Fleißige Biber

Leibniz Universität Hannover

Institut für theoretische Informatik

Lena Daniela John

Matrikelnummer: 3217310

Erstprüfer: Prof. Dr. Heribert Vollmer
Zweitprüfer: Dr. Arne Meier
Betreuer: Prof. Dr. Heribert Vollmer

24. Februar 2020

Erklärung der Selbstständigkeit

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 24. Februar 2020

_____

Lena Daniela John

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Ein Teil der theoretischen Informatik beschäftigt sich mit der Berechenbarkeit von Problemen. Berechenbare Probleme stellen hier Aufgaben oder Funktionen dar, die durch einen Algorithmus formuliert werden können. Eine grundlegende Frage der Berechenbarkeitstheorie ist, ob diese Algorithmen terminieren. Um dies zu untersuchen, wird die Turingmaschine als Berechenbarkeitsmodell gewählt, da sich mithilfe der Church'schen These [s. 22, S. 41] Turing-Berechenbarkeit und Berechenbarkeit im intuitiven Sinne gleichsetzen lassen.

In dieser Arbeit wird ein Problem auf Berechenbarkeit sowie den Fortschritt in der Geschichte hinsichtlich zunehmender Erkenntnisse in der theoretischen Informatik untersucht. Im Jahr 1961 wurde das Thema der Fleißigen Biber von Tibor Radó beschrieben.

> *„Fleißige Biber sind spezielle Turingmaschinen, die möglichst viele Einsen auf das Band schreiben und die nach einer endlichen Anzahl Rechenschritte den Halt-Zustand einnehmen (also anhalten)." [26]*

Interessant hierbei ist, wieviele Einsen diese Turingmaschinen nun schreiben können, sodass sie noch in einen Endzustand (oder Halt-Zustand) wechseln.

Diese Beobachtung fasst Radó als Funktion $\Sigma(n)$ auf. Gegeben ist eine Anzahl an Nicht-Endzuständen n für die Turingmaschine, der Funktionswert stellt die maximale Anzahl an Einsen dar, die auf das Band geschrieben werden können. In Kapitel 2 wird gezeigt, dass diese Funktion unberechenbar ist.

In Radós Paper *On non-computable functions* [18] dokumentiert er das Problem als *Busy-Beaver Game*. Hierin treten Turingmaschinen mit festem n gegeneinander an und Gewinner sind diejenigen Turingmaschinen, die am meisten Einsen schreiben können. Das galt scheinbar als Ansporn für andere Interessierte und somit gab es auf dem Gebiet der Fleißigen Biber große Fortschritte. Zu einem festen n wurden immer fleißigere Fleißige Biber gefunden (mehr dazu in Kapitel 3).

Zum Abschluss der Arbeit wird eine programmierte Simulation einer Turingmaschine dokumentiert. Damit gibt es die Möglichkeit, die Fleißigen Biber zu simulieren und Abbildungen der Konfigurationen generieren zu lassen. Mit diesen Abbildungen kann die Arbeitsweise einer Turingmaschine besser nachvollzogen werden.

1.2 Grundlagen zu Turingmaschinen

Die Turingmaschine wurde nach Alan Turing benannt und ist ein vereinfachtes Modell für einen Computer. In seiner Veröffentlichung *On computable numbers* [20] beschreibt er *automatic machines* (oder auch a-Maschinen) und *choice machines* (oder auch c-Maschinen), die heute besser bekannt sind als deterministische und nichtdeterministische Turingmaschinen.

Eine Turingmaschine kann informell als ein Modell beschrieben werden, das auf einem unendlichen Band mit unbegrenztem Speicher arbeitet. Auf diesem Band befindet sich die Eingabe in einzelnen Zellen. Zudem besitzt die Turingmaschine einen Lese-/Schreibkopf, der sich auf dem Band mit jedem Konfigurationsübergang nach links und rechts bewegen kann. Weiterhin gibt es eine Programmeinheit, welche die Steuerung des Kopfes übernimmt. Die grundlegende Funktionsweise einer Turingmaschine wird hier als bekannt voraus gesetzt.

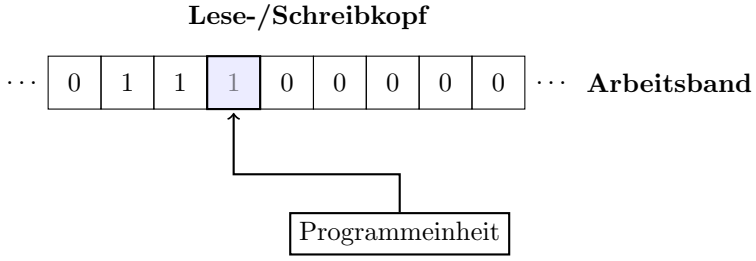

Abbildung 1: Informelle Arbeitsweise einer Turingmaschine

Des Weiteren kann eine Turingmaschine formal als 7-Tupel beschrieben werden. Diese Definition ist aus [22] entnommen.

Definition 1. *Eine Turingmaschine M ist ein Tupel $M := (Z, \Omega, \Gamma, \delta, z_0, \square, E)$, wobei für die einzelnen Komponenten gilt:*

- *Z ist die Menge der Zustände*
- *Ω ist das Eingabealphabet*
- *$\Gamma \supset \Omega$ ist das Bandalphabet*
- *δ ist die Übergangsfunktion*

  Bei deterministischen Turingmaschinen (DTM, TM) gilt:
  $\delta : Z \times \Gamma \to Z \times \Gamma \times \{L, N, R\}$

  Bei nichtdeterministischen Turingmaschinen (NTM) gilt:
  $\delta : Z \times \Gamma \to \mathcal{P}(Z \times \Gamma \times \{L, N, R\})$

- $z_0 \in Z$ *ist der Startzustand*
- $\square$ *ist das Leersymbol oder Blank*
- $E \subseteq Z$ *ist die Menge der Endzustände*

Bei den Turingmaschinen, die Fleißige Biber beschreiben, handelt es sich um Einband-Turingmaschinen. Insofern gibt es ein Eingabeband, welches vorher komplett mit Nullen beschrieben ist und dann mit Einsen beschrieben wird (s. Abbildung 1), damit entfällt $\square$. Außerdem sind diese Turingmaschinen deterministisch, was sich aus der jeweiligen Konstruktion der Übergangsfunktion ergibt. Kopfbewegungen sind zudem nur nach links oder rechts erlaubt.

Definition 2. *Eine Fleißige Biber-Turingmaschine ist eine Turingmaschine* $M := (Z, \Omega, \Gamma, \delta, z_0, 0, E)$*, wobei gilt:*

- Z *ist die Menge der Zustände, wobei* $|Z| = n+1$*,* $n \in \mathbb{N}$ *(s.Radó-Funktion* $\Sigma(n)$ *in Abschnitt 2.1)*
- $\Omega = \{0, 1\}$ *ist das Eingabealphabet*
- $\Gamma \supset \Omega$*,* $\Gamma = \{0, 1\}$ *ist das Bandalphabet*
- $\delta : Z \times \Gamma \to Z \times \Gamma \times \{L, R\}$*, da TM deterministisch*
- $z_0 \in Z$ *ist der Startzustand*
- *0 ist das Leersymbol*
- $E \subseteq Z$*,* $E = \{z_e\}$ *ist die Menge der Endzustände*

Die Fleißige Biber-Turingmaschinen sollen nach einer endlichen Anzahl an Konfigurationsübergängen anhalten. Radó definiert dafür die Schrittfunktion S (s. Abschnitt 2.1).

Für Maschinen mit kleinem n, also wenigen Zuständen, kann $\Sigma(n)$ einfach zu bestimmen sein. Je größer das n wird, desto größer kann der Funktionswert von Σ ausfallen. Neben vielen verschiedenen Möglichkeiten für die Konstruktion der Übergangsfunktion bei großen n, die untersucht werden müssen, ist es ebenfalls komplex, festzustellen, ob solch eine Maschine anhält.

Um festzustellen, ob Teilnehmer des Fleißigen Biber-Spiels halten, wird eine Schrittzahl verlangt, für die die Maschine getestet werden kann. Die Schrittfunktion S bildet auf die maximal mögliche Anzahl an Schritten ab, die mit gegebenen n getätigt werden können.

1.2.1 Fleißiger Biber mit einem Zustand

Bevor im weiteren Verlauf die theoretischen Aspekte der Fleißigen Biber beleuchtet werden, beschreibt dieser Abschnitt zunächst ein Beispiel für einen Fleißigen Biber. Hier wird ein Fleißiger Biber mit einem Nicht-Endzustand dargestellt, also $n = 1$. Zunächst wird diese Turingmaschine mit einem Nicht-Endzustand formal beschrieben als

$M_1 := (\{z_0, z_e\}, \{0, 1\}, \{0, 1\}, \delta, z_0, 0, \{z_e\})$, wobei

$$\delta : \quad z_0\ 0 \to z_e\ 1\ R.$$

Die δ-Funktion besitzt einen Übergang. Die Turingmaschine startet in Zustand z_0, liest eine Null, geht über in den Endzustand z_e, schreibt eine Eins und bewegt den Lese-/Schreibkopf nach rechts. Damit kann ein Fleißiger Biber mit einem Zustand eine Eins schreiben, bevor er anhält. Offenbar ist damit $\Sigma(1) = 1$.
Der Fleißige Biber mit einem Zustand kann mit Hilfe eines Modells dargestellt werden. Dabei ist ein Übergang im Modell mit einem lesenden Zeichen, einem Trennsymbol „|“, einem schreibendem Zeichen und einer Kopfbewegung dargestellt. Die Zustände im Modell gleichen sich mit denen aus der δ-Funktion. Ein Endzustand wird mit einer zusätzlichen Umrandung dargestellt:

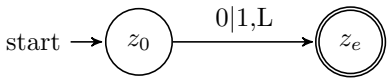

Abbildung 2: Modell für Fleißigen Biber mit einem Zustand

Analog zur Übergangsfunktion startet das Modell in Zustand z_0 und geht über in den akzeptierenden Zustand z_e. Falls eine Null gelesen wird, schreibt es dann eine Eins und bewegt den Kopf um eine Zelle nach rechts. Damit wird die eine Eins in einem Schritt erzeugt (damit auch $S(1) = 1$, s. Abschnitt 2.1 für die Funktion S).
Für $n = 1$ ist die Konstruktion der Turingmaschine trivial. Das Wachstum von $\Sigma(n)$ nimmt jedoch so stark zu, dass die Funktion für große n nicht mehr zu berechnen ist.

2 Unberechenbarkeit

Dieser Abschnitt definiert den Begriff der (Un-)Berechenbarkeit und klärt, wie dieser mit dem Begriff der Entscheidbarkeit zusammenhängt. Alle Definitionen aus diesem Abschnitt gehen aus [22] hervor.

> *„Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt berechenbar, falls es ein Rechenverfahren bzw. einen Algorithmus gibt [...], das f berechnet, d. h. gestartet mit Eingabe $(n_1, ..., n_k) \in \mathbb{N}^k$ hält der Algorithmus nach endlich vielen Schritten mit Ausgabe $f(n_1, ..., n_k)$." [s. 22, S. 3]*

Die Church'sche These hilft dabei den Begriff der Berechenbarkeit weiter verständlich zu machen.

Church'sche These. *Eine Funktion ist berechenbar im intuitiven Sinne genau dann, wenn sie Turing-berechenbar ist.*[1]

Definition 3. *Eine Funktion $f : \Omega^* \to \Delta^*$ heißt Turing-berechenbar, falls es DTM M gibt, sodass für alle $x \in \Omega^*$ und $y \in \Delta^*$ gilt:*
$f(x) = y \Rightarrow$ M mit Eingabe x hält mit $\square...\square y\square...\square$ auf dem Ausgabeband
$f(x)$ undefiniert $\Rightarrow$ M mit Eingabe x stoppt nicht

Eine DTM stoppt nicht, wenn sie in eine Endlosschleife gerät, und somit keinen Endzustand annimmt. Eine Funktion ist folglich *unberechenbar*, wenn keine DTM existiert, die für jeden Wert des Definitionsbereiches eine entsprechende Ausgabe produziert und auf dieser hält.

Turingmaschinen berechnen nicht nur Funktionen, sondern entscheiden auch Sprachen. Eine Sprache A ist eine Teilmenge der Worte über dem Alphabet Ω.

Definition 4. *Eine Sprache $A \subseteq \Omega^*$ heißt entscheidbar, wenn die Funktion $c_A(w) : \Omega^* \to \{0, 1\}$ mit*

$$c_A(w) = \begin{cases} 1, & falls\ w \in A \\ 0, & sonst \end{cases}$$

(Turing-)berechenbar ist. c_A heißt charakteristische Funktion von A.

Somit ist eine Sprache entscheidbar, wenn es eine DTM gibt, die auf allen Eingaben hält.

[1] berechenbar im intuitiven Sinne ist hier nicht formal gefasst

Definition 5. *Eine Sprache $A \subseteq \Omega^*$ heißt semi-entscheidbar, wenn die Funktion $\chi_A(w) : \Omega^* \to \{0, 1\}$ mit*

$$\chi_A(w) = \begin{cases} 1, & falls\ w \in A \\ undefiniert, & sonst \end{cases}$$

berechenbar ist.

Bei semi-entscheidbaren Sprachen muss damit unter Umständen *unendlich lange* auf eine Entscheidung gewartet werden.

Definition 6. *Sei $A \subseteq \Omega^*$. Es gilt:*
A ist entscheidbar genau dann, wenn A und $\bar{A}$ sind semi-entscheidbar.

Ein Beispiel für eine Sprache, welche semi-entscheidbar und gleichzeitig unentscheidbar ist, ist das *Halteproblem*.

Definition 7. *Das Halteproblem ist die Sprache*

$$H = \{w\#x \mid M_w\ hält\ bei\ Eingabe\ x\}$$

Das Halteproblem beschreibt, ob eine Turingmaschine bei gegebener Eingabe hält. Dieses Problem ist unentscheidbar, da es keine Turingmaschine gibt, welche die charakteristische Funktion des Halteproblems berechnet, und das Komplement des Halteproblems nicht semi-entscheidbar ist.

Entgegen der Ansicht der Mathematiker Anfang des 20. Jahrhunderts, z.B. David Hilbert[2], ist somit nicht jedes (mathematische) Problem algorithmisch lösbar. Daher ist die Unentscheidbarkeit des Halteproblems eine wichtige Erkenntnis in Hinblick auf die Unberechenbarkeit der Radó-Funktion.

2.1 Radó-Funktion Σ und Schrittfunktion S

Alle Ergebnisse aus diesem Kapitel, sofern nicht anders angegeben, gehen aus [18] hervor. Um für einen Fleißigen Biber mit n Zuständen den Fleißigsten festzustellen, also die Turingmaschine, welche am meisten Einsen schreibt, müssen alle möglichen Biber getestet werden. Für einen Fleißigen Biber nach Definition 2 mit n Zuständen, gibt es demnach

$$N(n) = (4 \cdot (n + 1))^{2n}$$

mögliche Turingmaschinen.
Diese Zahl setzt sich zusammen aus den δ-Übergängen. Für n Zustände und

[2] „[..] daß ein jedes bestimmte mathematische Problem einer strengen Erledigung notwendig fähig sein müsse, sei es, daß es gelingt, die Beantwortung der gestellten Frage zu geben, sei es, daß die Unmöglichkeit seiner Lösung [...] dargethan wird. [...] “ [s. 5, S. 261]

zwei mögliche lesende Symbole, gibt es zwei mögliche Kopfbewegungen, zwei mögliche schreibende Symbole und $n + 1$ mögliche Folgezustände (n Zustände mit einem zusätzlichen Endzustand).
Hierbei ist die Schwierigkeit, nicht-haltende Turingmaschinen auszusondern. Wie im Abschnitt zuvor bereits erwähnt, ist es nicht entscheidbar, ob eine Turingmaschine auf einer gegebenen Eingabe hält.

Sei $E(n)$ definiert als die Menge aller validen Fleißigen Biber-Eingaben. Als Eingabe wird hier das Tupel (M, s) aufgefasst, wobei M die Fleißige Biber-Turingmaschine und s die Anzahl der endlichen Konfigurationsübergänge sind, bevor die Turingmaschine hält. Eine Eingabe ist valide, falls M nach höchstens s Schritten hält. Es ist damit entscheidbar, ob $(M, s) \in E(n)$.

Die Anzahl der möglichen validen Einträge $N_e(n)$ ist durch die Anzahl aller möglichen Turingmaschinen begrenzt.

Lemma 1. $1 < N_e(n) < N(n) = (4 \cdot (n + 1))^{2n}$.

Sei $\sigma(M, s)$ die Anzahl Einsen, die M mit s Schritten schreibt. Dann sei $\Sigma(n)$ - die wohldefinierte Radó-Funktion - die maximale Anzahl Einsen, die für ein gegebenes n geschrieben werden.

Definition 8. $\Sigma(n) = max\{\sigma(M, s) \mid (M, s) \in E(n)\}$.

Eine nicht-leere endliche Menge von positiven ganzen Zahlen hat immer ein größtes Element, dieses ist der Wert von $\Sigma(n)$.
Die zweiten Komponenten der validen Einträge (M, s) bilden ebenfalls eine endliche nichtleere Menge von positiven ganzen Zahlen. Sei $S(n)$ das größte Element dieser Menge, also beschreibt $S(n)$ die maximale Anzahl an Schritten, die eine Turingmaschine mit n Zuständen leisten kann. Dabei kann der Wert von $S(n)$ zu einer anderen Turingmaschine gehören als der von $\Sigma(n)$. Es gilt:

Lemma 2. $S(n) \geq \Sigma(n)$

Lemma 2 gilt offensichtlich, da in einem Schritt höchstens eine Eins geschrieben werden kann.

2.1.1 Unberechenbarkeit der Radó-Funktion

Der folgende Beweis ist sinngemäß [24] entnommen.
$\Sigma(n)$ beschreibt nun nach Definition 8 die maximale Anzahl Einsen, die mit n Zuständen geschrieben werden können.
Angenommen $\Sigma(n)$ ist berechenbar, dann gibt es eine Turingmaschine M, die diese Funktion berechnet (s. Definition 3). Außerdem nehmen wir an, dass Σ eine Folge von hintereinander stehenden Einsen als Eingabe für n sieht und ebenfalls $\Sigma(n)$ Einsen auf das Band schreibt.

Wir definieren folgende Maschinen:

Double	Turingmaschine, die den Bandinhalt verdoppelt
Inc	Turingmaschine, die eine zusätzliche 1 auf das Band schreibt
Create_n_0	Turingmaschine, die n_0 Einsen auf ein leeres Band schreibt, sie hat n_0 Zustände

Kreieren wir nun die Komposition **Double;M;Inc**, welche n_0 Zustände besitzt. Diese Maschine verdoppelt den Bandinhalt, simuliert M und schreibt eine weitere Eins auf das Band.
Defininiren wir des Weiteren $\hat{M}$ als die Komposition **Create**_n_0**;Double;M;Inc**. $\hat{M}$ besitzt damit $N = 2n_0$ Zustände und schreibt n_0 Einsen auf das Band, verdoppelt diese, simuliert dann M mit den Einsen als Eingabe und schreibt eine weitere Eins. $\hat{M}$ wird damit $\Sigma(N) + 1$ Einsen drucken.
Dies ist ein Widerspruch zur Definition von $\Sigma(n)$, da mit N Zuständen höchstens $\Sigma(N)$ Einsen geschrieben werden können, hier werden jedoch $\Sigma(N) + 1$ Einsen geschrieben.
Somit kann es keine solche Turingmaschine M geben, die $\Sigma(n)$ berechnet.

Satz 1. *Die Radó-Funktion ist unberechenbar.*

Der Beweis funktioniert für die Schrittfunktion S ähnlich und führt ebenfalls zur Unberechenbarkeit dieser.
Aus der Erkenntnis von Satz 1 kann nun eine Aussage über das Halteproblem getroffen werden. Angenommen das Halteproblem H aus Definition 7 ist entscheidbar, dann wäre auch die Radó-Funktion berechenbar. Folgender Algorithmus zeigt die Unentscheidbarkeit des Halteproblems.

```
   Eingabe : w#n
 1 σ = ∅;
 2 für alle TM's M_q mit n Zuständen tue
 3 |   wenn q#n ∈ H dann
 4 |   |   Simuliere M_q(n) und bestimme Anzahl der Einsen in der Ausgabe;
 5 |   |   Füge diesen Wert zur Menge σ hinzu;
 6 |   Ende
 7 Ende
 8 Sei Σ der maximale Wert der Menge σ;
 9 wenn w#n ∈ H dann
10 |   Simuliere M_w(n) und bestimme Anzahl der Einsen in der Ausgabe;
11 |   wenn dieser Wert= Σ dann  akzeptiere;
12 |   sonst verwerfe;
13 Ende
```

Im Algorithmus wird eine spezielle Turingmaschine darauf geprüft, ob sie der fleißigste Fleißige Biber für n Zustände ist. Zunächst werden alle Turingmaschinen für n Zustände simuliert und ihre geschriebenen Einsen gespeichert.

Da nach Annahme das Halteproblem entscheidbar ist, ist es möglich, alle Maschinen so zu testen. Dann wird die maximale Anzahl an Einsen der Menge σ bestimmt. Falls die eingegebene Turingmaschine genau diese Anzahl schreibt, ist sie der fleißigste Fleißige Biber.
Da nach Satz 1 und vorangegangenem Beweis die Radó-Funktion unberechenbar ist, kann das Halteproblem nicht entscheidbar sein.

3 Fortschritt in der Geschichte

Nachdem das vorhergehende Kapitel geklärt hat, dass die Radó-Funktion unberechenbar ist, zeigt das nachstehende Kapitel einen Überblick über die partiellen Lösungen für diese Funktion. Außerdem werden verschiedene Herangehensweisen vorgestellt, da es bekanntlich keine effektive Möglichkeit der Berechenbarkeit gibt. Auf dem Gebiet der Fleißigen Biber konnte Interessantes bezüglich nicht-haltenden Turingmaschinen beobachtet werden. Einige Ergebnisse davon möchte ich in diesen Kapiteln vorstellen.
Zunächst ist der folgenden Tabelle die Anzahl möglicher Turingmaschinen zu gegebener Anzahl an Zuständen nach Kapitel 2.1 zu entnehmen. Diese kann eine Vorstellung für die Ausmaße der zu testenden Turingmaschinen geben, um einen Fleißigen Biber zu finden.

Anzahl Zustände n	Anzahl Turingmaschinen
1	64
2	20.736
3	16.777.216
4	$2,56 * 10^{10}$
5	$\approx 6,34 * 10^{13}$
6	$\approx 2,32 * 10^{17}$
7	$\approx 1,18 * 10^{21}$

Tabelle 1: Anzahl möglicher Turingmaschinen zu einer gegebenen Anzahl an Zuständen

Des Weiteren ist in der nachstehenden Tabelle ein Überblick über die angenommenen Funktionswerte von $\Sigma(n)$ und $S(n)$ zu Fleißigen Bibern mit einem bis sieben Zuständen zu finden. Darauf, wie diese Werte zustande kommen, wird in den anschließenden Kapiteln eingegangen. Zu beachten ist noch einmal, dass sich der Wert der Schrittfunktion nicht unbedingt auf die gleiche Turingmaschine bezieht, die den Wert von $\Sigma(n)$ produziert.

Anzahl Zustände n	$\Sigma(n)$	Schrittfunktion $S(n)$
1	1	1
2	4	6
3	6	21
4	13	107
5	4.098	47.176.870
6	$> 3,514 * 10^{18.267}$	$7,4 * 10^{36.534}$
7	$> 10^{10^{10^{10^{18.705.353}}}}$	$> 10^{10^{10^{10^{18.705.353}}}}$

Tabelle 2: Funktionswerte von $\Sigma(n)$ und $S(n)$ zu gegebenen n

3.1 Fleißiger Biber mit zwei Zuständen

Das Fleißige Biber-Problem für zwei Zustände konnte noch von Tibor Radó gelöst werden. Im Jahr 1962 veröffentlicht er in *On a simple source for non-computable functions* weitere Studien zur unberechenbaren Σ-Funktion und erwähnte lediglich, dass $\Sigma(2) = 4$ mithilfe des IBM 7090 gezeigt werden konnte [s. 17, S. 77].
Außerdem wird in *Computer studies of Turing Machine Problems* von Tibor Radó und Shen Lin folgendes angegeben:

„As an exercise in a seminar, it has been shown that $\Sigma(2) = 4$*.“ [19]*

Um nun den fleißigsten Biber zu finden, können folgende Überlegungen getroffen werden. Zunächst ist der erste Übergang festgelegt als $z_0\ 0 \rightarrow z_1 1R$. Ein Zustandswechsel muss stattfinden, damit die Maschine nicht in eine Endlosschleife gerät. Eine Kopfbewegung nach links würde ein äquivalentes gespiegeltes Programm ergeben und falls keine Eins geschrieben wird, würde mit einem Tausch der zwei Zustände eventuell eine schnellere Lösung gefunden werden. Zudem soll das Programm nach endlicher Schrittzahl halten, so dass eine „rechte“ Seite die Form $z_e 1R$ hat. Dabei ist die Kopfbewegung irrelevant und eine Eins wird in jedem Fall geschrieben, da eine Null nicht zu einem Gewinner führen würde.
Des Weiteren ist durch Festlegen des ersten Schrittes bekannt, dass sich die Maschine danach im Übergang $z_1\ 0 \rightarrow$ befinden wird. Dabei kann ebenfalls eine Kopfbewegung nach links vorausgesetzt werden, da hier sonst eine Schleife entsteht.
Nun werden übrige Übergänge weiterhin reduziert, z.B. durch Erkennen von Schleifen, und kombinatorisch getestet. Dabei entsteht folgender Fleißiger Biber, entnommen aus der historischen Studie von Michel [15]:

$M_2 := (\{z_0, z_1, z_e\}, \{0, 1\}, \{0, 1\}, \delta, z_0, 0, \{z_e\})$, wobei

$$\begin{aligned} \delta: \quad & z_0\ 0 \rightarrow z_1\ 1\ R, \\ & z_0\ 1 \rightarrow z_1\ 1\ L, \\ & z_1\ 0 \rightarrow z_0\ 1\ L, \\ & z_1\ 1 \rightarrow z_e\ 1\ R. \end{aligned}$$

Das zugehörige Modell sieht wie folgt aus:

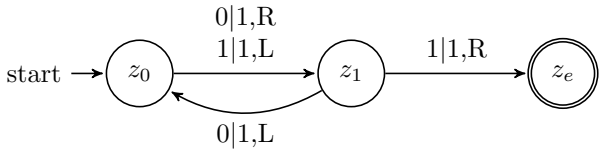

Abbildung 3: Modell für Fleißigen Biber M_2 mit zwei Zuständen

Die Simulation von M_2 ergibt folgende Konfigurationsübergänge:

Delta-Übergang	Konfiguration
Initial	$z_0 0$
$z_0\ 0 \rightarrow z_1 1R$	$1z_1 0$
$z_1\ 0 \rightarrow z_0 1L$	$z_0 11$
$z_0\ 1 \rightarrow z_1 1L$	$z_1 011$
$z_1\ 0 \rightarrow z_0 1L$	$z_0 0111$
$z_0\ 0 \rightarrow z_1 1R$	$1z_1 111$
$z_1\ 1 \rightarrow z_e 1R$	$11z_e 11$

Tabelle 3: Konfigurationen bei Simulation des Fleißigen Bibers M_2 mit zwei Zuständen

Nach Simulation von M_2 befinden sich vier Einsen auf dem Arbeitsband, damit ist $\Sigma(2) = 4$. Außerdem ist Tabelle 3 zu entnehmen, dass sechs Schritte gemacht werden, bevor M_2 anhält. Mithilfe des Computers oder manuellem Testen kann auch $S(2) = 6$ gezeigt werden.
Nachdem Radó in [17] Beobachtungen über die Σ-Funktion niedergeschrieben hatte und $\Sigma(2)$ gezeigt werden konnte, schien die gleiche Aufgabe für drei und mehr Zustände fast unmöglich, wie Radó zum Ausdruck bringt:

> *„In any case, even though skilled mathematicians and experienced programmers attempted to evaluate $\Sigma(3)$ and $S(3)$, there is no evidence that any presently known approach will yield the answer, even if we avail ourselves of high-speed computers and elaborate programs. As regards $\Sigma(4)$, $S(4)$, the situation seems to be entirely hopeless at present.“ [s. 19, S. 78]*

Trotzdem veröffentlichte Radó, zusammen mit Lin, zwei Jahre später einen Beweis mit *Computer Studies of Turing Machine Problems* für das Fleißige Biber-Problem mit drei Zuständen.

3.2 Fleißiger Biber mit drei Zuständen

Für eine Turingmaschine mit drei Zuständen gibt es nach Tabelle 1 bereits 16.777.216 Möglichkeiten, diese zu konstruieren. Es ist ineffizient alle Maschinen zu testen. Deswegen ist es eine sinnvolle Überlegung, die zu prüfendenden Maschinen im Vorhinein zu reduzieren.
Der fleißigste Fleißige Biber für drei Zustände stammt von Tibor Radó und Shen Lin aus dem Jahr 1965, ihre Ergebnisse sind im bereits erwähnten Paper *Computer studies of Turing Machine Problems* [19] dokumentiert.
Tibor Radó und Shen Lin haben zunächst durch Normalisierung eine prinzipielle Form der δ-Funktion erstellt. Zum Beispiel wird, wie auch schon bei dem Biber mit zwei Zuständen, im letzten Konfigurationsübergang zu dem Endzustand in jedem Fall noch eine Eins geschrieben und der Kopf bewegt sich nach rechts, so auch im Startübergang. Falls sich die Maschine nach links bewegt, ist sie lediglich eine Spiegelung der vorherigen Maschine und kann außer Acht gelassen werden. Mit weiteren Restriktionen verbleiben dann vier noch nicht definierte Übergänge, *lots* genannt, einer davon soll in den Endzustand wechseln. Damit können die Maschinen auf 82.944 beschränkt werden.
Radó und Lin äußerten ferner die Vermutung, dass eine Turingmaschine mit drei Zuständen nach höchstens 21 Schritten hält. Deswegen wurden die Maschinen für höchstens diese 21 Schritte simuliert. Dabei wurden fünf Maschinen identifiziert, die jeweils sechs Einsen produzieren und eine, die 21 Schritte tätigt [s. 19, S. 7].

Ein möglicher dieser Fleißigen Biber kann wie folgt aussehen:

$M_3 := (\{z_0, z_1, z_2, z_e\}, \{0, 1\}, \{0, 1\}, \delta, z_0, 0, \{z_e\})$, wobei

$$\begin{aligned}
\delta: \quad & z_0\ 0 \to z_1\ 1\ L,\\
& z_0\ 1 \to z_2\ 1\ R,\\
& z_1\ 0 \to z_0\ 1\ R,\\
& z_1\ 1 \to z_1\ 1\ L,\\
& z_2\ 0 \to z_1\ 1\ R,\\
& z_2\ 1 \to z_e\ 1\ L.
\end{aligned}$$

Das zugehörige Modell sieht wie folgt aus:

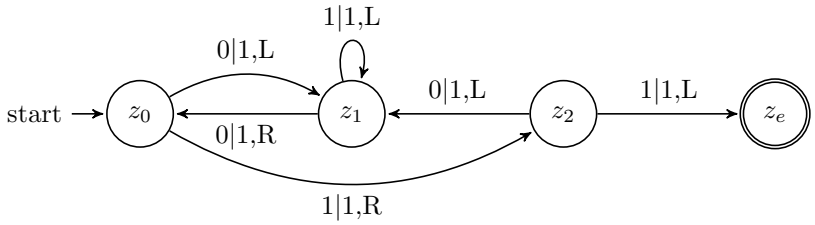

Abbildung 4: Modell für Fleißigen Biber M_3 mit drei Zuständen

Delta-Übergang	Konfiguration
Initial	$z_0 0$
$z_0\ 0 \to z_1 1L$	$z_1 01$
$z_1\ 0 \to z_0 1R$	$1 z_0 1$
$z_0\ 1 \to z_2 1R$	$11 z_2 0$
$z_2\ 0 \to z_1 1R$	$111 z_1 0$
$z_1\ 0 \to z_0 1R$	$1111 z_0 0$
$z_0\ 0 \to z_1 1L$	$111 z_1 11$
$z_1\ 1 \to z_1 1L$	$11 z_1 111$
$z_1\ 1 \to z_1 1L$	$1 z_1 1111$
$z_1\ 1 \to z_1 1L$	$z_1 11111$
$z_1\ 1 \to z_1 1L$	$z_1 011111$
$z_1\ 0 \to z_0 1R$	$1 z_0 11111$
$z_0\ 1 \to z_2 1R$	$11 z_2 1111$
$z_2\ 1 \to z_e 1L$	$111 z_e 111$

Tabelle 4: Konfigurationen bei Simulation des Fleißigen Bibers M_3 mit drei Zuständen

Auf dem Arbeitsband stehen nach der Simulation von M_3 sechs Einsen, also $\Sigma(3) = 6$. Außerdem werden 13 Konfigurationsübergänge durchlaufen.

Nun bleibt noch zu zeigen, dass alle übrigen Maschinen, die nicht nach 21 Übergängen den Halt-Zustand eingenommen haben, niemals halten werden. Radó und Lin identifizierten zwei Muster: *partial and total recurrence pattern* [s. 19, S. 9].

Totale Wiederkehr tritt ein, wenn in einem Zustand nach einer bestimmten Anzahl an Schritten immer wieder dasselbe Zeichen gelesen wird und die Umgebung auf dem Arbeitsband ebenfalls dieselbe bleibt, wie aus folgender Abbildung ersichtlich. Damit halten Turingmaschinen, die dieses Muster aufzeigen, nicht.

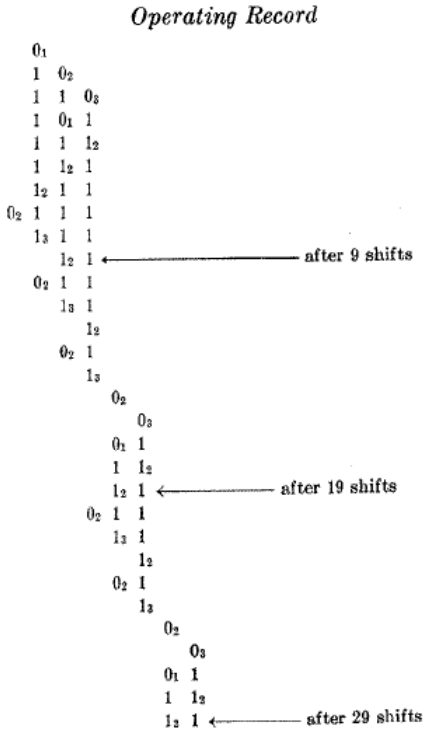

FIG. 3. Operating record of the Turing machine whose serial number is 73075226 (octal) showing the total recurrence pattern

Abbildung 5: Konzept der totalen Wiederkehr [s. 19, S. 8]

In jeder Zeile ist ein Teil des Arbeitsbandes abgedruckt, der Index an einer Zelle beschreibt den aktuellen Zustand und somit auch die Kopfposition. Nach jeder Zeile geht die Maschine einen Schritt weiter. In Abbildung 5 ist der Inhalt des Arbeitsbandes nach 9, 19 und 29 Schritten der Gleiche. Die Maschine funktioniert so, dass sich nach zehn Schritten erneut dieses Muster ergibt.
Dahingegen beschreibt partielle Wiederkehr eine sich wiederholende Sequenz von Zahlen auf dem Band in einem Block.

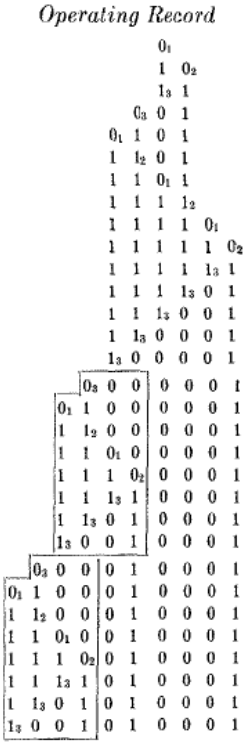

FIG. 5. Operating record of the Turing machine whose serial number is 73136623 (octal) showing the partial recurrence with right barrier

Abbildung 6: Konzept der partiellen Wiederkehr [s. 19, S. 10]

In den eingezeichneten Blöcken lässt sich das Muster gut erkennen. Auf dem Band erscheint erneut dieselbe Sequenz. Kopfpositionen und aktuelle Zustände gleichen sich in den Blöcken ebenfalls.

Bei beiden Mustern bleibt eine fixe linke oder rechte Grenze bestehen und die Bandzellen, die besucht werden, erstrecken sich in die entgegengesetzte Richtung.
Die übrigen Maschinen werden erneut für 50 Schritte auf diese Muster hin untersucht, nur 40 Turingmaschinen fielen nicht in das Schema und wurden nicht von dem Computer eliminiert, die *holdouts*.
Die holdouts mussten per Hand untersucht werden, dabei entdeckten Radó und Lin ebenfalls ein Muster für Schleifen. Sie gingen nach dem Prinzip des Backtracking vor und nutzten zudem Flowcharts um festzustellen, dass das Programm in eine Schleife gerät und somit niemals die Folge von Konfigurationen durchlaufen kann, die zum Endzustand führen. Kein Programm machte dabei mehr als 21 Schritte und erreichte den Endzustand. Damit konnte auch $S(3) = 21$ bewiesen werden.

3.3 Fleißiger Biber mit vier Zuständen

Nachdem Fleißige Biber für zwei und drei Zustände bekannt waren, tauchten weitere Ergebnisse auf. 1966 stellt Allen H. Brady die Vermutungen $\Sigma(4) = 13$ und $S(4) = 107$ auf. Es dauerte einige Jahre bis der vollständige Beweis zu seinen Vermutungen veröffentlicht wurde. Zwischenzeitlich werden 1973 ausführliche Untersuchungen veröffentlicht, die unter anderem einen Beweis für $\Sigma(4) = 13$ liefern, dazu am Ende des Kapitels mehr.
Die folgenden Ergebnisse beziehen sich auf das Paper *The Determination of the Value of Rado's Noncomputable Function* $\Sigma(k)$ *for Four-State Turing Machines* [2] von Allen H. Brady aus dem Jahr 1983.
Bradys Ansatz besteht ebenfalls daraus, die $2,56 * 10^{10}$ möglichen Maschinen zu filtern, um gewissermaßen die „faulen" Biber zu eliminieren. Dabei sind interessante Beobachtungen entstanden. So gelang es ihm, nicht-haltende Turingmaschinen weiter zu klassifizieren. In seinem Paper beschreibt er *Xmas Tress* und *Counter*. Doch zunächst sind im Folgenden die zwei Turingmaschinen beschrieben, welche den Wettbewerb für vier Zustände gewannen und Bradys Vermutungen bestätigten:

$M_4 := (\{z_0, z_1, z_2, z_3, z_e\}, \{0, 1\}, \{0, 1\}, \delta, z_0, 0, \{z_e\})$, wobei

$$
\begin{aligned}
\delta : \quad & z_0\ 0 \to z_1\ 1\ R, \\
& z_0\ 1 \to z_1\ 1\ L, \\
& z_1\ 0 \to z_0\ 1\ L, \\
& z_1\ 1 \to z_2\ 0\ L, \\
& z_2\ 0 \to z_e\ 1\ R, \\
& z_2\ 1 \to z_3\ 1\ L, \\
& z_3\ 0 \to z_3\ 1\ R, \\
& z_3\ 1 \to z_0\ 0\ R.
\end{aligned}
$$

Das zugehörige Modell sieht wie folgt aus:

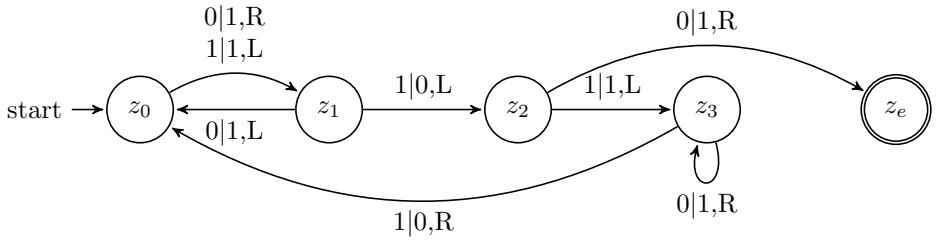

Abbildung 7: Modell für Fleißigen Biber M_4 mit vier Zuständen

Der Fleißige Biber von Brady produziert $\Sigma(4) = 13$ Einsen und benötigt dafür 107 Konfigurationsübergänge. Damit ist M_4 auch der Gewinner für die meisten Schritte mit $S(4) = 107$.

Die folgende Maschine schreibt ebenfalls $\Sigma(4) = 13$ Einsen, jedoch in 96 Schritten:

$M_{4'} := (\{z_0, z_1, z_2, z_3, z_e\}, \{0, 1\}, \{0, 1\}, \delta, z_0, 0, \{z_e\})$, wobei

$$
\begin{aligned}
\delta : \quad & z_0\ 0 \to z_1\ 1\ R, \\
& z_0\ 1 \to z_2\ 0\ R, \\
& z_1\ 0 \to z_0\ 1\ L, \\
& z_1\ 1 \to z_0\ 1\ R, \\
& z_2\ 0 \to z_e\ 1\ R, \\
& z_2\ 1 \to z_3\ 1\ R, \\
& z_3\ 0 \to z_3\ 1\ L, \\
& z_3\ 1 \to z_1\ 0\ L.
\end{aligned}
$$

Das zugehörige Modell sieht wie folgt aus:

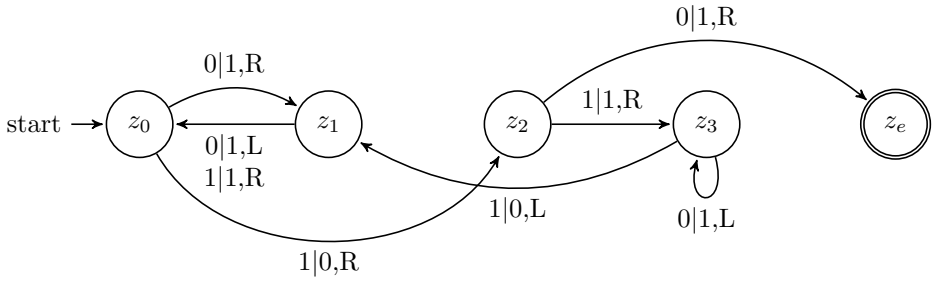

Abbildung 8: Modell für Fleißigen Biber $M_{4'}$ mit vier Zuständen

Diese beiden gegebenen Maschinen für vier Zustände funktionieren fast äquivalent. Wenn für $M_{4'}$ der Startzustand z_1 gewählt wird, verhält sich $M_{4'}$ sehr ähnlich zu M_4.

Brady begrenzt zunächst durch logische Ansätze von Radó und Lin die zu prüfende Anzahl. Mit Hilfe von *tree generation*, auch als Backtracking bekannt, minimiert er die Zahl auf ungefähr 550.000, wobei 5.820 holdouts verbleiben. Diese halten auch nach 500 Simulationsschritten nicht. Brady führte sein Programm gleichzeitig für Maschinen mit drei Zuständen aus und konnte die Ergebnisse auf diese mit vier Zustände übertragen. Dabei entdeckte er die bereits genannten Muster Xmas Tree und Counter, welche ein repetitives Schema aufweisen. Nachfolgend sind diese beiden speziellen Maschinen für jeweils drei Zustände aufgeführt:

$M_{xmas} := (\{z_0, z_1, z_2, z_e\}, \{0, 1\}, \{0, 1\}, \delta, z_0, 0, \{z_e\})$, wobei

$$
\begin{aligned}
\delta : \quad & z_0\ 0 \to z_1\ 1\ R, \\
& z_0\ 1 \to z_2\ 0\ L, \\
& z_1\ 0 \to z_1\ 1\ L, \\
& z_1\ 1 \to z_0\ 1\ R, \\
& z_2\ 0 \to z_e\ 1\ R, \\
& z_2\ 1 \to z_0\ 1\ L.
\end{aligned}
$$

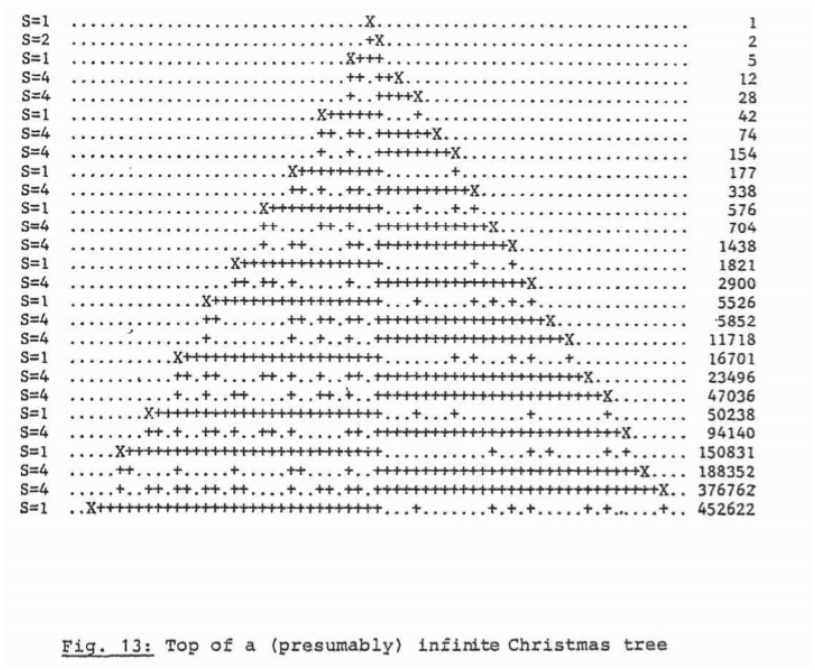

Fig. 13: Top of a (presumably) infinite Christmas tree

Abbildung 9: Arbeitsweise eines Xmas Trees [s. 21, S. 29]

In Abbildung 9 ist ein schematischer Xmas Tree zu sehen. Für eine Null wurde „.“ verwendet und für eine Eins ein „+“. Das „X“ in jeder Zeile stellt die Kopfposition dar, wobei der Kopf auf dem Zeichen rechts vom „X“ steht. Auf der linken Seite sind die Zustände gegeben, in der sich die Maschine zu diesem Zeitpunkt befindet, auf der rechten Seite die Anzahl der Schritte, die gemacht wurden.
Der Lese-/Schreibkopf erweitert die linke und rechte Grenze der besuchten Zellen des Arbeitsbandes immer weiter und erzeugt damit eine repetitive Struktur. Insgesamt ist die Arbeitsweise komplexer als ein simpler Zyklus, nach näherer Betrachtung jedoch trotzdem sich wiederholend, wie Abbildung 9 zeigt. Damit stoppt ein Xmas Tree nicht.

$M_{counter} := (\{z_0, z_1, z_2, z_e\}, \{0, 1\}, \{0, 1\}, \delta, z_0, 0, \{z_e\})$[3], wobei

$$
\begin{aligned}
\delta: \quad & z_0\ 0 \to z_1\ 1\ R, \\
& z_0\ 1 \to z_2\ 1\ L, \\
& z_1\ 0 \to z_0\ 0\ L, \\
& z_1\ 1 \to z_1\ 0\ R, \\
& z_2\ 0 \to z_0\ 1\ L, \\
& z_2\ 1 \to z_e\ 1\ R.
\end{aligned}
$$

Counter zählen Binärzahlen hoch, indem eine Null als 00 kodiert angesehen wird und eine Eins als 10. Die Einsen werden von rechts angefügt und werden während des Übertrags zwischenzeitlich als 11 dargestellt. Die folgende Grafik dokumentiert die Konfigurationen in 60 Schritten, jeweils ein Teil des Bandes wurde abgedruckt. Der „*“ stellt die Kopfposition dar und die Zahl daneben den aktuellen Zustand.

```
0000000000000  1        0000001000000  21       0000100011100  41
      * 1                      * 1                    * 2
0000000000100  2        0000001000100  22       0000100001100  42
      * 2                      * 2                    * 2
0000000000100  3        0000001000100  23       0000100000100  43
      * 1                      * 1                    * 2
0000000000100  4        0000001000100  24       0000100000000  44
      * 3                      * 3                    * 2
0000000001100  5        0000001001100  25       0000100000000  45
      * 1                      * 1                    * 1
0000000011100  6        0000001011100  26       0000100000100  46
      * 2                      * 2                    * 2
0000000010100  7        0000001010100  27       0000100000100  47
      * 2                      * 2                    * 1
0000000010000  8        0000001010000  28       0000100000100  48
      * 2                      * 2                    * 3
0000000010000  9        0000001010000  29       0000100001100  49
      * 1                      * 1                    * 1
0000000010100  10       0000001010100  30       0000100011100  50
      * 2                      * 2                    * 2
0000000010100  11       0000001010100  31       0000100010100  51
      * 1                      * 1                    * 2
0000000010100  12       0000001010100  32       0000100010000  52
      * 3                      * 3                    * 2
0000000011100  13       0000001011100  33       0000100010000  53
      * 1                      * 1                    * 1
0000000011100  14       0000001011100  34       0000100010100  54
      * 3                      * 3                    * 2
0000000111100  15       0000001111100  35       0000100010100  55
      * 1                      * 1                    * 1
0000001111100  16       0000001111100  36       0000100010100  56
      * 2                      * 3                    * 3
0000001011100  17       0000011111100  37       0000100011100  57
      * 2                      * 1                    * 1
0000001001100  18       0000111111100  38       0000100011100  58
      * 2                      * 2                    * 3
0000001000100  19       0000101111100  39       0000100111100  59
      * 2                      * 2                    * 1
0000001000000  20       0000100111100  40       0000101111100  60
      * 2                      * 2                    * 2
```

FIGURE 2. *Behavior of failure number two (Counter)*

Abbildung 10: Arbeitsweise eines Counters [s. 2, S. 5]

In dieser Grafik ist zu erkennen, dass in Schritt 1 eine 0 auf dem Band steht, in Schritt 8 eine 2 (kodiert als 1000), in Schritt 9 eine 3 (kodiert als 1010) und schließlich in Schritt 20 eine 4 (kodiert als 100000). Ein Counter zählt potenziell unendlich und hält auch damit nicht.

[3] Ein Übergang in der δ-Funktion ist fehlerhaft in [2]. Hier wurde $z_0\ 1 \to z_2\ 0\ L$ in $z_0\ 1 \to z_2\ 1\ L$ geändert, damit das Muster der Abbildung 10 entsteht.

Nachdem nun diese beiden besonderen Muster entdeckt und die zugehörigen Turingmaschinen ausgesondert wurden, bleiben noch 218 Maschinen zur Überprüfung aus. Einige Maschinen wurden fälschlicherweise nicht von Bradys Programmen identifiziert, da erst nach vielen Schritten eine Wiederholung gezeigt werden konnte. Außerdem konnten andere Maschinen als Variation der Counter ausgemacht werden und haben Zahlen zur Basis drei oder vier repräsentiert. Damit gab es keine anderen Maschinen, die mehr Einsen geschrieben haben oder mehr Übergänge getätigt haben.
In [2] können weitere interessante Klassifikationen der holdouts nachgelesen werden, so wie *Tail-eating dragons*, die in ähnlicher Weise zu Xmas Trees arbeiten, jedoch zusätzlich einen „Schweif" aus Zahlen auf dem Band besitzen und diesen immer wieder verkleinern - sozusagen „aufessen".
Wie zu Beginn dieses Abschnitts erwähnt, erscheinen 1973 *Untersuchungen über haltende Programme für Turing-Maschinen mit 2 Zeichen und bis zu 5 Befehlen* von Weimann, Casper und Fenzl [8]. Dieses Paper beschreibt ausführlich einen unabhängigen Beweis zu Bradys Vermutungen. Hier werden die Bildung von Äquivalenzklassen und verschiedene Algorithmen dokumentiert.
Weiterführende Literatur zu Fleißigen Bibern mit vier Zuständen findet sich im Paper *Complex behavior on simple machines* von Rona Machlin und Quentin Stout [16]. Hierin wurde 1990 der Beweis von Brady nachvollzogen. Das Paper charakterisiert Turingmaschinen mit bis zu vier Zuständen.

3.4 Fleißiger Biber mit fünf Zuständen

Bis zu diesem Abschnitt konnten alle Fleißigen Biber eindeutig bestimmt werden, für mehr als vier Zustände sind allerdings bislang nur untere Grenzen bekannt.
Der folgende Fleißige Biber ist aus [21] entnommen. Er wird von Uwe Schult im Jahr 1983 beschrieben und schreibt 501 Einsen in 134.467 Schritten. Im eben erwähnten Paper wird er auch *Castor Schultis* genannt, angelehnt an die Gattung der Biber. Dieser Biber stellt eine erste untere Grenze für einen Fleißigen Biber mit fünf Zuständen dar.
$M_5 := (\{z_0, z_1, z_2, z_3, z_4, z_e\}, \{0, 1\}, \{0, 1\}, \delta, z_0, 0, \{z_e\})$, wobei

$$
\begin{aligned}
\delta : \quad & z_0\ 0 \to z_1\ 1\ R, \\
& z_0\ 1 \to z_2\ 0\ L, \\
& z_1\ 0 \to z_2\ 1\ R, \\
& z_1\ 1 \to z_3\ 1\ R, \\
& z_2\ 0 \to z_0\ 1\ L, \\
& z_2\ 1 \to z_1\ 0\ R, \\
& z_3\ 0 \to z_4\ 0\ R, \\
& z_3\ 1 \to z_e\ 1\ R, \\
& z_4\ 0 \to z_2\ 1\ L, \\
& z_4\ 1 \to z_0\ 1\ R.
\end{aligned}
$$

Das zugehörige Modell sieht wie folgt aus:

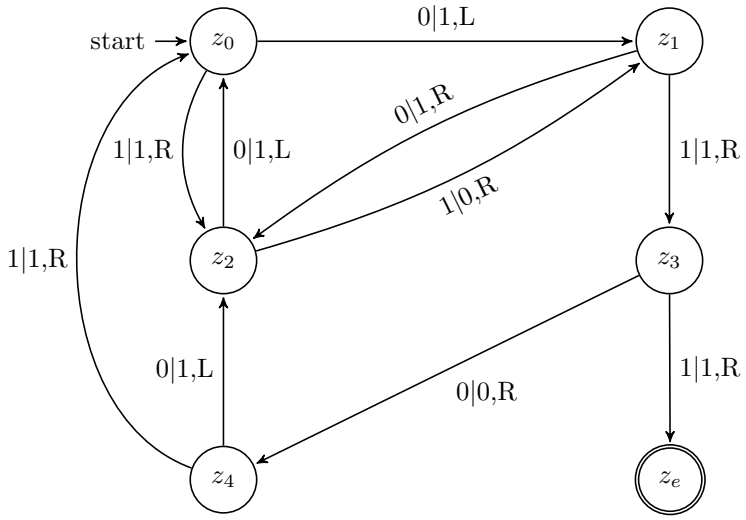

Abbildung 11: Modell für Fleißigen Biber M_5 mit fünf Zuständen

Um diesen Fleißigen Biber zu finden, bedient man sich der gleichen Herangehensweise wie zuvor. Die möglichen Übergänge der δ-Funktion werden aufgezählt, wobei Start- und Endübergang äquivalent zu den anderen Fleißigen Bibern festgelegt werden. Zunächst hat Schult eine Art Suchmechanismus in Baumstruktur entwickelt, um die δ-Funktion sinnvoll zu komplettieren. Die Knoten des Baumes bezeichnen eben diese Übergänge, welche noch nicht festgelegt sind. Der Baum verzweigt sich, falls es dafür mehrere Möglichkeiten gibt.

Dabei können Zweige ignoriert werden, welche zu einem Kreislauf führen würden. Die Identifikation der holdouts gelingt mit Hilfe von Algorithmen und einer Festlegung für eine untere Grenze nach einer gewissen Anzahl von Schritten für besuchte Bandzellen. Diese untere Grenze besagt, dass eine zu prüfende Turingmaschine mehr als diese Anzahl an Zellen beschrieben haben muss, um weiterhin berücksichtigt zu werden.

Im Jahr 1990 fanden Heiner Marxen und Jürgen Buntrock einen noch fleißigeren Fleißigen Biber durch Ausführung von zwei Computerprogrammen. Dieser schreibt 4.098 Einsen und hält nach 47.176.870 Schritten. Damit ist diese Maschine auch der bisherige Gewinner für die Schrittfunktion S. Das Vorgehen und diese Turingmaschine sind in [7] zu finden.

$M_{5'} := (\{z_0, z_1, z_2, z_3, z_4, z_e\}, \{0, 1\}, \{0, 1\}, \delta, z_0, 0, \{z_e\})$, wobei

$$
\begin{aligned}
\delta: \quad & z_0\ 0 \to z_1\ 1\ L,\\
& z_0\ 1 \to z_2\ 1\ R,\\
& z_1\ 0 \to z_2\ 1\ L,\\
& z_1\ 1 \to z_1\ 1\ L,\\
& z_2\ 0 \to z_3\ 1\ L,\\
& z_2\ 1 \to z_4\ 0\ R,\\
& z_3\ 0 \to z_0\ 1\ R,\\
& z_3\ 1 \to z_3\ 1\ R,\\
& z_4\ 0 \to z_e\ 1\ L,\\
& z_4\ 1 \to z_0\ 0\ R.
\end{aligned}
$$

Das zugehörige Modell sieht wie folgt aus:

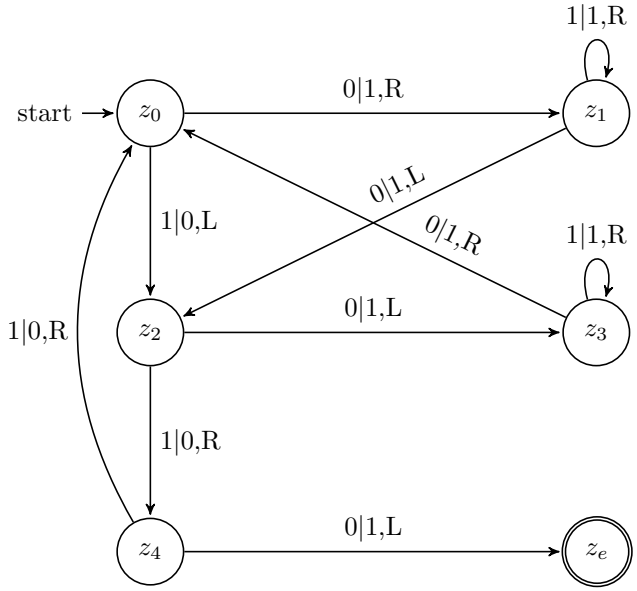

Abbildung 12: Modell für Fleißigen Biber $M_{5'}$ mit fünf Zuständen

Die Computerprogramme, welche in C und ML geschrieben sind, identifizieren zwei Maschinen, welche jeweils 4.098 Einsen produzieren und lassen dabei circa 26 Millionen ungeklärte Turingmaschinen zurück. Bislang ist $M_{5'}$ der aktuell fleißigste Fleißige Biber für fünf Zustände, da viele holdouts einer manuellen oder Computerprüfung widerstehen.

3.5 Schranken für weitere Fleißige Biber

Nachdem nun die Ergebnisse der Forschung auf diesem Gebiet zusammengefasst sind, bleibt die Frage wie sich Fleißige Biber mit mehr als fünf Zuständen verhalten. Für sechs Zustände wurden im Laufe der Geschichte einige untere Grenzen bestimmt. Die ersten Ereignisse in der Hinsicht starten ein Jahr nach Veröffentlichung des Busy Beaver Contests 1963. Die aktuell bekannte untere Grenze für sechs Zustände stammt aus 2010, bestimmt von Pavel Kropitz. Für sieben Zustände wurde bislang nur ein Resultat 2014 gezeigt.

Im Jahr 1964 dokumentierte Milton W. Green erste Ansätze für untere Grenzen der Fleißigen Biber. In *A Lower Bound on Rado's Sigma Function for Binary Turing Machines* [4] finden sich Schranken für sechs, sieben und acht Zustände. Damals beliefen sich diese auf 35 Einsen für sechs Zustände, 22.961 für sieben Zustände und $3 * (7 * 3^{92} - 1)/2$ für acht Zustände.
Donald Lynn konnte den Wert für $\Sigma(6)$ im Jahr 1972 auf 42 aktualisieren [10].
Ludewig, Schult und Wankmüller übertrugen ihre Ergebnisse für fünf Zustände auf sechs, ihre Maschine schreibt $\Sigma(6) = 2.075$ Einsen [s. 21, S. 17].
Heiner Marxen und Jürgen Buntrock demonstrierten in ihrem Paper *Attacking the Busy Beaver 5* ebenfalls eine Turingmaschine für sechs Zustände mit $\Sigma(6) \geq 136.612$ [s. 7, S. 4]. Sie beschäftigten sich in den darauffolgenden Jahren weiterhin mit dem Problem der Fleißigen Biber und erzielten weitere Fortschritte für sechs Zustände. Denn zehn Jahre später, 1982, veröffentlichte Heiner Marxen eine Turingmaschine, welche $\Sigma(6) > 2,5 * 10^{21}$ Einsen auf das Band schreibt [11]. Erneut ein Jahr danach erscheint von ihm eine weitere Maschine, die $\Sigma(6) > 1,2 * 10^{865}$ Einsen produziert [12].
Seit 2004 werden weitere Ergebnisse per E-mail an Heiner Marxen und Pascal Michel übermittelt, welche Michel auf seiner Website auflistet [13]. Die folgenden Grenzen für sechs Zustände wurden von Terry und Shawn Ligocki eingereicht: Im November 2007 fanden sie $\Sigma(6) > 2,5 * 10^{881}$ und überarbeiteten diese Zahl im Dezember auf $\Sigma(6) > 4,6 * 10^{1.439}$. Diese Resultate und eine umfassende historische Studie sind in Pascal Michels Paper *The Busy Beaver Competition: a historical survey* [15] zu finden.
2010 veröffentlicht Pavel Kropitz seine Bachelorarbeit. In dieser Arbeit schilderte er eine Maschine, welche $\Sigma(6) > 3,514 * 10^{18.267}$ Einsen in $7,4 * 10^{36.534}$ Schritten schreibt [s. 9, S. 32] und damit der aktuelle Rekordhalter für sechs Zustände ist.
Für sieben Zustände gibt es bis auf Bradys Ansätze von 1964 nur ein weiteres Resultat, das der momentane Rekordhalter ist: Im Jahr 2014 erschien online ein Beitrag von dem Nutzer „Whytagoras". Dieser erweiterte, zusammen mit Michel und einem anderen Nutzer „Cloudy176", die gefundene Maschine von Kropitz und konnte damit $S(7) > \Sigma(7) > 10^{10^{10^{10^{18.705.353}}}}$ zeigen [23]. Den Beweis verfasste „Cloudy176" in [3].

4 Praxis

4.1 Simulation einer Turingmaschine

Neben der Radó-Funktion, ihrer Unberechenbarkeit und den partiellen Werten dieser, habe ich mich mit einer Simulation für eine Turingmaschine in Java beschäftigt. Mein Programm simuliert die Fleißigen Biber M_1, M_2, M_3, M_4 und $M_{5'}$. Zudem kann eine eigene δ-Funktion für eine Turingmaschine eingegeben und simuliert werden. Außerdem gibt es die Möglichkeit, Bilder der Konfigurationen der jeweiligen Turingmaschine erstellen zu lassen, um das Verhalten dieser besser evaluieren zu können.
Zunächst ist in Abbildung 13 ein Überblick über die Funktionen des Programms zu sehen. Ganz oben im Fenster befindet sich das Arbeitsband, als Liste dargestellt. Links darunter befindet sich ein Textfeld, das die Übergänge der δ-Funktion anzeigt. Rechts davon werden in der Statusbox die Anzahl der Zustände ausgewählt. In der Statusbox sind zudem der aktuelle Zustand und die Kopfposition auf dem Arbeitsband sichtbar; hier wird auch die aktuelle Anzahl der Einsen und Schritte dargestellt.

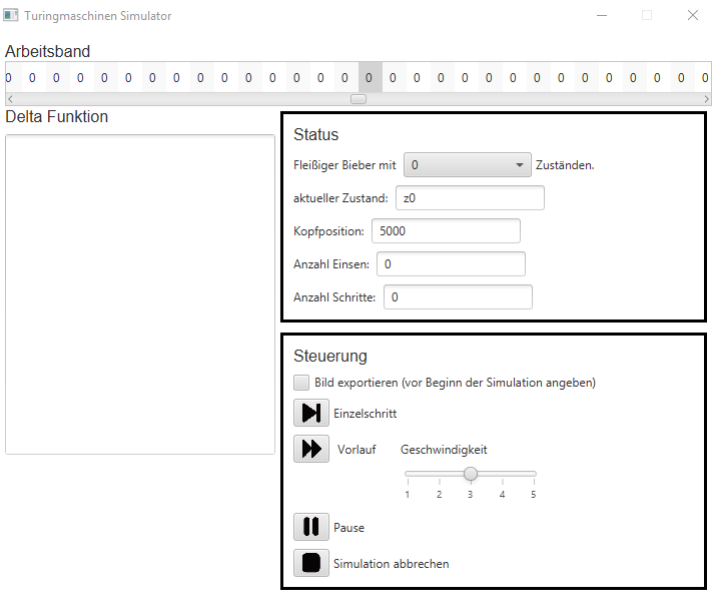

Abbildung 13: Turingmaschinen-Simulator

In der Steuerungsbox darunter wird die Simulation mit den Buttons „Einzelschritt“ oder „Vorlauf“ gestartet, mit dem Pause-Button wird der Vorlauf pausiert und mit dem Stopp-Button wird die komplette Simulation abgebrochen. Eine weitere Simulation kann anschließend neu gestartet werden.

Während des Vorlaufs kann einer höhere oder geringere Geschwindigkeit eingestellt werden, mit der die Maschine simuliert wird. Zudem kann mit der Checkbox (vor Beginn der Simulation) ausgewählt werden, ob ein Bild nach dem Durchlauf generiert werden soll. Dieses Bild stellt Einsen als schwarze Pixel dar und Nullen als weiße. Dabei wird der Inhalt des Arbeitsbandes zeilenweise untereinander generiert. Mit Hilfe des Bildes lässt sich die Arbeitsweise der jeweiligen Turingmaschine besser nachvollziehen.

Zunächst wird über das Dropdown-Menü die gewünschte Anzahl der Zustände eingestellt. Hier kann zwischen Zuständen null bis fünf und eigener Eingabe gewählt werden. Ein Fleißiger Biber mit null Zuständen stellt lediglich die Initialeinstellung des Programms dar.

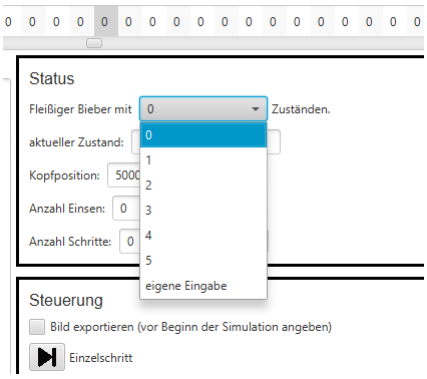

Abbildung 14: Auswahl der Anzahl der Zustände

Nach der Auswahl wird das Programm initialisiert und die entsprechende δ-Funktion erscheint an der linken Seite. Nun kann die Simulation mit den Buttons gestartet werden.

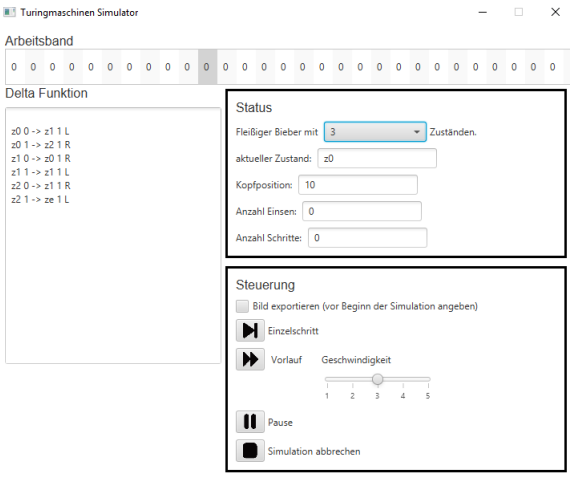

Abbildung 15: Initialisierung des Programms

Nachdem die Simulation erfolgreich anhielt und damit den Endzustand erreicht, taucht eine entsprechende Meldung auf, wie die nachfolgende Abbildung zeigt:

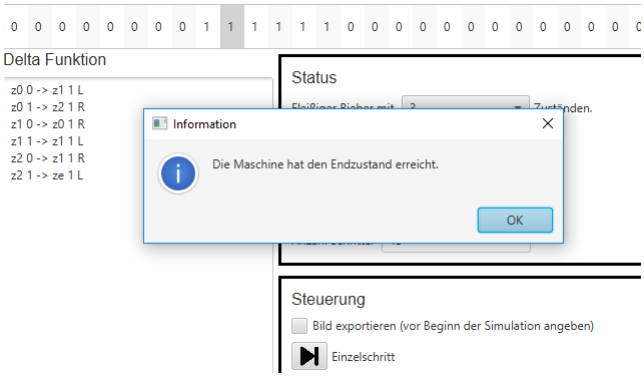

Abbildung 16: Erfolgreiche Beendung des Programms

Wenn nun eine eigene δ-Funktion eingegeben werden soll, wird zunächst über das Dropdown-Menü „eigene Eingabe“ ausgewählt. Das Textfeld für die δ-Übergänge ist nun beschreibbar. Dabei sollte die Konvention eingehalten werden, da kein *error handling* implementiert wurde.

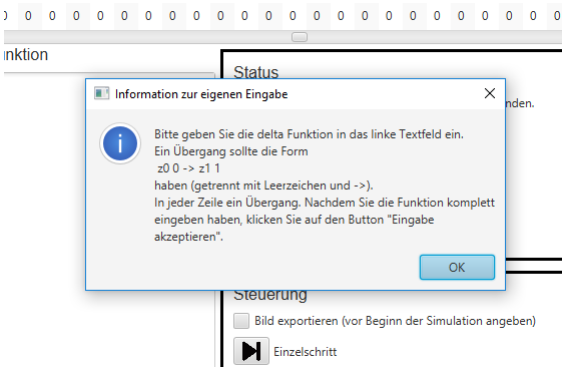

Abbildung 17: Konvention der eigenen Eingabe

Falls die Eingabe verarbeitet werden soll, kann das mit dem Button „Eingabe akzeptieren“ geschehen. Nun kann das Programm wie gewohnt über die Steuerungs-Buttons gestartet werden. Falls die eingegebene δ-Funktion unvollständig ist und das Programm keinen Übergang zu der jeweiligen Konfiguration finden kann, ist die weitere Simulation nicht mehr möglich.

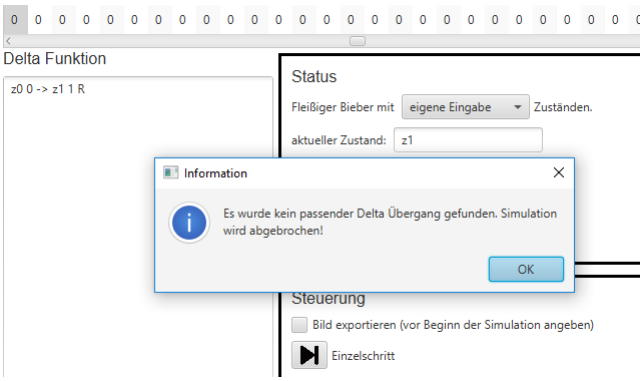

Abbildung 18: Erfolglose Beendung des Programms

Zudem ist das Band für die eigene Eingabe auf 10.000 Zellen begrenzt. Der Kopf befindet sich zu Beginn auf Zelle 5000. Ein potenziell *unendliches* Arbeitsband wäre eine sinnvolle Erweiterung, die noch implementiert werden kann.

Das Simulationsprogramm hat weiterhin die bereits erwähnte Möglichkeit, Bilder der Konfigurationen zu generieren. Bevor eine Simulation gestartet werden soll, muss die Checkbox aktiviert werden.

Nachdem die Simulation beendet wurde (entweder über das Programm selbst oder über den Stopp-Button), befindet sich im selben Ordner des Programms eine Textdatei mit den Konfigurationen aus Einsen und Nullen und ein Bild, das

aus der Textdatei generiert wurde. Dabei stellt jede Zeile des Bildes den Inhalt des Arbeitsbandes dar. Weiße Pixel repräsentieren Nullen, schwarze Pixel Einsen. Die nächste Zeile des Bildes beschreibt den Inhalt nachdem die Maschine einen Schritt gemacht hat.

Nachfolgend befinden sich die Bilder, die mein Programm für die Fleißigen Biber M_1, M_2, M_3 und M_4 erstellt hat:

Abbildung 19: Konfigurationen des Fleißigen Bibers M_1

Abbildung 20: Konfigurationen des Fleißigen Bibers M_2

Abbildung 21: Konfigurationen des Fleißigen Bibers M_3

Abbildung 22: Konfigurationen des Fleißigen Bibers M_4

Im Folgenden ist das generierte Bild für $M_{5'}$, simuliert für die ersten 1.000 Schritte:

Abbildung 23: Konfigurationen des Fleißigen Bibers $M_{5'}$ für die ersten 1.000 Schritte

Diese Abbildung zeigt die schematische Arbeitsweise von $M_{5'}$. Es werden durchgehende Strings aus Einsen erzeugt und danach Teile davon mit Nullen ersetzt. Der durchgehende String wird mit der Zeit länger.
Außerdem habe ich mithilfe der Simulation Bilder für M_{xmas} und $M_{counter}$ aus Abschnitt 3.3 über Fleißige Biber mit vier Zuständen erstellt. Die Maschinen wurden jeweils für etwa 100 Schritte simuliert.

Abbildung 24: Konfigurationen der Maschine M_{xmas} für die ersten 100 Schritte

Auf dieser Abbildung lässt sich das oszillatorische Muster eines Xmas Trees gut erkennen. Das Muster dehnt sich in beide Seiten des Arbeitsbandes aus und erzeugt ebenfalls immer wieder einen durchgängigen String aus Einsen, welcher von links aufgebaut und von rechts abgebaut wird. Dieses Bild hat Ähnlichkeit zu der Abbildung 23 der Arbeitsweise von $M_{5'}$. Die Maschine $M_{5'}$ hat jedoch zwei Zustände mehr und kann damit halten.

Abbildung 25: Konfigurationen der Maschine $M_{counter}$ für die ersten 100 Schritte

Dieses Bild kann als grafische Darstellung der Abbildung 10 für $M_{counter}$ verstanden werden. Die Maschine fügt von rechts (LSB) Einsen an und zählt damit Binärzahlen in der Kodierung aus Abschnitt 3.3 hoch. Zwischenzeitlich steht auf dem Band ein Block aus Einsen, wenn das MSB um ein Bit erhöht werden muss.

5 Fazit

Tibor Radó stieß mit der Definition des Busy Beaver Game 1961 einen Wettbewerb an, der bis heute aktuell ist. Um dieses Spiel zu gewinnen, muss die Hürde der Unberechenbarkeit überwunden werden. Dieses Hindernis fordert Informatiker immer wieder heraus, kreative Ansätze für die Bestimmung der Fleißigen Biber zu finden.

Die Aufgabe scheint einfach: Mithilfe einer Turingmaschine gilt es die meisten Einsen auf das Band zu schreiben und die Maschine danach anzuhalten. Dabei ist es unproblematisch ein Ergebnis zu finden. Dieses Ergebnis als Lösung zu validieren ist der Reiz des Spiels. Weiterhin steigt bei zunehmenden Zuständen die Komplexität der Möglichkeiten, so dass es überdies problematisch wird, ein kompetitives Ergebnis zu finden.

Die Geschichte der Fleißigen Biber zeigt, dass technischer Fortschritt der Computer, algorithmisches Testen vereinfacht. Gerade zu Beginn der Geschichte gelangten Informatiker schnell an die Grenzen von Computerleistungen. Die Entwicklung des Computers als Alltagsgegenstand ermöglicht nun mehr Menschen sich am Fleißigen Biber Contest zu beteiligen, und schnellere Ergebnisse zu erlangen. Damit muss zunächst nicht auf mathematische Beweistechniken zurück gegriffen werden.

Tibor Radó und Shen Lin legten zunächst einen wichtigen Grundstein mit der Validierung des Fleißigen Bibers mit drei Zuständen. Dieser fungierte als Ansatz für weitere Fleißige Biber um vernachlässigbare Turingmaschinen im Vorhinein heraus zu filtern.

Die Entdeckung von besonderen nicht-haltenden Turingmaschinen sind ein erachtenswertes Nebenprodukt jenseits der tatsächlichen Fleißigen Biber-Turingmaschinen, die anhalten. Da die Unberechenbarkeit der Radó-Funktion feststeht, ist die Erkennung von Mustern von nicht-haltenden Turingmaschinen eine gute Perspektive um weitere Fleißige Biber ausfindig zu machen.

Gegenwärtig lassen sich Fleißige Biber mit bis zu vier Zuständen weitesgehend effizient bestimmen. Damit tut sich hier die neue verschobene Grenze der Leistung des Computers auf. Für Fleißige Biber ab fünf Zuständen sind bislang nur untere Grenzen bekannt und diese haben teilweise schwer vorstellbare Werte.

Die aktuelle untere Schranke für sieben Zustände beläuft sich auf

$$> 10^{10^{10^{10^{18.705.353}}}} \text{ Einsen}$$

- eine Zahl, die das menschliche Vorstellungsvermögen übersteigt. Solche Größen machen eine manuelle Prüfung unmöglich. Damit ist der Einsatz von Computern unvermeidlich.

Meine Simulation am Computer ermöglichte mir, bereits bekannte Fleißige Biber innerhalb kürzester Zeit zu untersuchen, was manuell wesentlich länger ge-

braucht hätte. Mit solch einem Programm verringert sich die Fehleranfälligkeit des Testens von Turingmaschinen auf ein bestimmtes Verhalten deutlich. Der weitere praktische Aspekt des Programms, Bilder generieren zu lassen, bietet die Möglichkeit, auch komplexere Maschinen schnell visuell aufzufassen.

Das Simulationsprogramm kann durch eine automatische Suche nach Fleißigen Bibern erweitert werden, indem alle möglichen Konstruktionen der δ-Funktion für ein gegebenes n untersucht werden. Dabei werden Algorithmen zur Erkennung von Schleifen eingebaut, die nicht-haltende Turingmaschinen eliminieren.

Weiterhin können Fleißige Biber auf die mathematische Analysis übertragen werden. Es gibt Untersuchungen über die Zusammenhänge der Funktionen $\Sigma(n)$ und $S(n)$. Diese sind zum Beispiel im Paper *A note on busy beaver and other creatures* [1] zu finden. Das Paper behandelt zudem die Ermittlung des Wachstums der Radó-Funktion und Möglichkeiten, Schranken für diese zu bestimmen. Das Buch *Computability and Logic* [6] beschreibt mathematische Beobachtungen zur Radó-Funktion.

Die Ansätze einer visuellen Darstellung von Turingmaschinen sowie die algorithmischen Elemente der Bestimmung von Fleißigen Bibern als auch die mathematischen Ansätze der Charakterisierung der Radó-Funktion sind bedeutsame Vorgehensweisen um die Unberechenbarkeit der Funktion $\Sigma(n)$ zu umgehen.

Dabei könnten diese Methoden auf andere bislang ungelösten Probleme übertragen werden, wie zum Beispiel das Collatz-Problem. Beim Collatz-Problem ist es unklar, ob jede Zahlenfolge mit einem einfachen Bildungsgesetz in einen bestimmten Zyklus mündet [25]. Unter anderem beschäftigte sich Michel bereits im Paper *Small Turing machines and generalized busy beaver competition* [14] mit einem Zusammenhang zwischen dem Collatz-Problem und dem Fleißigen Biber-Problem.

Der Fleißige Biber Contest breitete sich ferner bereits auf Turingmaschinen mit mehr als zwei Symbolen aus. Wissenschaftler beißen sich auch hieran die Zähne aus, da es hier wesentlich mehr zu testende Turingmaschinen gibt. Die Ergebnisse dazu werden auf Michels Website [13] aktuell gehalten und können ausführlich in seiner historischen Studie [15] nachgelesen werden.

Letzten Endes bleibt es interessant um die Suche nach Fleißigen Bibern. Eindeutig ist, dass Informatiker noch lange an diesem Thema zu „nagen“ haben.

Literatur

Paper

[1] A. M. Ben-Amram und B. A. Julstrom. »A note on Busy Beavers and other creatures«. In: *Mathematical Systems Theory* 29 (1996), S. 375–386.

[2] Allen H. Brady. »The determination of the value of Rado's noncomputable function $\Sigma(k)$ for four-state Turing machines«. In: *Mathematics of Computation* 40 (1983), S. 647–665.

[4] Milton W. Green. »A Lower Bound on Rado's Sigma Function for Binary Turing Machines«. In: *1964 Proceedings of the Fifth Annual Symposium on Switching Circuit Theory and Logical Design* (1964), S. 91–94.

[5] David Hilbert. »Mathematische Probleme«. In: *Nachrichten der Königliche Gesellschaft zur Wissenschaften zu Göttingen, Mathematische-physikalischen Klasse* 3 (1900), S. 253–297.

[7] Heiner Marxen und Jürgen Buntrock. »Attacking the Busy Beaver 5«. In: *Bulletin of the EATCS* 40 (1990), S. 247–251.

[8] B. Weimann und K. Casper und W. Fenzl. »Untersuchungen über haltende Programme für Turing-Maschinen mit 2 Zeichen und bis zu 5 Befehlen«. In: *Gesellschaft für Informatik* 78 (1973), S. 72–81.

[9] Pavel Kropitz. »Busy Beaver Problem«. In: (2010).

[10] Donald S. Lynn. »New Results for Rado's Sigma Function for Binary Turing Machines«. In: (1972).

[14] Pascal Michel. »Small Turing machines and generalized busy beaver competition«. In: (2004).

[15] Pascal Michel. »The Busy Beaver Competition: a historical survey«. In: (2009).

[16] Rona Machlin und Quentin F. Stout. »The complex behavior of simple machines«. In: *Physica D: Nonlinear Phenomena* 42 (1990), S. 85–98.

[17] Tibor Radó. »On a simple source for non-computable functions«. In: *Proceedings of the symposium on mathematical theory of automata* (1962), S. 75–81.

[18] Tibor Radó. »On non-computable functions«. In: (1961).

[19] Tibor Radó und Shen Lin. »Computer Studies of Turing Machine Problems«. In: *Journal of the ACM* 12 (1965), S. 196–212.

[20] Alan Turing. »On computable numbers, with an application to the Entscheidungsproblem«. In: (1936).

[21] Jochen Ludewig und Uwe Schult und Frank Wankmüller. »Chasing the Busy-Beaver-Notes and Observations on a Competition to find the 5-state Busy Beaver«. In: (1983).

[22] Prof. Dr. Heribert Vollmer. »Grundlagen der theoretischen Informatik«. In: (2018).

Bücher

[6] George S. Boolos und John P. Burgees und Richard C. Jeffrey. *Computability and Logic*. 5. Aufl. 2007.

Internet

[3] Cloudy176. *Proving the bound for S(7)*. zuletzt besucht: 27.01.2020. 2014. URL: https://googology.wikia.org/wiki/User_blog:Cloudy176/Proving_the_bound_for_S(7).

[11] Heiner Marxen. *A new 6-state busy beaver champion ($> 10^{21}$ ones)*. zuletzt besucht: 27.01.2020. 2000. URL: https://groups.google.com/forum/#!search/marxen%5C$20buntrock/comp.theory/_m85smf7vkw/VrXGoRvIULgJ.

[12] Heiner Marxen. *A new 6-state busy beaver champion ($> 10^{865}$ ones)*. zuletzt besucht: 27.01.2020. 2001. URL: https://groups.google.com/forum/#!search/marxen%5C$20buntrock/comp.theory/yOIcRb5FNzU/T1Jv_6jyB3UJ.

[13] Pascal Michel. zuletzt besucht: 27.01.2020. 2019. URL: http://www.logique.jussieu.fr/~michel/index.html.

[23] Whytagoras. *A good bound for S(7)?* zuletzt besucht: 27.01.2020. 2014. URL: https://googology.wikia.org/wiki/User_blog:Wythagoras/A_good_bound_for_S(7)%3F.

[24] Wikipedia. *Busy Beaver Game*. zuletzt besucht: 03.12.2019. URL: https://en.wikipedia.org/wiki/Busy_Beaver_game.

[25] Wikipedia. *Collatz-Problem*. zuletzt besucht: 22.02.2020. URL: https://de.wikipedia.org/wiki/Collatz-Problem.

[26] Wikipedia. *Fleißige Biber*. zuletzt besucht: 22.11.2019. URL: https://de.wikipedia.org/wiki/Flei%C3%9Figer_Biber.