var output = '';
401             var string = String(j) + oldState;
402             for (l=0;l<numOutputs;l++) {
403                 var temp = 0;
404                 for (m=0;m<g[0].length;m++) {
405                     if (parseInt(g[l].substr(m,1)))
406                         temp = (parseInt(string.substr(m,1))+temp)%2;
407                 }
408                 output = output + String(temp);
409             }
410             line.push(output);
411         }
412         for (j=0;j<2;j++) {
413             // newState
414             var string = String(j) + oldState;
415             line.push(binToDec(string.substr(0,stateCoding)));
416         }
417         trellis.push(line);
418     }
419     return trellis;
420 }
421
422 function puncturingFunction ()
423 {
424     p = new Array();
425
426     var numElements = (document.getElementsByTagName("p")[3].childNodes.length+1)/6;
427     for (i=0;i<numElements;i++) {
428         var punc = document.getElementsByName("p")[i];
429         if (i==0) {
430             if (punc.value.length != 0)
431                 p.push(punc.value);
432             else
433                 return;
434         } else {
435             if (punc.value.length != 0 && punc.value.length == p[0].length)
436                 p.push(punc.value);
437             else
438                return;
439         }
```

```
440     }
441     return p;
442 }
443
444 function puncturingCode (b, coding)
445 {
446     puncturing = puncturingFunction();
447     if (puncturing == undefined) {
448         addInformation("Fehler", "Waehlen Sie einen andere Punktierungsmatrix P!
449                                   Weiter ohne Punktierung.");
450         return;
451     }
452
453     if (!catastrophicProperties(trellis,puncturing))
454         addInformation("Warnung", "Diese Punktierungsmatrix verursacht
455                                    katastrophale Eigenschaften!");
456
457     var column = 0;
458     var row = 0;
459     var new_b = '';
460     while (b.length > 0) {
461         var zsp = puncturing[row];
462         var compare = zsp.substr(column,1);
463         //unterscheidung zw. kodieren und dekodieren
464         if (coding=='decoding') {
465             if (compare == '0')
466                 new_b = new_b + 'x';
467             else {
468                 new_b = new_b + b[0];
469                 b = b.substr(1,b.length-1);
470             }
471         } else {
472             if (compare == '0')
473                 b = b.substr(1,b.length-1);
474             else {
475                 new_b = new_b + b[0];
476                 b = b.substr(1,b.length-1);
477             }
478         }
479         row++;
480         if (row%puncturing.length==0) {
481             row = 0;
482             column++;
483         }
484         if (column%puncturing[0].length==0)
485             column = 0;
486     }
487     return new_b;
488 }
489
490 //check for catastrophic properties of the transfer function
491 function catastrophicProperties (trellis, p) {
492     // step 1: initialisation
493     markedNodes = new Array();
494     for (i=0;i<trellis.length;i++) {
495         tmp = new Array();
496         tmp.push(i);
497         markedNodes.push(tmp);
498     }
499
500     // step 2: the loop of the algorithm
501     var k=0;
502     tempNodes = new Array();
503
504     while (markedNodes.length) {
505         path0 = markedNodes.pop();
506         path1 = copy1DArray(path0);
507         var lastState = path0[path0.length-1];
508         var output='';
509
510         if (lastState != 0) {
511             output = puncturingOutput(p,k,trellis[lastState][0]);
512             if (binToDec(output) == 0) {
513                 path0.push(trellis[lastState][2]);
```

```
514                     tempNodes.push(path0);
515                 }
516             }
517
518             output = puncturingOutput(p,k,trellis[lastState][1]);
519             if (binToDec(output) == 0) {
520                 path1.push(trellis[lastState][3]);
521                 tempNodes.push(path1);
522             }
523
524             if (markedNodes.length == 0) {
525                 k++;
526                 k = k%p[0].length;
527                 if (k == 0) {
528                     for (l=0;l<tempNodes.length;l++) {
529                         if (tempNodes[l][0] == tempNodes[l][tempNodes[l].length-1])
530                             return false;
531                     }
532                 }
533                 if (tempNodes.length != 0) {
534                     markedNodes = copy2DArray(tempNodes);
535                     tempNodes = new Array();
536                 }
537             }
538         }
539         return true; // no loop
540     }
541
542     function puncturingOutput ( p, column, code )
543     {
544         var zsp='';
545         for (k=0;k<p.length;k++) {
546             if (p[k].substr(column,1)=='1')
547                 zsp=zsp+code.substr(k,1);
548         }
549         return zsp;
550     }
551
552     function copy1DArray (oldArray) {
553         array = new Array();
554         width = oldArray.length;
555         for (o=0; o<width; o++)
556             array.push(oldArray[o]);
557         return array;
558     }
559
560     // don't give an empty array!
561     function copy2DArray (oldArray) {
562         array = new Array();
563         width = oldArray.length;
564         height = oldArray[0].length;
565         for (o=0; o<width; o++) {
566             array.push(new Array());
567             for (p=0; p<height; p++)
568                 array[o][p] = oldArray[o][p];
569         }
570         return array;
571     }
572
573     function binToDec (string) {
574         var dec = 0;
575         for (k=string.length; k>0; k--) {
576             dec = dec + Math.pow(2, (k-1))*string.substr(string.length-k,1);
577         }
578         return dec;
579     }
580
581     function decToBin (number, length) {
582         // http://kevin.vanzonneveld.net
583         // +   original by: Enrique Gonzalez
584         var string = parseInt(number).toString(2);
585         while(string.length < length) {
586             string = '0'+string;
587         }
```

```
588     return string;
589 }
590
591 function binToText (string) {
592     alphabet = new Array('','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O',
593                          'P','Q','R','S','T','U','V','W','X','Y','Z',' ','','','','','','');
594     var text = '';
595     while (string.length>0) {
596         text = text + alphabet[binToDec(string.substr(0,5))];
597         string = string.substr(5,string.length-5);
598     }
599     return text;
600 }
601
602 function isValid (string) {
603     for (i=0;i<string.length;i++) {
604         if (string.substr(i,1) != '0' && string.substr(i,1) != '1') {
605             addInformation("Fehler", "Bitte nur Binaerfolgen nutzen!");
606             return false;
607         }
608     }
609     return true;
610 }
611
612 // strings must have same length
613 function binaryComparator (string0, string1) {
614     var diff = 0;
615     if (string0.length != string1.length) {
616         return null;
617     }
618     for (k=0; k<string0.length; k++) {
619         if (string0.substr(k,1) != string1.substr(k,1))
620             diff++;
621     }
622     return diff;
623 }
624
625 function removeChilds (element) {
626     if (element.hasChildNodes()) {
627         while (element.childNodes.length>=1) {
628             element.removeChild(element.firstChild);
629         }
630     }
631 }
632
633 function divProperties ()
634 {
635     var div = document.getElementById("properties");
636     if (div.style.display == 'none')
637         div.style.display = 'inline';
638     else
639         div.style.display = 'none';
640 }
641
642 function removeRow ()
643 {
644     var elementH = document.getElementsByTagName("p")[2];
645     var elementP = document.getElementsByTagName("p")[3];
646     if (elementH.childNodes.length <= 13)
647         return;
648     for (i=0;i<6;i++) {
649         elementH.removeChild(elementH.lastChild);
650         elementP.removeChild(elementP.lastChild);
651     }
652 }
653
654 function addRow ()
655 {
656     //var pElements = document.getElementsByName("p");
657     var elementP = document.getElementsByName("p")[0].parentNode;
658     var elementH = document.getElementsByName("h")[0].parentNode;
659
660     // ie makes it difficult
661     var numElements = (elementH.childNodes.length+1)/6+1;
```

```
662
663     var brH = document.createElement("br");
664     var brP = document.createElement("br");
665     var spanH = document.createElement("span");
666     spanH.appendChild(document.createTextNode(' '+numElements));
667     var spanP = document.createElement("span");
668     spanP.appendChild(document.createTextNode(' '+numElements));
669
670     var inputH = document.createElement("input");
671     inputH.type = "text";
672     inputH.size = "7";
673     inputH.name = "h";
674     inputH.value = "";
675
676     var inputP = document.createElement("input");
677     inputP.type = "text";
678     inputP.size = "7";
679     inputP.name = "p";
680     inputP.value = "";
681
682     elementH.appendChild(document.createTextNode('\n'));
683     elementH.appendChild(brH);
684     elementH.appendChild(document.createTextNode('G'));
685     elementH.appendChild(spanH);
686     elementH.appendChild(document.createTextNode(': '));
687     elementH.appendChild(inputH);
688
689     elementP.appendChild(document.createTextNode('\n'));
690     elementP.appendChild(brP);
691     elementP.appendChild(document.createTextNode('P'));
692     elementP.appendChild(spanP);
693     elementP.appendChild(document.createTextNode(': '));
694     elementP.appendChild(inputP);
695 }
696
697 function addInformation (infoName, infoText)
698 {
699     var ul = document.getElementsByTagName("ul")[0];
700     var li = document.createElement('li');
701     var span = document.createElement('span');
702     span.style.fontWeight = "bold";
703     span.appendChild(document.createTextNode(infoName+': '));
704     li.appendChild(span);
705     li.appendChild(document.createTextNode(infoText));
706     ul.appendChild(li);
707 }
708
709 function outputBox()
710 {
711     var divOutput = document.getElementById("output");
712     divOutput.style.width = "388pt";
713     removeChilds(divOutput);
714
715     divOutput.style.display = "block";
716
717     var ulEle = document.createElement('ul');
718     divOutput.appendChild(ulEle);
719 }
```

Listing 2: HTML-Seite

```
720 <html><head>
721 <!-- quirks mode to support Canvas on IE 8 -->
722 <meta http-equiv="content-type" content="text/html; charset=UTF-8">
723 <!--[if IE]><script type="text/javascript" src="excanvas.js"></script><![endif]-->
724 <script type="text/javascript" src="convolutionalCode.js"></script>
725 <style type="text/css">
726     body{font-family:arial,verdana,helvetica;font-size: 10pt;}
727     a {text-decoration:none;color:black;margin:10pt;}
728     a:hover {text-decoration:underline;}
729     div {padding:5pt;}
730     span {font-size:8pt;}
731     #container {margin: 0pt auto;padding:0pt; width: 400pt;}
732     #input {border:solid #d0e1f5 1px;background-color: #edfaf6;margin-bottom:5pt;}
```

```
    #properties div {width: 110pt;float:left;}
    #output {border:solid #e77f8a 1px;background-color: #fbddfd;display:none;}
    .put{font-weight:bold;}
</style>
</head>
<body>
<div id="container">
<div id="input">
  <p class="put">Bitte uebertragene Kodefolge eingeben:</p>
  <form name="formular" action="">
    <p>
      <input type="text" name="word" value="000111011010011110010111000111
       011010100001001010111000000111110010011110010111111011110011110101 0
       110011100010100000111011111111011010000011101000011010011110010000010
       001010010000001010001001100100101011111100001001010111" size="45">
      <input type="button" name="button" value="berechnen"
          onclick="calculateCode(this.form.word.value);"><br>
      <input type="radio" name="coding" value="decode" checked>dekodieren
      <input type="radio" name="coding" value="encode">kodieren
    </p>
  </form>
  <a href="javascript:divProperties()">weitere Einstellungen</a>
  <div id="properties" style="display:none;">
    <form name="formProperties" action="">
      <div><p>
          G<span> 1</span>: <input type="text" name="h" value="1011" size="7">
          <br>G<span> 2</span>: <input type="text" name="h" value="1101" size="7">
          <br>G<span> 3</span>: <input type="text" name="h" value="1111" size="7">
        </p>
      </div>
      <div><p>
          P<span> 1</span>: <input type="text" name="p" value="100" size="7">
          <br>P<span> 2</span>: <input type="text" name="p" value="111" size="7">
          <br>P<span> 3</span>: <input type="text" name="p" value="111" size="7">
        </p>
      </div>
      <div><p><a href="javascript:removeRow()">weniger</a>
        <a href="javascript:addRow()">mehr</a>
        <br><input type="checkbox" name="puncturing" value="puncturing">Punktierung
        <br><input type="checkbox" name="termination" value="termination" checked>
                                                          Terminierung
        <br><input type="checkbox" name="graphics" value="graphics" checked>Grafik
        </p>
      </div>
      <p style="clear:left">HINWEIS: Es empfiehlt sich, bei Automaten mit vielen
         Speichern, keine Grafik zeichnen zu lassen! (Haeckchen bei "Grafik"
         loeschen)</p>
    </form>
  </div>
</div>
<div id="output"></div>
</div>
</div>
</body></html>
```