

Universität Karlsruhe

Lehrstuhl Informatik für Ingenieure und Naturwissenschaftler

Proseminar Zellulare Automaten WS 01/02

Erzeugung von Primzahlen mit Zellularen Automaten

Autor: Michael Biebl

Betreuer: Dr. Thomas Worsch

7. Februar 2002

Zusammenfassung

Es wird ein Algorithmus vorgestellt, der mit Hilfe eines eindimensionalen zellularen Automaten die Erkennung bzw. Erzeugung von Primzahlen ermöglicht. Arbeitsweise und Funktion des zellularen Automaten werden erläutert und Vorgehensweisen beschrieben, wie sich der Algorithmus praktisch implementieren lässt.

Inhaltsverzeichnis

1	Einleitung	3
1.1	Vorwort	3
1.2	Das Firing Squad Synchronisation Problem	3
2	Der Algorithmus in der Theorie	4
2.1	Vorbereitung	4
2.2	Konstruktion des zellularen Automaten	5
2.3	Arbeitsweise des zellularen Automaten	6
2.4	Der Algorithmus unter der Lupe	7
3	Implementierung	8
3.1	Vorüberlegungen	8
3.2	Vorgehensweise	9
4	Zusammenfassung	10

1 Einleitung

1.1 Vorwort

In der hier vorgelegten Proseminararbeit stütze ich mich zu großen Teilen auf die Arbeit von Patrick C. Fisher[1]. Ich werde ähnlich wie in [1] zuerst das Firing Squad Synchronisation Problem (FSSP) beschreiben, bevor ich zum eigentlichen Thema der Proseminararbeit, das Erzeugen von Primzahlen mit zellularen Automaten, komme. Das FSSP wird nur kurz behandelt und soll eine Möglichkeit darstellen, wie sich auf einer abstrakteren Ebene die Funktionsweise von zellularen Automaten beschreiben lässt. Man beschreibt nicht konkret die Überföhrungsfunktion sondern formuliert mit Hilfe von Signalen, die innerhalb des Gitters für den Informationsaustausch zuständig sind, die Abläufe im zellularen Automaten.

1.2 Das Firing Squad Synchronisation Problem

Beim Firing Squad Synchronisation Problem geht es darum, eine Gruppe von Soldaten, die alle in einer Reihe stehen, gleichzeitig in Anschlag gehen und feuern zu lassen. Übertragen lässt sich das auf ein eindimensionales Gitter endlicher Länge aus Automaten mit endlich vielen Zuständen. Diese Automaten sind, bis auf die beiden Automaten an den beiden Enden, identisch und zu Anfang in einem sog. Ruhezustand. Nach einem Eingabesignal an dem linken Automaten A_0 (oft auch "General" genannt) sollen nach einer bestimmten Zeitdauer alle Automaten gleichzeitig und zum ersten Mal in einen "Feuern-Zustand" übergehen. Die Größe des Gitters soll bei der Konstruktion keine Rolle spielen.

Für dieses Problem gibt es viele interessante Lösungen. Ich werde hier ein relativ einfaches Lösungsverfahren schildern.

Hierbei emittiert A_0 zu einem Zeitpunkt t_0 zwei Signale (hier a und b genannt) mit den Geschwindigkeiten $v_a = 1$ und $v_b = \frac{1}{3}$, n bezeichne die Anzahl der Automaten. Signal a wird am rechten Ende reflektiert und schneidet b in der Mitte des Gitters. Das veranlasst den Automaten im Schnittpunkt. Der Automat im Schnittpunkt der beiden Signale wechselt seinen Zustand und verhält sich wie ein Endautomat. Ist n geradzahlig liegt der Schnittpunkt zwischen zwei Automaten und folglich wechseln beide ihren Zustand. Vom Schnittpunkt aus werden in beide Richtung zwei neue Signale a und b losgeschickt. Der Vorgang wiederholt sich von neuem und unterteilt das Gitter in immer kleinere, untereinander synchronisierte Abschnitte. Man verfährt hier im Prinzip des Divide-and-Conquer so lange, bis die Abschnitte die Länge 1 haben und im nächsten Zeittakt alle Automaten in den "Feuern-Zustand" übergehen. Abbildung 1 zeigt den Verlauf der Signale.

Es gibt viele weitere Lösungen für das Firing Squad Synchronisation Problem, vor allem solche die es in optimaler Zeit, nämlich $t = 2n - 2$ schaffen. Der interessierte Leser sei hier auf [3, 4, 5] verwiesen.

Die Vorgehensweise bei der Konstruktion des eindimensionalen zellularen Automaten zur Erzeugung von Primzahlen wird ähnlich sein.

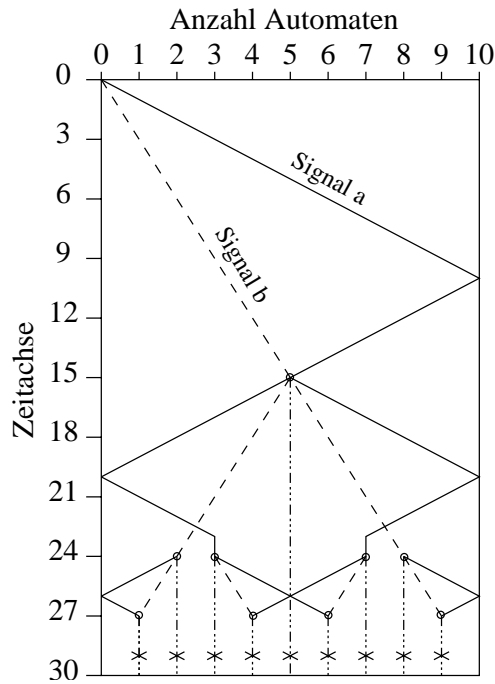


Abbildung 1: Lösung für das FSSP, $n=11$

2 Der Algorithmus in der Theorie

2.1 Vorbereitung

Folgende Bedingungen muss der zellulare Automat erfüllen:

- die Zahl der Automaten A_n ist unendlich, mit $n \in \mathbb{N}$.
- alle Automaten, bis auf A_0 sind identisch und zur Zeit $t = 0$ in einem Ruhezustand.
- der Zustand von A_n zum Zeitpunkt t hängt nur vom Zustand von A_{n-1} , A_n und A_{n+1} zum Zeitpunkt $t - 1$ ab, für alle $n > 0$.
- Ein- und Ausgaben des Gitters erfolgen über A_0 , wobei der Zustand von A_0 zum Zeitpunkt t nur von der Eingabe und dem Zustand von A_1 zum Zeitpunkt $t - 1$ abhängt.

Die Ausgabe des zellularen Automaten zum Zeitpunkt $3t+1$ ist 1, wenn t Primzahl und zum Zeitpunkt $3t$ die Eingabe 1 war, oder wenn die Eingabe 0 war zum Zeitpunkt $3t + 1$ und t keine Primzahl ist. Ansonsten ist die Ausgabe 0.

Verständlicher ausgedrückt bedeutet das folgendes: Wenn ich wissen möchte, ob die Zahl m Primzahl (keine Primzahl) ist, lege ich zum Zeitpunkt $t = 3m$ eine 1 an die Eingabe von A_0 . Erhalte ich zum Zeitpunkt $t = 3m + 1$ eine 1 ist die Zahl prim (nicht prim).

Zwar spielt die Anzahl der Zustände bei der Konstruktion nicht direkt eine Rolle, doch ist zu erwähnen, dass die hier beschriebenen Automaten A_n , mit $n \geq 1$, 37 verschiedene Zustände einnehmen können. A_0 kommt aufgrund der speziellen Position mit nur 11 verschiedenen Zuständen aus.

Da die einzelnen Automaten nur eine Zelle weit nach links und rechts “schauen” können, ist die maximale Geschwindigkeit mit der sich ein Signal innerhalb des Gitters ausbreiten kann $v_{max} = 1$. Um dann also den Faktor 3 in der jetzigen Konstruktion loszuwerden, könnte man immer drei Automaten zu neuen, größeren Automaten zusammenfassen. Diese neue Automaten A_0', A_1', \dots hätten dann eine theoretische Zustandszahl von $37^3 = 50653$. Weitere Informationen dazu findet man unter [2].

2.2 Konstruktion des zellularen Automaten

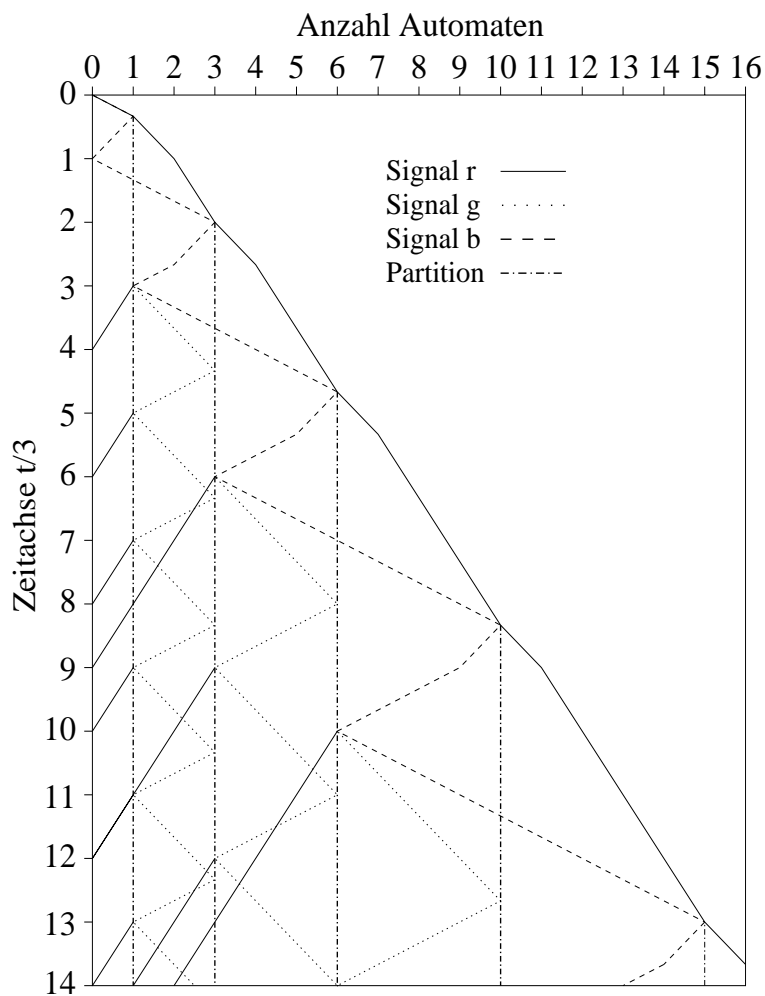


Abbildung 2: Lösung für das Primzahl-Problem

Bei der Konstruktion möchte ich wie in 1.2 vorgehen, und sie über Signale beschreiben, die sich innerhalb des Gitters ausbreiten. Abbildung 2 zeigt den Signalverlauf der Signale r, g und b. Dabei ist zu erwähnen, dass Signale auch überlagert werden können.

Ebenfalls wichtig ist eine Unterscheidung der Automaten in N (nonpartition) und P (partition), wobei zu Beginn alle A_n mit $n > 0$ im N-Zustand sind. Wechselt ein Automat in den P-Zustand, so bleibt er in diesem.

Tabelle 1: Verhalten der Automaten

Zeile	Typ	Eingabesignal zur Zeit t	Ankunft von	Ausgabesignal	nach	zur Zeit	neuer Typ
1	N	r und b	<i>links</i>	$\begin{cases} b \\ r \end{cases}$	<i>links</i> <i>rechts</i>	$\begin{cases} t + 2 \\ t + 2 \end{cases}$	P
2	N	r und nicht b	<i>links</i>	r	<i>rechts</i>	$t + 1$	N
3	N	r	<i>rechts</i>	r	<i>links</i>	$t + 1$	N
4	N	b und nicht r	<i>links</i>	b	<i>rechts</i>	$t + 3$	N
5	N	b	<i>rechts</i>	b	<i>links</i>	$t + 3$	N
6	N	g	<i>links</i>	g	<i>rechts</i>	$t + 2$	N
7	N	g	<i>rechts</i>	g	<i>links</i>	$t + 1$	N
8	P	r	<i>links</i>	r	<i>rechts</i>	$t + 1$	P
9	P	r	<i>rechts</i>	$\begin{cases} r \\ g \\ b \end{cases}$	<i>rechts</i> <i>rechts</i> <i>links</i>	$\begin{cases} t + 1 \\ t + 2 \\ t + 3 \end{cases}$	P
10	P	b	<i>links</i>	<i>unmöglich</i>			
11	P	b	<i>rechts</i>	b	<i>links</i>	$t + 3$	P
12	P	g	<i>links</i>	g	<i>links</i>	$t + 1$	P
13	P	g	<i>rechts</i>	$\begin{cases} g \\ b \end{cases}$	<i>rechts</i> <i>links</i>	$\begin{cases} t + 2 \\ t + 3 \end{cases}$	P

Tabelle 1 zeigt das Verhalten der einzelnen Automaten auf. Um den ganzen Prozess in Gang zu bringen muss noch folgendes gelten:

- Die erste Eingabe von 0 oder 1 kennzeichnet den Zeitpunkt $t = 0$
- Zum Zeitpunkt $t = 1$ schickt A_0 ein r und b -Signal nach rechts
- Zum Zeitpunkt $t = 4$ schickt A_0 ein b -Signal nach rechts

Das Ausgabeverhalten wurde schon im vorherigen Abschnitt geschildert und lässt sich folgendermaßen realisieren:

- Da 0 und 1 keine Primzahlen sind, ist für $t = 1$ und $t = 4$ die Ausgabe 1, falls zum Zeitpunkt $t - 1$ die Eingabe 0 war.
- Ist $t \leq 6$ und $t \equiv 0 \pmod{3}$ so gilt: Kommt bei A_0 zum Zeitpunkt t ein(kein) r -Signal an und war zum Zeitpunkt t die Eingabe 1, so ist für $t + 1$ die Ausgabe 0(1)
- ansonsten ist die Ausgabe 0

2.3 Arbeitsweise des zellularen Automaten

Das Sieb des Eratosthenes ist ein bekanntes Verfahren zur Bestimmung von Primzahlen. Man stelle sich eine Liste mit allen natürlichen Zahlen größer 2 vor. Dann markiert man 2 als Primzahl und

streicht alle Vielfachen davon aus der Liste. Man nimmt die nächstgrößere, nicht ausgekreuzte Zahl, markiert sie als Primzahl und streicht ebenfalls deren Vielfache aus der Listen und setzt das immer weiter fort.

Der hier vorgestellte Algorithmus basiert auf dem gleichen Prinzip, nur arbeitet er weniger effizient, da er auch für schon durchgekennzeichnete Zahlen deren Vielfache aus der Liste streicht. Für jedes $k > 2$ wird, beginnend mit k^2 , das Vielfache aus der Liste gestrichen. Dazu unterteilen die Signale b und r das Gitter in Blöcke der Länge l_i . Wobei die Blocklänge durch den Abstand zweier benachbarter P-Automaten definiert ist. Beginnend mit $l_0 = 2$ gilt $l_n = l_{n-1} + 1$. Das heisst, für jedes $k > 2$ existiert ein auch ein Block der Länge k . Innerhalb dieses Blocks beginnt zum richtigen Zeitpunkt t ein g-Signal mit der Periodendauer $3k$ zu oszillieren. Kommt das g-Signal am linken Ende des Blocks an wird ein r-Signal mit $\frac{2}{3}$ nach links losgeschickt. Das erste dieser r-Signale erreicht A_0 zum Zeitpunkt $3k^2$. Durch das mit $3k$ oszillierende g-Signal, kommen im Abstand $3k$ immer weitere r-Signale an. Das Ankommen eines r-Signales zum Zeitpunkt $3t$ bedeutet also, dass t einen echten Teiler hat und somit nicht prim ist.

Wichtigster Bestandteil des Algorithmuses ist also Tabelle 1, die den genauen Ablauf der Signale regelt. Dass diese wirklich das gewünschte Ergebnis liefert wird im nächsten Abschnitt gezeigt.

2.4 Der Algorithmus unter der Lupe

LEMMA 1: Für jedes $k \geq 1$ gilt: $A_{k(k+1)/2}$ wird durch gleichzeitige Ankunft eines r- und b-Signals zu einem P-Automaten. Dies geschieht zum Zeitpunkt $t = (3k^2 + k)/2 - 1$. Ein b-Signal erreicht danach zum Zeitpunkt $t = (3k^2 + 3k)/2$ den Automaten $A_{(k-1)k/2}$. Nur Automaten mit dem Index der Form $k(k+1)/2$ werden zu P-Maschinen.

BEWEIS: Beweis durch Induktion über k :

Induktionsanfang: Für $k=2$ gilt: A_3 wird zum Zeitpunkt $t = 6$ durch gleichzeitige Ankunft eines r- und b-Signals zum P-Automaten. Ein b-Signal wird nach rechts geschickt und erreicht A_1 zum Zeitpunkt $t = 9$. A_2 wird nicht zum P-Automaten, da ihn nie r- und b-Signale gleichzeitig erreichen.

Induktionsvoraussetzung: Die Behauptung gelte für ein $k \leq n$.

Induktionsschritt: $A_{n(n+1)/2}$ schickt zum Zeitpunkt $t = (3n^2 + n)/2 + 1$ ein r-Signal nach rechts. Dieses bewegt sich mit $v = \frac{1}{3}$ und erreicht somit A_x zum Zeitpunkt

$$t = \frac{1}{v}(x - (n^2 + n)/2 - 1) + (3n^2 + n)/2 + 1$$

Zum selben Zeitpunkt $t = (3n^2 + n)/2 + 1$ schickt $A_{(n-1)n/2}$ ein b-Signal mit $v = 1$ nach rechts, welches A_x zum Zeitpunkt

$$t = \frac{1}{v}(x - (n^2 - n)/2 - 1) + (3n^2 + 3n)/2 + 1$$

erreicht. Somit schneiden sich das r- und b-Signal zum Zeitpunkt

$$t = 3x - (3n^2 + 3n)/2 + (3n^2 + n)/2 - 2 = x - (n^2 - n)/2 + (3n^2 + 3n)/2$$

d.h wenn gilt

$$x = (n^2 + 3n + 2)/2 = (n + 1)(n + 2)/2$$

und

$$t = (n^2 + 3n + 2)/2 - (n^2 - n)/2 + (3n^2 + 3n)/2 = (3n^2 + 7n + 2)/2 = (3(n + 1)^2 + (n + 1))/2 - 1$$

Das bedeutet, dass zwischen $A_{n(n+1)/2}$ und $A_{(n+1)(n+2)/2}$ keine weiteren Automaten zu P-Automaten werden. Zum Schluß schickt $A_{(n+1)(n+2)/2}$ zum Zeitpunkt $t+2$ ein b-Signal nach links mit Geschwindigkeit $v = 1$, das $A_{n(n+1)/2}$ zum Zeitpunkt

$$t + 2 + \frac{1}{v}((n + 1)(n + 2)/2 - n(n + 1)/2 - 1) = (3n^2 + 9n + 6)/2 = (3(n + 1)^2 + 3(n + 1))/2$$

erreicht.

LEMMA 2: Für jedes $k \geq 2$ gilt: $A_{(k-1)k/2}$ schickt r -Signale nach links, die A_0 zu den Zeiten $t = 3k^2 + 3ik$, (mit $i \in \mathbb{N}$) erreichen.

BEWEIS: Das innerhalb des Blocks $A_{(k-1)k/2} - A_{k(k+1)/2}$ oszillierende g -Signal hat eine Periodendauer von $3k$. Jedesmal, wenn das g -Signal von $A_{(k-1)k/2}$ reflektiert wird, wird ein Takt später ein r -Signal mit $v = \frac{1}{3}$ nach rechts geschickt. Das erstmal geschieht das zum Zeitpunkt $t = (3k^2 + 3k)/2 + 3$. Daraus folgt, dass das b -Signal bei A_0 zum ersten Mal zum Zeitpunkt

$$t = (3k^2 + 3k)/2 + 3 + \frac{1}{v}((k - 1)k/2 - 1) = 3k^2$$

ankommt.

SATZ: Der in 2.2 und 2.3 beschriebene zellulare Automat, ist in der Lage für jedes x , mit $x = 3n$ und $n \geq 0$ zu entscheiden, ob x prim ist oder nicht.

BEWEIS: Für $x < 2$ sorgt A_0 für die Richtigkeit der Ausgabe. Für $x \geq 2$, wie Lemma 2 zeigt, bedeutet die Ankunft eines r -Signals bei A_0 zum Zeitpunkt t , dass t von der Form $3k(k + c)$ ist, mit $k \geq 2$, $c \geq 0$ und $k, c \in \mathbb{N}$. Folglich hat x einen echten Teiler und kann nicht prim sein. A_0 überprüft nur die Ankunft eines r -Signals und gibt einen Zeittakt später das Ergebnis aus.

3 Implementierung

3.1 Vorüberlegungen

Ziel der praktischen Implementierung des Algorithmuses war es, von der abstrakteren Beschreibung über Signale und Laufzeiten wieder zu einer auf Zustandstupeln definierten Übergangsfunktion zu gelangen. Es erwies sich als sinnvoll für alle Signale inklusive der Laufrichtung jeweils eine eigene Zustandsvariable einzuführen, was dazu führte, dass pro Automat folgende Zustandsvariablen verwendet wurden:

int: rL, rR, bL, bR, gL, gR (R steht jeweils für rechts, L für links)

boolean: *partition, leftBorder, rightBorder*

Die Variable `rightBorder` musste ich einführen um unangenehme Effekte auszugleichen, die dadurch entstanden, dass mir kein Gitter unendlicher Länge bei der Programmierung zur Verfügung stand, `leftBorder` um die spezielle Aufgabe von A_0 zu kennzeichnen. Initialisiert wurden alle Variablen per default mit 0 bzw. `false`, wobei natürlich bei A_0 `leftBorder = true` und bei A_l `rightBorder = true` gesetzt wurde. Zur Implementierung wählte ich die Programmiersprache Java, im speziellen das Framework von[7].

Die Endlichkeit des Gitters hat aber auch noch andere Auswirkungen. Es kann nicht mehr für alle $k > 1$ garantiert werden, dass ein entsprechender Block dieser Größe im Gitter existiert. Anders formuliert bedeutet das, dass für eine gegebene Länge l des Gitters die Richtigkeit bezüglich prim oder nicht prim, nur bis zu einer bestimmten Zahl z garantiert werden kann.

Sei k_{max} bei gegebenem Gitter die Länge des größten Blockes. Dann folgt daraus, dass für die Richtigkeit der Primzahlen nur bis zur Zahl $z = (k_{max} + 1)^2 - 1$ garantiert werden kann, da $m + 1$ bereits wieder Teiler sein könnte.

Wie groß muss man nun l wählen, wenn man bis zu einer gegebenen Zahl z für die Richtigkeit garantieren will? Aus $z = (k_{max} + 1)^2 - 1$ und $l = \frac{(k_{max} + 1)k_{max}}{2} + 1$ folgt, unter der Beachtung, k_{max} immer ganzzahlig sein muss,

$$l = \frac{(\lfloor \sqrt{z} \rfloor + 1)(\lfloor \sqrt{z} \rfloor)}{2} + 1$$

Daraus folgt, dass l annähernd linear mit der gesuchten Zahl z wächst.

Beispiel: Möchte man, dass bis zur Zahl $z = 150$ die Primzahlen garantiert richtig sind, dann muss man das Gitter mindestens $l = \frac{12 \cdot 13}{2} + 1 = 79$ groß machen.

3.2 Vorgehensweise

Um nun zu einer Überföhrungsfunktion zu kommen, die auf den oben definierten Zustandvariablen operiert, bietet sich folgendes Vorgehen an: innerhalb der Überföhrungsfunktion wird eine Kette von Regeln aufgebaut, wobei die Regeln bei absteigender Kette immer spezieller werden. Unabhängig vom Zustand der beiden Nachbarn zählt ein Automat bei jedem Schritt seine (integer) Variablen um 1 herunter, bis die jeweilige Variable den Wert 0 (Ruhezustand) erreicht.

Beispiel:

```

if (gL != 0) { // Regel 1
    gL--;
}
if (greenR != 0) { // Regel 2
    greenR--;
}

```

Nun muss es aber auch Bedingungen innerhalb der Überföhrungsfunktion geben, die eine der Variablen auf einen Wert $\neq 0$ setzen. Um beispielsweise ein Signal g nachzubilden, das sich mit $v = \frac{1}{2}$ nach rechts bewegt, definiert man je Automat eine (integer) Variable `gR` und fügt in die Überföhrungsfunktion nach den oben genannten Regeln, folgende zusätzliche Regel ein:

```

if(nachbar[links].gR == 1) { // Regel 3
    gR = 2;
}

```

Schon hat man das gewünschte Verhalten. Möchte man nun, dass das Signal, falls es von links kommend auf einen P-Automaten trifft, mit der Geschwindigkeit $v = 1$ zurückgeschickt wird, werden zwei zusätzliche Zustandvariablen eingeführt: eine (boolean) Variable `partition`, die per default mit `false` initialisiert wird und eine (integer) Variable `gL`. Ausserdem werden folgende Regeln der Kette hinzugefügt:

```

if(nachbar[rechts].gL == 1) { // Regel 4
    gL = 2;
}
if(nachbar[links].gR == 1 && partition) { // Regel 5
    gR = 0; // Wichtig
    gL = 1;
}
if(nachbar[rechts].gL == 1 && partition) { // Regel 6
    gL = 0; // Wichtig
    gR = 2;
}

```

Dabei ist zu beachten, dass in Regel 5 die Variable `gR` wieder auf 0 zurückgesetzt werden muss, um ein Weiterlaufen des Signal nach rechts zu verhindern. Ein von rechts kommendes `g`-Signal wird reflektiert und mit $v = 1$ nach links zurückgeschickt. Regel 6 schließlich sendet ein von rechts kommendes `g`-Signal mit $v = \frac{1}{2}$ zurück nach rechts, ein Weiterlaufen des Signal nach links wird ebenfalls verhindert. Mit diesen 6 Regeln haben wir also schon das Verhalten eines zwischen zwei P-Automaten oszillierenden `g`-Signals nachgebildet.

Nach diesem Schema wird nun Schritt für Schritt Tabelle 1 abgearbeitet und bei Bedarf neue Regeln und Variablen eingefügt. Eine komplette Auflistung der resultierenden Regelkette möchte ich mir hier ersparen, es sollte aber deutlich geworden sein, dass man durch dieses Vorgehen jedes beliebige Signalverhalten nachbilden kann.

4 Zusammenfassung

Der vorgestellte Algorithmus illustriert auf eindrucksvolle Weise, wie sich mit einem eindimensionalen zellularen Automaten Primzahlen in Echtzeit berechnen lassen. Die Beschreibung des zellularen Automaten über Signale und deren Laufzeiten war dabei ein wesentliches Element. Die dem Algorithmus zu Grunde liegende Idee stammt von Eratosthenes und wird oft als "Sieb des Eratosthenes" bezeichnet.

Es wurde schliesslich gezeigt, wie sich der zellulare Automat praktisch implementieren lässt und welchen Einschränkungen er dabei unterliegt.

Literatur

- [1] Patrick C. Fischer: Generating of Primes by a One-Dimensional Real Time Iterative Array. Journal of the Association for Computing Machinery, Vol. 12, No. 3 (July, 1965), pp. 388-394
- [2] S.N. Cole: Real Time Computation by iterative arrays of finite-state machines. Doctoral Thesis, Report BL-36, Harvard University, 1964
- [3] E. F. Moore: The Firing Squad Synchronisation Problem. In Sequential Machines: Selected Papers, Addison-Wesley, (1964), pp. 213-214.
- [4] A. Waksman: An Optimum Solution to the Firing Squad Synchronisation Problem. In press.
- [5] J. Mazoyer: A Six-State Minimal-Time Solution to the FSSP, Theoretical Computer Science 50 (1987), pp 59-98
- [6] J. Dana Eckart: Cellular und Cellang, Framework zur Simulation von Zellularen Automaten.
<http://www.cs.runet.edu/dana/ca/cellular.html>
- [7] Jörg Richard Weimar: JCasim, Java-Framework zur Simulation von Zellularen Automaten
<http://www.sc.cs.tu-bs.de/weimar/jcasim>