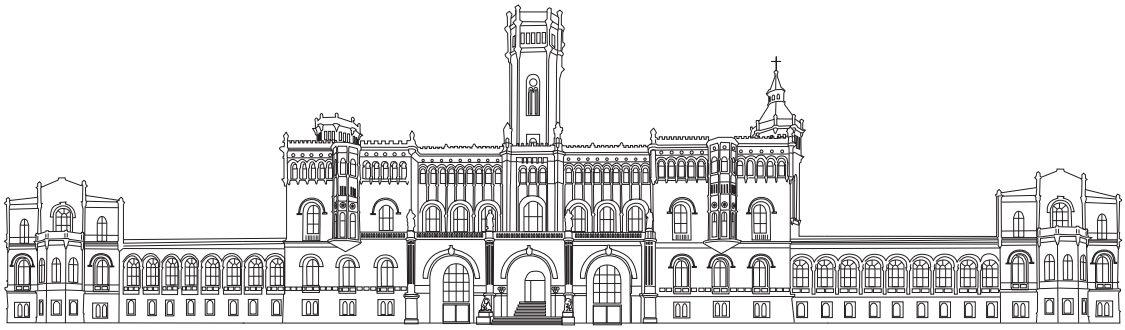

On the progression of finding the smallest known universal Turing machine

Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover
Institut für Theoretische Informatik

Abschlussarbeit

zur Erlangung des akademischen Grades
Bachelor of Science

vorgelegt von

Laura Strieker
10004978

im Februar 2020

Erstprüfer:	Prof. Dr. H. Vollmer
Zweitprüfer:	Dr. A. Meier
Betreuer:	Prof. Dr. H. Vollmer

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 21. Februar 2020

_____

Vorname Nachname

Contents

1 Introduction

The history of universal Turing machines began when Alan Turing came up with the first universal Turing machine in 1937 [15]. The following years remained silent measured by the invention of new universal Turing machines but in the 1950s, Shannon brought the factor size into the field [12]. This lead to a rigorous competition in the 1960s between various people to find the smallest possible universal Turing machine, amongst them Watanabe and Minsky, a universal Turing machine of the latter was the smallest known for several years [8].
A few more little improvements in size were made through out the years but it was not until 2002 that another big dent was made in terms of size. Wolfram [19] found two very small universal Turing machines which are still the smallest known universal Turing machines today.
The reason smaller and smaller universal Turing machines were found is the creativity and ingenuity of the researchers. Every new big idea and every smart new approach could possibly make the next breakthrough in the size of universal Turing machines; striving further towards a minimum.
This work will examine universal Turing machines with regards to their size and the methods that were invented over the course of time to reduce this size. How small can universal Turing machines possibly be? How complex can these small structures get?

2 Fundamentals

Before examining universal Turing machines, it is inevitable to discuss what Turing machines are in general. The first Turing machine was introduced by Alan Turing in 1937 [15] as a model of computation. He designed these computation machines that were later named after him, such that they consist of an alphabet of symbols, a set of states, an infinitely long working tape divided into cells and a head that operates on this tape [15]. The input of the machine is placed on the tape. Each cell of the tape bears a symbol of the alphabet where one certain symbol is identified with a blank cell, therefore being called the blank symbol. A $\square$ is often used to depict this symbol. The head moves on the tape by shifting one cell to the left or the right. The head is able to read the symbol on the single cell that is currently under the head and write a new symbol on that cell.
The machine cannot remember the symbols it has read on the tape directly but the states can be used to serve as a memory. A pair of the current state and the symbol currently read on the tape is called a state-symbol pair or a configuration. Each configuration determines a distinct step of computation of a Turing machine. A step consists of changing the state, writing a new symbol on the current cell of the tape and moving the head on the tape; this is called a transition. Note that the new state may be the same as the current state, as well as the new symbol may be the same as the current symbol. The head is not allowed to move further than one cell to the left or the right but it might stay at the same cell. For every configuration that may be encountered there is such a transition to a new state, symbol and cell. The entirety of these transitions make up the transition function.
Turing also defined two groups of symbols: one kind he called figures consisting only of 0 and 1 and another kind that is not defined specifically. In more recent definitions this is expressed by two different alphabets: an input alphabet, often but not always consisting solely of 0 and 1, and a working alphabet, consisting of all symbols that are used on the tape, including the input alphabet and the blank symbol. Turing's rather limited definition is due to the fact that he envisioned Turing machines to be pure computation machines that work on binary sequences.
Formally, a Turing machine may be denoted as a 7-tupel $(Q, \Sigma, \Gamma, \delta, \square, q_0, E)$ where

- Q is the set of states $\{q_0, ..., q_m\}$
- $\Sigma \subseteq \Gamma$ is the set of input symbols
- Γ is the set of tape symbols, including the blank symbol
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, N, R\}$ is the partial transition function with $\{L, N, R\}$ being the possible head movements shift left, no movement and shift right.
- $\square$ is the blank symbol
- q_0 is the initial state
- $E \subseteq Q$ is the set of accepting or final states

A Turing machine operates on the input given on its tape such that it reads the first symbol in the initial state, looks up the corresponding transition, executes it and then processes the new configuration it is in. This operation cycle continues to be repeated until the Turing machine halts. A Turing machine halts and accepts, whenever it reaches a final state and does not move out of it again by some transition. If the transition function is not defined for a state-symbol pair, the Turing machine is either envisioned to be in an infinite loop or halt but not accept unless being in a final state.
In addition to Turing's original idea of a computation machine, Turing machines are able to recognize semi-decidable languages. A Turing machine recognizes a language, if for every word of the language, the Turing machine halts in a final state and for every word that does not belong to the language, it never accepts. Turing's definition did not include the possibility of accepting, as the halting of the machine would only occur if the computation of the binary sequence was finished.

2.1 Universal Turing machines

The first universal Turing machine was also found by Alan Turing in the same paper in which he invented the Turing machine in [15]. It is important to understand what universality in Turing machines means and what the size of such a machine is.
A system is called universal or capable of universal computation if it can execute any algorithm when given the data for the algorithm in some encoded form readable by the system [3].
A Turing machine is said to be universal if it can simulate every other possible Turing machine, given the Turing machine that it should simulate as some kind of input. This means it can universally do every computation that can be done by Turing machines. As Turing machines compute partial recursive functions due to their partial transition functions, an universal Turing machine is able to compute every partial recursive function. The Turing machines that make up the input for the universal Turing machine, need to be encoded such that the universal Turing machine can reconstruct them unambigiously. Therefore, some kind of a Goedel number is used to encode these Turing machines. A Goedel number is a number that unambigiously identifies a word with a certain procedure. It will later be clear that the choice of encoding for the input has a huge impact on the size of a UTM.
As mentioned previously, the size of universal Turing machines and how this size was achieved is the focus of this work. The size of an universal Turing machine is defined as the product of the number of states and the number of symbols as suggested by Shannon in [12]. The number of symbols refers to the size of Γ, i.e. the tape symbols, since these are the symbols that are used in total. Some definitions of Turing machines do not even include the input alphabet Σ. In the following Σ may be omitted.

2.2 Notation

The notation of Turing machines differs through time and author. To be clear, the way of notation in this work will be explained briefly. Furthermore, the abbreviations TM for Turing machine and UTM for universal Turing machine will be used in the following chapters.

2.2.1 Symbols, States and Transitions

The notation of a complete Turing machine and the way these machines work will be explained using an example. Let M_1 be the TM that decides whether a binary number is odd, i.e. if its value is equivalent to 1 mod 2. So M_1 checks for every input sequence over the alphabet $\{0,1\}$ if it is an odd number. A binary number is odd if the last digit is 1. $M_1 = (\{q_0, q_R, q_1, q_{acc}\}, \{0,1\}, \{0,1,\square\}, \delta, \square, q_0, \{q_{acc}\})$ where δ is defined as:

$$
\begin{aligned}
&q_0\square \to q_0\,\square\,N && &&q_00 \to q_R0R\\
&q_01 \to q_R1R && &&q_R0 \to q_R0R\\
&q_R1 \to q_R1R && &&q_R\square \to q_1\,\square\,L\\
&q_10 \to q_10N && &&q_11 \to q_{acc}1N
\end{aligned}
$$

Note that M_1 is constructed under the assumption that the head is placed on the first symbol of the input sequence.
The machine checks if the input is empty, if it is not, it moves the head to the end of the input and checks if the last symbol is a 1.
Note that the transition function alone is able to describe the whole Turing machine, as every state, symbol and transition is mentioned.

2.2.2 Transition tables

To describe Turing machines briefly, transition tables will be used. Let M_1 be the TM as stated above. The δ-function of M_1 can also be depicted as a transition table as in Table 2.1.

	q_0	q_R	q_1
0	q_R0R	$0R$	q_10N
1	q_R1R	$1R$	$HALT$
$\square$	$\square N$	$q_1\,\square\,L$	-

Table 2.1: Transition table of M_1

A transition table is read by choosing the current state from the top row and the current symbol from the left column. The following state, symbol and head movement is stated in the corresponding cell of the table. If the state does not change, it is not written out. Note that the transition into the accepting state was changed to the instruction $HALT$, as there is no further computation done in state q_{acc}. The dash (-) in the bottom right cell denotes that there is no transition for this state-symbol pair.
If the specific transitions of a Turing machine are discussed, a transition table will be used to express these, as it is very compact.

3 The first universal Turing machine

Alan Turing was the one to have the idea of a universal Turing machine as well as the Turing machine later named after him. In 1937, he published a paper called "On computable numbers, with an application to the Entscheidungsproblem" [15] stating his universal Turing machine. This chapter is dedicated to understanding the first definition of a Turing machine as well as explaining which underlying mechanisms Turing used to construct his universal Turing machine.

3.1 The first definition of Turing machines

Turing's definition differs slightly from the one that is stated above. These differences are found in the definition of the machine itself as well as the type of calculation it carries out. Turing understood TMs as computation machines which printed a sequence of 0's and 1's (he calls these symbols figures) on a formerly empty tape. Turing machines are not yet viewed as a tool to recognize or decide languages.
Another difference is the use of the tape. In this first definition, alternating squares are used. Every first, or left as you will, cell is called an F-square, bearing a figure of the calculated sequence. The other, right cells are called E-squares. These cells do not contain any figures belonging to the sequence the machine computes, but rather "mark" the F-squares to the left with some kind of symbol. This alternating outline of the tape is not necessary, since it can be made up for with a bigger variety of symbols. If a machine following Turing's definition had an alphabet consisting of, say, the figures 0 and 1 for printing on F-squares and the symbols x, y and z for marking on the E-squares, making up an alphabet of size 5, the markers could easily be substituted with the figures, say $0_x, 0_y, 0_z, 1_x, 1_y$ and 1_z. This would lead to an alphabet size of 8 including the unmarked figures 0 and 1 but every cell could be used equally.
Besides these structural differences, there are a few definitions that are needed to understand the method of work of Turings universal TM. A *configuration* is a pair of a state and a symbol that defines the next action of a Turing machine, namely the symbol printed, the direction of the head movement and the state entered next. *Configurations* are not dependent on any other cells of the tape but the one currently read by the machine. *Complete configurations* on the other hand include the current state, the current cell read by the machine (not necessarily the input but the number of the cell) and the whole sequence on the F-squares of the tape. Please note that no blanks occur in the sequence of the F-squares. The difference between a *configuration* and a *complete configuration* is important to understand the universal Turing machine correctly.

3.2 Standard Description

Turing came up with his own encoding of Turing machines that is explicit as well as reversable, similar to a Goedel number. Given such an encoding of a Turing machine,

his UTM is able to compute the same sequence as the encoded machine. He called this encoding the *Standard Description* or short *S.D.* of a machine.
A *S.D.* is entirely made up of the symbols $A, C, D, L, N, R, ;$ where ; acts as a seperating character. Every transition of a Turing machine is coded on its own, the complete *S.D.* is then made up by chaining all encoded transitions after each other, seperating them with semi-colons. Note that Turing ordered transitions in a different way, the final state comes last in a transition, e.g. Turing used the order $q_i S_j \rightarrow S_k X q_l$ instead of $q_i S_j \rightarrow q_l S_k X$ where X denotes the head movement. The definite method to construct the *S.D.* of a Turing machine is as follows:
Let $\{q_1, ..., q_R\}$ be the set of states where q_1 is the initial state and $S_0, S_1, S_2, ..., S_m$, be the alphabet of the machine, where $S_0 = \square$, $S_1 = 0$ and $S_2 = 1$.
Then for every transition

$$q_i S_j \rightarrow S_k X q_l$$

write down

$$DA^i DC^j DC^k X DA^l;$$

where $X \in \{L, N, R\}$. When appending one encoded transition to another, the *S.D.* is retrieved. Note that the last semi-colon is not necessary, since there is nothing to seperate.
This *Standard Description* can be encoded furthermore, such that it only consists of numbers. Turing calls this the *Description Number* of a machine which is basically a Goedel number for Turing machines. The essential idea is to assign a distinct integer to every symbol of the *S.D.* starting by 1 and use the assigned integers instead of the symbol, creating a sequence solely consisting of numbers.

3.3 Turing's UTM

Turing's UTM now uses this *Standard Description* rather than the *Description Number* to simulate another Turing machine. For the remainder of the chapter M will be the Turing machine which is to be simulated and U will denote the universal Turing machine that simulates M.

3.3.1 Rough outline

Let M' be a Turing machine that writes all successive *complete configurations* on the F-squares of its tape. Suppose that M' knows how M works by knowing the *S.D.* of M internally. M' writes down the *complete configurations* as follows:
For every *complete configuration*, the complete sequence on the F-squares of the tape is written out. The current state is written in this sequence, marking the current cell at the same time by putting it right in front of it. If the sequence of the F-squares is, say, 0010 and the current cell is the one containing 1 and the current state is q_5, then the *complete configuration* for this scenario would be $00q_51$. All consecutive *complete configurations* are written down after each other, each seperated by a colon.
The *complete configurations* are further encoded in a similar way to the *S.D.*:
Every symbol S_j of the alphabet is encoded as DA^j and every state q_i of the set of states is encoded as DC^i. For the above example this would yield $DADADCCCCCDAA$, since 0,1 are encoded as S_1, S_2, as stated in Chapter 3.2. Again, these successive *complete configurations* are seperated by colons.

Every TM has such a corresponding sequence of *complete configurations*, so M has as well. If M exists, it is easy to construct a machine M' that prints out this sequence on the tape based on the *Standard Description* of M, as it only needs to append the same *complete configuration* with the change of one transiton, i.e. the appendance of one symbol or the change of one symbol in the sequence and the movement of the head one cell to the left or right and the change of the state. All this information can be gathered by looking at the *S.D.* Turing's definition of TMs calls for figures to be written on the F-squares, since these make up the computed sequence. Therefore, M' has to print figures. This is done by printing the figures that are new in a *complete configuration* between the *complete configuration* they first appear and the *complete configuration* before again seperated by colons. Going back to the example from before; if the next *complete configuration* was $001q_61$, encoded as $DADADAADCCCCCCDAA$, these two succesive *complete configurations* would yield

$$DADADCCCCCDAA : DADADAADCCCCCCDAA$$

and M' would then write

$$DADADCCCCCDAA : 1 : DADADAADCCCCCCDAA$$

because a 1 was printed by M between these two successive *complete configurations*. If this is done for all *complete configurations*, the sequence of figures printed by M' would then represent the sequence of figures printed by M and therefore M' would simulate M.

3.3.2 Detailed description

The universal Turing machine U is implementing the outline given above. The tape of U is prepared with an ə on an F-square and another e on the next E-square. Then follows the *S.D.* of M on F-squares; encoded as explained as before. After the *S.D.*, a double colon :: is written on the next F-square as one single symbol.

Size of the UTM

The alphabet of the UTM U is $\{A, C, D, L, N, R, 0, 1, u, v, w, x, y, z\}$ reaching a size of 14. Unfortunately, Turing uses abbreviations for the transitions which he calls *m-functions*, making it hard to depict the number of states clearly. But given the fact that he uses more than three pages throughout the aforementioned paper [15], it is clearly a lot - he already uses over 50 *m-functions*. Every *m-function* stands for some series of transitions with at least one but probably a few states involved. That said, Turings UTM bears the idea of simulating other Turing machines but is far off from other results regarding the size of the UTM, as can be seen in Table 7.1.

Operation cycle

Following the idea stated above, the UTM will print out all consecutive *complete configurations* and the figures that are new in the last *complete configuration* between them. Let one operation cycle be the printing of one *complete configuration* and the next new figure

if there is one, as M might print a blank.
The UTM begins with the first *complete configuration* that can be hard coded because it is always q_1 and an empty tape. As all *complete configurations*, it prints it after the *S.D.* on the tape, beginning on the F-square after the double colon. q_1 is per definition the initial state of M and before the first transition is carried out, the tape will be blank.
The general operation cycle consists of

- printing the *complete configuration* on the tape right after the last one and
- marking the current *configuration*, meaning the current state and the current symbol under the head of the TM, as explained in Chapter 3.1.
- Find the corresponding transition of the marked *configuration* in the *S.D.* of M by comparing the *configurations* and
- mark the operation to be carried out, the next state and the head movement.
- If the marked operation involves the printing of 0 or 1; append ”0:” or ”1:” on the next F-squares after the current *complete configuration*
- print the next *complete configuration* on the tape after the last occupied F-square by reference to the marked transition in the *S.D.* and the current *complete configuration*

That is the basic concept of the first universal Turing machine. Note again that Turing defined these machines under the convention of never leaving an F-square within the sequence of figures blank and never changing any figure that has already been printed.
Under these conventions, U is indeed an universal Turing machine, i.e. U can simulate an arbitrary Turing machine M given its *Standard Description*, such that U prints the figures of the sequence computed by M and no other figures on F-squares. It does print other symbols on F-squares but the computed sequence can easily be distinguished from those, since none of these symbols will be 0 or 1.
Compared to the other universal Turing machines discussed in this work, Turing’s machine will appear to be one of the easiest regarding the encoding of the simulated Turing machine and the procedure of calculation. On the other hand this comes with the burden of size, while Turing did of course not anticipate to create a small UTM.

4 The idea of *small* universal Turing machines

In 1956, Claude E. Shannon published a paper called "A universal Turing machine with two internal states" and thereby introduced the factor size to universal Turing machines [12].
He examined three things in this paper: first, that it is possible to construct a universal Turing machine with one tape and two states. Second, that it is not possible to construct a universal Turing machine with only one state and last, similar to the first point made, universal Turing machines with two tape symbols are constructable. This chapter focuses mainly on the machine with two states and how this construction was achieved.

4.1 A universal Turing machine with 2 states

Shannon did not exactly construct a universal Turing machine but rather gave a method to transform every possible Turing machine into one with only two states. Therefore, universal Turing machines with two states can be constructed using this method and an existing universal Turing machine such as the UTM discussed above by Alan Turing.
The complete size of the resulting UTM is still depending on the universal Turing machine that was transformed in the first place. Let A be the UTM that is to be transformed using Shannon's method with an alphabet of size m $\{A_1, A_2, ..., A_m\}$ and n internal states $\{S_1, S_2, ..., S_n\}$, $m, n \in \mathbb{N}$. Let B be the resulting UTM, then B has 2 states (α and β) and an alphabet size of at most $4mn + m$. So as the number of states decreases (there was no UTM with only two states known when Shannon published this paper), the number of tape symbols increases. The exact alphabet size can be explained through looking at the method Shannon proposed:
Machine B should model the behaviour of machine A and display the same result as machine A at the end of the same calculation. To do this with only 2 states, B emulates the states of A by coding them in the tape symbols. If A is in state S_i and going to state S_j, B counts up to the next state on the next cell of its tape. Therefore, it jumps back and forth between the current and the next cell on the tape to count from 0 to j, thus saving the state information of A on the tape. To make sure that the symbol that was in this next cell beforehand is not lost, the tape symbols used to count carry the information about the state and the symbol. The old cell on the other hand is already carrying such a special state-symbol symbol. This has to be converted back to a normal symbol that is not carrying any information about the state as in machine A after finishing this bouncing operation, so that the result of the calculation may be the same as the result of machine A.
Therefore, machine B has m tape symbols B_i that directly correspond to the m tape

symbols of machine A. Furthermore, B has four tape symbols for each state-symbol pair of A. These are denoted by $B_{\mathrm{i,j,x,y}}$, where x and y are binary indices. The indice i describes the corresponding state of A as before, j denotes the j-th tape symbol of A, $y = L$ or R saving the motion of the head and $x = +$ or $-$ which corresponds to wether this cell is receiving $(-)$ or transmitting information $(+)$, as we will discuss later on. These tape symbols of B make up $m*n*2*2 = 4mn$, so that we get $4mn+m$ tape symbols altogether. The states α and β serve two purposes: in the beginning, α is used to signify that the old cell is to the right and β corresponds to the old cell being to the left. Else, an α signals the end of the operation.
The definite method is as follows: Machine B has the states α and β and tape symbols as listed above. Regarding the state transitions, the state transitions of A have to be modeled for B to mimic the behaviour of A. For every

$$S_i A_k \to S_j A_l R$$

in A there is a transition of form

$$\alpha B_{\mathrm{k,i,-,x}} \to \beta B_{\mathrm{l,j,+,R}} R$$

in B and corresponding for every

$$S_i A_k \to S_j A_l L$$

in A there is a transition of form

$$\alpha B_{\mathrm{k,i,-,x}} \to \alpha B_{\mathrm{l,j,+,L}} L$$

in B. The remaining state transitions for B are made up as follows:

$$\alpha B_i \to \alpha B_{\mathrm{i,1,-,R}} R, i \in 1, 2, .., m \tag{4.1}$$

$$\beta B_i \to \alpha B_{\mathrm{i,1,-,L}} L, i \in 1, 2, .., m \tag{4.2}$$

$$\beta B_{\mathrm{i,j,-,x}} \to \alpha B_{\mathrm{i,(j+1),-,x}} x, i \in \{1, 2, .., m\}, j \in \{1, 2, ..., n-1\}, x \in \{L, R\} \tag{4.3}$$

$$\gamma B_{\mathrm{i,j,+,x}} \to \beta B_{\mathrm{i,(j-1),+,x}} x, \gamma \in \{\alpha, \beta\}, i \in \{1, 2, .., m\}, j \in \{2, 3, ..., n\}, x \in \{L, R\} \tag{4.4}$$

$$\gamma B_{\mathrm{i,1,+,x}} \to \alpha B_i x, \gamma \in \{\alpha, \beta\}, i \in \{1, 2, .., m\}, x \in \{L, R\} \tag{4.5}$$

We will discuss this using an example: If machine A is in state S_3 and reads symbol A_2 it might go into state S_2 whilst writing A_7 on the tape and then move right using a transition

$$S_3 A_2 \to S_2 A_7 R$$

Then, machine B would have a transition

$$\alpha B_{\mathrm{2,3,-,x}} \to \beta B_{\mathrm{7,2,+,R}} R \tag{4.6}$$

and of course all other transitions as stated above. So machine B would read the symbol $B_{2,3,-,x}$ (wether x is L or R is not relevant at this point) and be in state α. We assume that the symbol on the next cell to the right is B_9, but it could be any arbitrary symbol of B. The actions of machine B are denoted in Table 4.1, where the transitions given express

Table 4.1: Actions of the two state UTM B

old cell	new cell	state	transition
$\mathbf{B_{2,3,-,x}}$	B_9	α	4.6
$B_{7,2,+,R}$	$\mathbf{B_9}$	β	4.2
$\mathbf{B_{7,2,+,R}}$	$B_{9,1,-,L}$	α	4.4
$B_{7,1,+,R}$	$\mathbf{B_{9,1,-,L}}$	β	4.3
$\mathbf{B_{7,1,+,R}}$	$B_{9,2,-,L}$	α	4.5
B_7	$\mathbf{B_{9,2,-,L}}$	α	-

the transition used to get from the current line to the next and the fat cell is the one looked at by B.
The last α does indeed end the operation as described before and the old cell is reset to a tape symbol directly corresponding to a tape symbol of A. Machine B always ends up in state α after executing one transition of A. We would now need a transition in A starting from S_2A_9 modeled in B to go on in the same manner and simulate a whole Turing machine.
Using this method, it is now possible to create a UTM with only 2 states and $4mn + m$ tape symbols.

4.2 Impossibility of a one state universal Turing machine

Shannon [12] goes on to prove that it is not possible to construct a UTM with only one state. He uses the computation of the irrational $\sqrt{2}$ to show *reductio ad absurdum* that a universal Turing machine with only one state given a suitable description number cannot compute this number and therefore, it is not possible to construct a universal Turing machine with only one state.

4.3 A universal Turing machine with two tape symbols

The universal Turing machine with only two tape symbols is similarly constructed as the one discussed above, rather transforming a given UTM than directly constructing one. In this case, Shannon [12] introduced more states to compensate the loss of different tape symbols in a similiar method to the first transformation. In contrary to coding the different states into the tape symbols, he uses binary sequences of a set length to code the different symbols only using 0 and 1 and therefore needs more states to process the symbols, as the machine can only read one cell at a time.

5 A competition arises

A few years after Shannon published his paper [12], a competition emerged to find the smallest possible universal Turing machine, lasting for the better part of the 1960s. Amongst other participants, Marvin Minsky and Shigeru Watanabe stood out as the two leading forces of this battle. This chapter focuses on the new ideas they brought up to reducce the size of universal Turing machines.

5.1 Watanabe's contribution

Watanabe found several small universal Turing machines as listed in table 7.1, but the main focus is on his 5-symbol 8-state universal Turing machine published in [17], as his other UTMs are either constructed very similar or are bigger than this.

5.1.1 The (5,8)-UTM

The 5-symbol 8-state UTM uses the set alphabet $\{0, 1, 0', 1', *\}$. To be able to use this alphabet with every Turing machine that could possibly be simulated, the simulated Turing machine must be expressed in such a way that it uses the alphabet $\{0, 1\}$. To ensure this, Shannons method of constructing a Turing machine with only two tape symbols 0 and 1 is used, as mentioned in 4.3. Moreover, the machine has to be transformed into one with a left end on its tape; otherwise, Watanabe's method cannot work.
The essence of his method lies in dividing the tape into three different regions:

- the I_α-Region
- the I_β-Region
- the W-Region

which are arranged on the tape in a certain order as presented in Figure 5.1.
They all serve a different purpose. Whilst both the I-Regions as instruction regions are used to encode the states and transitions of the Turing machine N that is to be simulated, the W-Region is holding the computations of the simulated TM as a working region. The difference between the I-Regions is that the I_α-Region codes the symbol that is to be written and the head movement that is to be made in the W-Region for every instruction of N. The I_β-Region, on the other hand, holds all state-symbol pairs of N.
The I_α-Region is written out as follows: for every instruction of N, there is a sequence $(10)^n 0^{00}_{11} 0$ written out, where n corresponds to the number of 0's from the left end of this

Figure 5.1: Arrangement of Regions on the tape of the (5,8)-UTM

table element in the I_α-Region until the right end of the element coding the next state of N in the I_β-Region. This ensures that the UTM can move from the current instruction to the next state. Furthermore, the first distinction between 0 and 1 shows whether the next symbol of N is a 0 or a 1 and the second codes whether the next head movement is right (0) or left (1).
To the left of the I_α-Region, the I_β-Region is composed. The I_β-Region is used to mark the current state and symbol read in the W-Region. For every state there are two sequences: 01^n0 for the state paired with symbol 1 and 1^m0 for the pairing with symbol 0 written out right behind each other. The number of 1's in each sequence corresponds to the number of 0's from the right end of this sequence until the left end of the instruction matching the state-symbol pair coded in this sequence. If we are at state q_i and read symbol c_i in the W-Region, the number of 1's corresponds to the number of 0's until we reach the instruction $q_ic_i \to q_{i'}c_{i'}X$ in the I_α-Region. At the end (reading from left to right) of the I_β-Region there is an additional 0 to mark the beginning of the I_α-Region.
The tape of the UTM is being prepared with the I_α- and I_β-Regions as described above for a specific Turing machine N that is to be simulated; the W-Region takes the content of the tape of N at the beginning of the computation. If N is in state q_a at the beginning of the computation and the head is on a specific cell of the tape, then the UTM is in its own starting state A and the head is placed on the corresponding cell in the W-Region. All 0's from the left end of the tape until the state-symbol pair $(q_a, 0)$ in the I_β-Region are converted to $*$'s. This is needed to be able to start the computation using the correct starting state of N.

Method of operation

The described universal Turing machine functions in a 5-step manner:

1. Read the current symbol on the tape in the W-Region and go to the I_β-Region
2. Read the current state in the I_β-Region and go to the corresponding entry in the I_α-Region
3. Go to the next state in the I_β-Region and mark it
4. Read the next symbol in the I_α-Region and write it in the current cell of the tape in the W-Region
5. Read the head movement in the I_α-Region and execute it in the W-Region

These 5 steps are executed for every instruction that occurs in N during the computation that is to be done. The steps are intertwined with an intricate coding system to find the next needed sequence in a different region featuring the number of 0's between these elements as described above and marking cells with * or 0' and 1' instead of 0 and 1.

The underlying ideas

The main ideas Watanabe used to construct this universal Turing machine are the division of the tape into two main parts, an instruction region and a working region. The former is used to code the transitions and states of N, and the latter to act as the tape of the

simulated machine N.
He also used Shannons method in an unconventional way to reduce the number of symbols. Instead of using it on his UTM after construction, he uses it to reduce the number of symbols of the ingoing Turing machine. This allows him to assure that the alphabet of all simulated TMs is fixed to $\{0, 1\}$. Furthermore, this enables Watanabe to use a small, fixed alphabet himself, adding only a few options to mark cells.

5.1.2 The (5,6)-UTM

The fundamental principle of this smaller universal Turing machine is the same as that of the 8-state UTM. Watanabe modifies the instruction region to save on states. Every possible transition has its own I_r-Region, combining the function of the I_α- and I_β-Region by using the general layout of the I_α-Region but also coding the current symbol into each instruction. Starting with I_1 as the rightmost I_r-Region, the next state-symbol-transition sequence is marked in the next I_r-Region to the left named I_{r+1}. This UTM has no left end, as the I_r-Region is repeated infinitely often. Therefore, the (5,6)-UTM does not need to change between an I_α- and an I_β-Region and needs fewer states than the (5,8)-UTM.

5.2 Minsky's contribution

Minksy on the other hand introduced a completely new method to construct small UTMs by using tag systems which were originally developed by Emil Post [10]. Amongst Minsky's many universal Turing machines, his 6-symbol 6-state UTM using tag systems that he found in 1962 [8] will be discussed in detail.

5.2.1 Tag systems

Tag systems are string manipulation systems and can also be viewed as machines with one infinite tape that functions as a FIFO queue.
A tag system consists of an alphabet $\{a_1, ..., a_m\}$ of symbols, a set of transitions over this alphabet and an integer P. These transitions, also called productions, define a string of symbols for every of the m symbols in the alphabet:

$$a_i \rightarrow a_{i,1}...a_{i,n_i}$$

These strings are also called P_i, where $i \in \{1, ..., m\}$ is referring to the alphabet symbol.
An operation cycle of a tag system will begin with a certain given input string. The machine reads the first symbol of the string. If it is a_i, it appends P_i to the string and deletes the first P symbols. This procedure is repeated until the string is empty due to deletions or a special halting symbol a_H is read.
There are some parallels between tag system and Turing machines. Both work on some kind of input string and manipulate it based on a symbol they have read. Tag systems are more limited in the way they read symbols, as they can only ever read the first symbol of the input string. They also can not remember anything that happened before, as they do not have states as Turing machines do. Therefore, tag systems seem to be less complicated than Turing machines.

5.2.2 Universality through tag systems

Iff a function can be calculated by a Turing machine, it is partial recursive. Minsky showed in [7] that there exists a tag system for every partial recursive function $h(x)$ which calculates the value of $h(x)$ in the following sense:
The alphabet of the tag system consists of A, a, B, b. If the input string is Aa^{2^x}, the tag system will generate $Bb^{2^{h(x)}}$ and no other string starting with B. If B is identified with the halting symbol, the calculation will stop and $h(x)$ can be encoded from the result. If there is a Turing machine which can simulate an arbitrary tag system, then this TM could effectively calculate every partial recursive function $h(x)$ and therefore, do every calculation another Turing machine could do, given a tag system and an input string. So this machine would be universal, as it can simulate every Turing machine by the function it calculates. So the universal Turing machine only needs to be able to simulate the operations of any given tag system.

5.2.3 The (6,6)-UTM

The 6-symbol 6-state universal Turing machine Minsky developed is able to emulate any given tag system. So the machine reads the first symbol of the input on the tape, appends the correct string for this symbol based on a transition of the tag system and erases the first P symbols of the input string. To do this, he also uses a tape with a left end and divides it into different regions as seen previously with Watanabe's UTM. The regions are:

(I) *Termination region*
(II) *Erasure region*
(III) *Production region*
(IV) *Spacing region*
(V) *Erased region*
(VI) *Working region*
(VII) *Empty space*

The regions are arranged on the tape from left to right, the *Termination region* being the leftmost region at the left end of the tape.
The UTM uses the alphabet $\{A, B, X, Y, I, O\}$ and encodes the different regions using it. All regions serve a different purpose:
The *Termination region* is used to halt if the representation of the halting symbol is encountered and to detect the end of an operation cycle and restore all conditons before beginning the next. The encoding of this region is always the same.
The *Erasure region* is made up of a large number of I's; I^Q and is used when erasing the first P symbols of the input string. The size of Q will be discussed later.
The *Production region* contains an encoding of all transitions of the tag system that is to be simulated, written down from right to left, as it will be read in this direction. For every a_i of the tag systems alphabet, there is a transition $a_i \rightarrow a_{i,\alpha}...a_{i,\omega}$. A production is encoded from right to left such that it starts with one O followed by a number of O's corresponding to $a_{i,\alpha}$ followed by an I. This process is repeated for all symbols in the right side of the production. The production is ended with another I, such that there are two I's at the end of every production. Note that only the right side of the production is encoded. This will be called the production string.
The *Spacing region* again contains a large number of I's; I^R. The size of R will also be

discussed later, as it is an imported part of working method of the UTM.
The *Erased region* is the part of the tape, that formerly held parts of the input string that have been erased now. It is entirely made up of O's.
Working region: This region contains a representation of the input string that is worked on. Every symbol a_i has a corresponding number N_i and is encoded by Y^{N_i}. An A is used to distinguish between the symbols in the string and marks the end of the current and the beginning of the next symbol.
Furthest right is the *Empty space*, the infinite remainder of the tape, completely covered in O's, as O is also acting as the blank symbol. It partly becomes part of the *Working region* during the appending of symbols to the working string.
After encoding of all regions, the tape will generally look like depicted in Table 5.1.

Table 5.1: Outline of the tape of the (6,6)-UTM

$OXIAAO$	I^Q	$(P_m)...(P_1)$	I^R	$OOO...OOO$	$Y^{N_\alpha}A...AY^{N_\omega}$	$OOO...$

Operation cycle

The procedure for an operation cycle of the tag system can be divided into 4 phases:
Phase 1: Read the first symbol of the working string and find the corresponding production (P_i) in the *Production region*.
Phase 2: Copy the production to the end of the working string, which is the left end of the *Empty space*.
Phase 3: Erase the first P symbol representations of the working string.
Phase 4: Restore all conditions for the next operation cycle.
This basic pattern is repeated until the machine halts. In combination with the encoding described before, the implementations of the different phases can be explained in greater detail.

Phase 1

In the first phase, the N_i Y's representing the first symbol of the working string, need to be read and the corresponding production (P_i) needs to be found. We mark the location of (P_i) by changing the intermediate symbols read such that the original symbols can still be restored. Phase 1 will end as soon as the first A in the working string is encountered.

To ensure that the changing of symbols by the location marker can be reversed, it has to be unambigious. Up to this point, the encoding only used the symbols I, O, A and Y. The remaining symbols of the alphabet, B and X, are used to mark the symbols already visited. Beginning from the left end of the *Working region*, the location marker will mark one cell further to the left for every Y read in the working string until an A is encountered. While doing so, every Y and O read on the way left are changed into an X and every B is changed to an I. Back on the way to the right every X is changed back to an Y and every I is changed to a B. The location marker goes left until it reaches the first I that has not already been changed to a B and is therefore unmarked. It then marks it and starts it journey out to the right until it reaches the first unmarked symbol of the working string, marks it with an X and turns again to the left. This is all put into practice through state

q_1, as the smart changing of the symbols spares using more than one state. The particular transitions are denoted in Table 5.2.
The process of moving the location marker ends exactly at the beginning of the correct production (P_i) through the encoding of the symbols and productions and the smart choice of R. The *Spacing region* consists of R I's. Remember that for every symbol a_i there exists a corresponding number N_i already used in the encoding of the productions and symbols. Let n_i be the nmuber of symbols in the production string (P_i). Every production (P_i) contains $n_i + 1$ I's; one between every symbol representation and two at the end of a production. Then N_i is defined as

$$N_i = \sum_{j=1}^{i-1}(n_j + 1) + R$$

and R will be defined below. If one now moves N_i I's to the left beginning from the right end of the *Spacing region*, one reaches exactly the beginning of production (P_i) in the *Production region*. Note that the beginning of a production is the right end of it, as it is encoded from right to left. Corresponding to this, every symbol a_i is encoded as Y^{N_i} as stated before, such that moving one I to the left for every Y in the encoding of the first symbol of the working string leads to the location marker marking every production until the one corresponding to this symbol.

Phase 2

In the second phase, the (P_i) zone located in phase 1 is read and the production string is copied to the end of the working string.
As explained before, the productions in the *Production region* are encoded using O's and I's, where we have $N_i + 1$ O's for every symbol a_i of the right side of the production and I's as separation symbols. For every first O in (P_i), an A is written down at the left end of *Empty space*. For every other O until an I is read, a Y is written down. If an I is read, the next O will belong to a new symbol and thus be copied again as an A. If 2 I's are read directly after each other, the end of (P_i) is reached and phase 2 is brought to an end. In detail, this is done by using states q_2, q_3, q_4 and q_5, where q_2 decides, whether the end of a production is reached and the other three states do the copying. If phase 1 ends because an A was read in state q_1, the machine moves to q_2 and goes back left changing the symbols on the way in the same manner as q_1 did. Note that q_2 should never encounter an X in this phase, as all X's were changed back to O's in the last iteration of q_1. The machine stops going left as soon as it reads the first O and transitions into state q_3, which leads the machine to go right and write an A in the beginning of the *Empty space*, which is at the end of the working string. The machine then transitions into a loop of q_4 and q_5, where q_4 means going back left until the next O in (P_i) and marking it with an X and q_5 means going right again until reaching the *Empty space* and writing down a Y. The loop ends as soon as an I is encountered in q_4. Then the machine transitions into state q_2 and looks at the next symbol to the left. If it is an O, the process can begin again with going into q_3. If it is an I, two consecutive I's are read, which marks the end of the production and phase 2 ends.
This process leads to an A for every first O of the encoding of one symbol in the production string and N_i Y's if the encoded symbol is a_i because every symbol in a production string is encoded using one more O than the corresponding number N_i of the symbol a_i.

Phase 3

The third phase works exactly like the first using the same state. The first symbol of the working string was already erased in phase 1 through changing the Y's into X's as well as the O's, so they are non-distinguishable. Therefore, only $P-1$ symbols have to be erased in this phase. Using the same state to erase as in phase 1 is not going to lead to any copying after this phase because the location marker is not going to be placed in the *Production region*. The length of the *Erasure region* ensures that exactly P symbols are erased in total for every production cycle. This is controlled by the choice of R and Q. Let S be the total number of I's in the *Production region*, defined by

$$S = \sum_{i=1}^{m} (n_i + 1)$$

where m is the size of the alphabet. Then let R be defined as

$$R = PS$$

with P being the deletion number of the tag system and $R \leq N_i < R + S$. Lastly Q is defined as

$$Q = (P-1)R - S$$

such that the total number of I's in regions II, III and IV is $S + R + Q = PR$. Note that this number is strictly greater than the number of I's encountered by the location marker in phase 1, 2 and 3 for every combination of $P-1$ symbols at the beginning of the working string. This means that the *Termination region* will never be reached in the process of erasing the first $P-1$ symbols. On the other hand, the number of I's encountered during the erasure of P symbols would be strictly smaller than PR. So *Termination region* will always be reached if P symbols are erased.
This ensures that the two A's in the *Termination region* will always be read by the machine which leads to a transition from q_1 to q_2 and from there to q_6. So no copying of symbols will occur and with the transition into q_6, phase 4 begins.

Phase 4

In the fourth phase, the location marker has reached the *Termination region* after erasing P symbols. This leads the machine to restore the conditions for the next operation cycle by reversing the marks of the location marker and erasing possible residue of the last symbol to be erased on the tape including the next A. The machine is then able to start the next operation cycle with phase 1.
In detail, state q_6 leads the machine to go right from *Termination region* until the first A in the working string. On its way it changes all B's back to A's and all X's and Y's to O's. This restores the tape back into its standard encoding described before including possible remaining unread Y's at the beginning of the working string that have not been processed in the erasure process because the *Termination region* was reached before all Y's of the Pth symbol have been encountered. The next separating A of the working string is also erased by changing it to an O, as an operation cycle begins with the head of the machine on the first Y of the working string in *Working region*. The machine is now ready to transition back into q_1 and begin the next operation cycle.

If the machine should halt in this operation cycle instead of beginning a new one, then this phase will be encountered with the location marker marking exactly all Q I's in region II. This is then going to lead to halting of the machine because in all other cases the next symbol to the left is not going to be an O, which is the rightmost symbol in region I, but the erasure process of phase 3 is going to be stopped by encountering the two A's of region I. The machine only moves the location marker to exactly the leftmost I of the *Production region*, if a sequence of Y's was read in the working string that corresponds to a special halting symbol a_H. This number of Y's has to be exactly PR as this is the total number of I's inbetween the working string and the *Termination region*. So there exists a production (P_H) that is encoded as $(P_H) = IIO^{PR}OIO^{PR}O$ that leads to the appendance of the string $AY^{PR}AY^{PR}$ to the working string. It is necessary that Y^{PR} is repeated because otherwise this symbol representation may be encountered in an erasure phase. Then the total number of I's read would not be PR. But it would lead the machine to start the next operation cycle with the second Y^{PR} as a symbol, which then would lead to marking exactly PR I's and therefore reaching the rightmost O in the *Termination region* in state q_2. This can only occur by reading Y_{PR} in phase 1 and will then lead to halting because q_6 will read an O. So the UTM will halt, if and only if a halting symbol is encountered.

Table 5.2: Transition table of the (6,6)-UTM

	q_1	q_2	q_3	q_4	q_5	q_6
O	XL	XRq_3	ALq_4	XRq_5	YLq_4	$HALT$
I	BR	BRq_1	BR	ILq_2	BR	IR
A	XLq_2	BR	AR	AL	AR	ORq_1
B	IL	IL	-	IL	-	AR
X	YR	BLq_6	XR	XL	XR	OR
Y	XL	XL	YR	YL	YR	OR

5.2.4 The (4,7)-UTM

Minsky furthermore constructed an universal Turing machine with only four symbols, but seven states which is also based on simulating tag systems. This UTM can only simulate tag systems with deletion number $P = 2$ which is sufficient to construct an universal TM, as Minsky and Cocke prooved those to be universal in [2]. Fixing this parameter enables him to cut out the regions I, II and IV because after erasing the first symbol in phase 1, the erasure of the second symbol in phase 3 can be hard coded. Furthermore, the alphabet symbols X and B are not needed anymore. The machine does need one more state for phase 1 and phase 3 respectively because of this symbol loss.
Otherwise, the encoding is generally the same, as well as the method. This tiny bit of more information about the tag system that is to be simulated made it possible to decrease the symbol-state product from 36 to 28, which made the (4,7)-UTM the smallest known universal Turing machine for several years.

6 How small is possible?

After Minsky's (4,7)-UTM there were some more small UTMs found, improving only slightly on his results, as denoted in Table 7.1. In more recent times, Stephen Wolfram found two interesting and very small universal Turing machines, yet again with another method, improving massively on the previous results. He examined Turing machines by visualizing them and found one by using the Rule 110 cellular automaton.

6.1 The Rule 110 cellular automaton

A cellular automaton is given an infinitely long row of cells, similar to the tape of a Turing machine. Each cell bears a symbol. For every time step, a so called new generation is created, which is a new row of cells mostly displayed to the bottom of the current row. Cellular automata define the symbol of a cell of the next generation based on the symbol of the current cell and its neighbors by some transition function.
Rule 110 is one of the so called elementary rules [19] that consist only of two different symbols, or colours, and decide the symbol of a cell in the next generations based on the cell itself and its direct neighbours. The transition function of Rule 110 is given in Table 6.1.
Mostly, colours are used rather than symbols. A black cell corresponds to a 1, a white or blank cell corresponds to a 0. That is why Wolfram uses colours rather than symbols in his UTMs. The first ten generations are displayed in Figure 6.1 starting with one black cell. This particular Rule is called Rule 110 because it can be summarized with the sequence 01101110 for the transition function. If this sequence is interpreted as a binary number, its decimal value is 110. This numbering system was introduced by Wolfram in [18].

Table 6.1: Rule 110 transitions

111	110	101	100	011	010	001	000
0	1	1	0	1	1	1	0

6.2 Universality of Rule 110

The universality of the Rule 110 cellular automaton was proven separatly by Wolfram [19] and Cook [3]. Both use the fact that cyclic tag systems are already proven to be universal (by simulating tag systems [3]) and show that the rule 110 cellular automaton is capable of emulating an arbitrary cyclic tag system.
A cyclic tag system is similar to a tag system which was explained before in Chapter 5.2.1. The differences between those two are mainly that cyclic tag systems have an alphabet

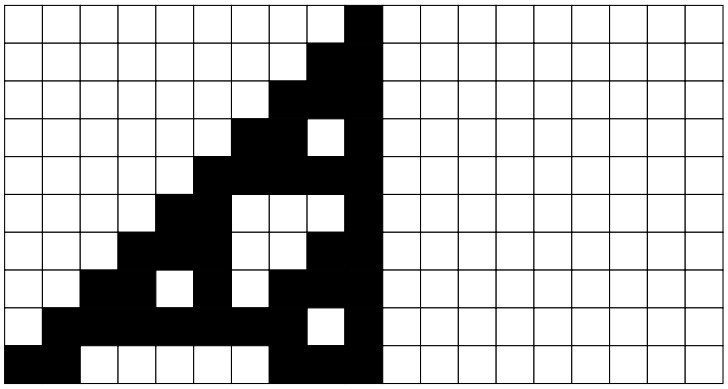

Figure 6.1: First 10 generations of Rule 110

that consists of 0 and 1 and only ever erase the first symbol of a string, corresponding to $P = 1$ in a tag system. Furthermore, the string appended does not depend on the symbol read. The transition function only consists of strings to be appended, which are appended in a cyclic manner; starting with the first, then the second until the last and then cycling back to the first again. But this appendance happens only if the symbol read was a 1, not if it was a 0.
While both Cook and Wolfram use structures generated by Rule 110 to prove its universality, the methods of proof differ between them. The details of these proofs are not discussed, as the rather important information is that Rule 110 is indeed universal. The (5,2)-UTM of Wolfram emulates Rule 110. Because Rule 110 is universal, any Turing machine that emulates it is universal, too. Note that none of these machines halt because cellular automata do not halt and differ in this way from the other universal Turing machines discussed.

6.3 The (5,2)-UTM

The (5,2)-UTM given by Wolfram in [19] are displayed by the transition table in Table 6.2.

Table 6.2: Transition Table of the (5,2)-UTM

	q_1	q_2
0	AL	BLq_1
1	BLq_2	CL
A	$0R$	$0Rq_1$
B	$1Rq_2$	$1R$
C	$0R$	$1Rq_1$

The symbols 0 and 1 correspond directly to the symbols 0 and 1 or the color white and black of the Rule 110 cellular automaton. The symbols A, B and C are used to emulate the actions of Rule 110. Since a Turing machine cannot look at more than one cell at the same time, it is necessary to include more symbols and also more than one state to

memorize the important colours of the neighbours of a cell. Wolfram initially depicted the universal Turing machine as a picture, similar to a cellular automaton. So A, B and C could also be displayed as colors in between 0 and 1 or white and black. This depiction is used to examine how this UTM works. A is a light grey tone, B a dark grey tone and C a grey tone in between A and B.
The same kind of Figure as Figure 6.1 is constructed, starting with a slightly altered input, featuring only one black cell but light grey cells from the left of the rightmost white cell that is still left of the black cell until the left end, as in Figure 6.2. The way of depiction will be the same as for a cellular automaton but a new generation, i.e. a new row of cells, will be added for every step of the universal Turing machine where nothing has changed besides the one cell that was examined by the machine. If this picture is compressed in its length, it is identical to the one created by Rule 110. If this is achieved for an arbitrary input, it does indeed show that the Turing machine simulates Rule 110 and is therefore to be considered universal.

Figure 6.2: Initial complete configuration of the (5,2)-UTM

The picture 6.3 shows the first generations of the UTM as explained before. Note that the little black drop marks the cell currently looked at and at the same time a downward drop means the machine is in state q_1 and correspondingly an upward drop denotes that the machine is in state q_2.
This makes up to the first 5 generations of the Rule 110 cellular automaton if compressed such that every uninterrupted line of black or white cells makes up to one generation. After each interruption by grey cells, the next generation is formed. The result after compression is depicted in Figure 6.4.
This shows how the universal Turing machine emulates Rule 110. Wolfram does neither give a thorough proof nor a detailed explanation of how this machine works in [19].

6.4 The (3,2)-UTM

The (3,2)-UTM was probably an educated guess by Wolfram, as he seemed to have looked at the visualized behaviour of possible (3,2)-UTMs in his book [19] and found this one to behave rather complex, i.e. it creates complex structures that do not resolve into simple forms.
Wolfram himself did not prove this machine to be universal but set up a competition to prove whether the machine is universal or not. Smith won the prize with his proof of universality [13], as Wolfram announced on his blog [20]. Table 6.3 shows the transitions of this very small universal Turing machine.
The proof of universality is very complex; Smith had to construct multiple intermediate steps to be able to set up every program to be executed by the UTM. The tape of the machine needs to be prepared with an infinite pattern of bits, taking a whole kind of compiler to be generated, which is very inefficient. But the computation done to generate this infinite sequence is not universal; the universal computation is solely done by the Turing machine, proving its universality. Smith again used cyclic tag systems to encode

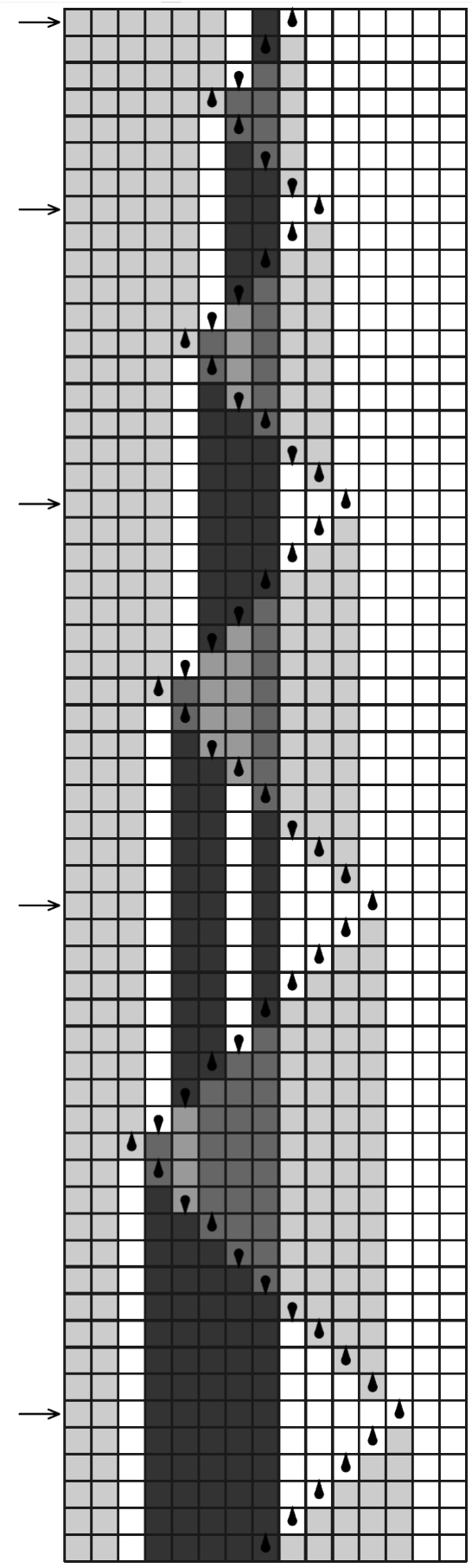

Figure 6.3: Detailed actions of the (5,2)-UTM, Wolfram [19], p.707

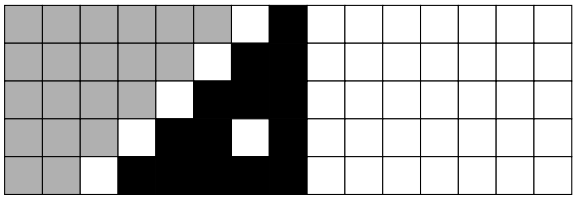

Figure 6.4: First 5 "generations" of the (5,2)-UTM

as the input rather than Turing machines themselves. The (3,2)-UTM also does not halt, which is a difference to other previously discussed Turing machines and a deviation from some definitions of universal Turing machines.
Besides the difficulties to encode the input for the (3,2)-UTM, Wolfram claims it is the smallest possible universal Turing machine [20]. As already discussed in Chapter 4.2, it is not possible to construct an universal Turing machine with only one state. Therefore, at least two states are needed. Furthermore, Margenstern published a paper stating that there is no universal Turing machine with 2 states and 2 symbols because all Turing machines with these properties have a decidable halting problem [5]. Because the halting problem is undecidable for Turing machines in general, these machines cannot be universal.

Table 6.3: Transition Table of the (3,2)-UTM

	q_1	q_2
A	BL	CRq_1
B	AL	AR
C	BRq_2	ALq_1

This leads to the understanding that there exists no smaller universal Turing machine than an UTM with a symbol-state product of 6. The (3,2)-UTM of Wolfram satisfies this condition and is therefore the smallest universal Turing machine that has been found. There might be more universal Turing machines with this symbol-state product, but definitely no smaller UTMs.

6.4.1 Concluding words

In conclusion, starting from the first inconceivably big universal Turing machine by Turing himself (see Chapter 3), to the idea of size of UTMs by Shannon (see Chapter 4) and the early race of finding the smallest UTM, mainly impelled by Watanabe and Minsky (see Chapter 5) to the new ideas of Wolfram, the smallest universal Turing machine eventually was found.
Of course there were some other entries to this competition along the way that were not discussed in this work. Some of them are mentioned in Chapter 7. But there are still more universal Turing machines that have been published and are not mentioned in this work; with more subcategories such as weakly universal Turing machines [21]. These do not follow the definition of a classical universal Turing machine, e.g. they have more than one tape. Some simply did not use a breakthrough idea that led to a significant reduction

in size. Please note that this work is not comprehensive regarding the progression of small universal Turing machines.

7 Overview

Table 7.1: Overview over published small UTMs

Symbol	State	Product	Reference
2	M	2M	Shannon [12], see Chapter 3
N	2	2N	Shannon [12], see Chapter 3
6	12	72	Takahashi [14]
6	10	60	Ikeno [4]
3	17	51	Watanabe [16]
2	25	50	Minsky [9]
6	7	42	Minsky [6]
5	8	40	Watanabe [17], see Chapter 4.1
6	6	36	Minsky [8], see Chapter 4.2
5	6	30	Watanabe [17], see Chapter 4.1
4	7	28	Minsky [8], see Chapter 4.2
4	7	28	Baiocchi [1]
6	4	24	Rogozhin [11]
5	2	10	Wolfram [19], see Chapter 5
3	2	6	Wolfram [19], see Chapter 5

Table 7.1 presents a brief overview over small universal Turing machines and their size. In addition to the previously discussed UTMs, a few mentionable results are named. This overview is not complete; many more universal Turing machines have been found.

List of Figures

List of Tables

Bibliography

[1] C. Baiocchi. Three small universal Turing machines. In *International Conference on Machines, Computations, and Universality*, pages 1–10. Springer, 2001.

[2] J. Cocke and M. Minsky. Universality of tag systems with p=2. *Journal of the ACM (JACM)*, 11(1):15–20, 1964.

[3] M. Cook. Universality in elementary cellular automata. *Complex systems*, 15(1):1–40, 2004.

[4] N. Ikeno. A 6-symbol 10-state universal Turing machine. In *Proceedings, Institute of Electrical Communications, Tokyo*, 1958.

[5] M. Margenstern. Turing machines with two letters and two states. *Complex Systems*, 19(1):29, 2010.

[6] M. L. Minsky. *A 6-symbol 7-state Universal Turing machine.* Massachusetts Institute of Technology, Lincoln Laboratory, 1960.

[7] M. L. Minsky. Recursive unsolvability of Post's problem of" tag" and other topics in theory of Turing machines. *Annals of Mathematics*, pages 437–455, 1961.

[8] M. L. Minsky. Size and structure of universal Turing machines using tag systems. In *Recursive Function Theory. Proceedings of the Symposium in Pure Mathematics*, volume 5, pages 229–238. AMS, 1962.

[9] M. L. Minsky. Universality of (p=2) tag systems and a 4 symbol 7 state universal Turing machine. 1962.

[10] E. L. Post. Formal reductions of the general combinatorial decision problem. *American journal of mathematics*, 65(2):197–215, 1943.

[11] Y. Rogozhin. Small universal Turing machines. *Theoretical Computer Science*, 168(2):215–240, 1996.

[12] C. E. Shannon. A universal Turing machine with two internal states. *Automata studies*, 34:157–165, 1956.

[13] A. Smith. Universality of wolfram's 2, 3 Turing machine. 2007.

[14] H. Takahashi. Keisan kikai ii. *Iwanami Gendai Oyo-Sugaku Koza*, 14(3):117–128, 1958.

[15] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.

[16] S. Watanabe. On a minimal universal Turing machine. *MCB Report*, 1960.

[17] S. Watanabe. 5-symbol 8-state and 5-symbol 6-state universal Turing machines. *Journal of the ACM (JACM)*, 8(4):476–483, 1961.

[18] S. Wolfram. Statistical mechanics of cellular automata. *Reviews of modern physics*, 55(3):601, 1983.

[19] S. Wolfram. *A new kind of science*, volume 5. Wolfram media Champaign, IL, 2002.

[20] S. Wolfram. The prize is won; the simplest universal Turing machine is proved, 2007. `https://blog.wolfram.com/2007/10/24/the-prize-is-won-the-simplest-universal-turing-machine-is-proved/`; accessed on 12.02.2020.

[21] D. Woods and T. Neary. The complexity of small universal Turing machines: A survey. *Theoretical Computer Science*, 410(4-5):443–450, 2009.