Bachelor of Science Thesis

Institut für Informatik
Ludwig-Maximilians-Universität München

State of the Church-Turing Thesis

Kilian Parigger
matriculation number: 10409103
date: 11.11.2019
supervised by: Professor Dr. Francois Bry

Contents

1 Introduction 1

2 History of the Church-Turing Thesis 2

2.1 The Concept of the Algorithm and Computation 2

2.2 The Entscheidungsproblem 4

2.3 Different Approaches to Formalise the Concept of the Algorithm 7

2.3.1 Recursive Functions 7

2.3.2 The Turing Machine 10

2.4 The Development of the Church-Turing Thesis 11

3 Epistemological State of the Church-Turing Thesis 12

4 Quantum Computing and the Church-Turing Thesis 14

4.1 Brief History of Quantum Computing 14

4.2 Basic Notions of Quantum Computing 16

4.3 Quantum Computation 18

4.3.1 The Qubit 19

4.3.2 Manipulating Qubits 20

4.3.3 Quantum Register 21

4.3.4 Measurement of Quantum registers 25

4.4 A Model of Quantum Computing 26

4.5 Deutsch's Algorithm 28

4.6 Quantum Computing and the Church-Turing Thesis 30

4.7 Future Prospects 30

5 DNA Computing 32

5.1 History of DNA Computing 33

5.2 Foundations of DNA Computing 34

5.3 The Adleman Experiment 36

5.4 DNA Computing Models and the Church-Turing Thesis 40

5.5 Future Prospects 40

6 Summary and Conclusion 41

7 References 43

8 Statement of Originality 45

1 Introduction

In 1936, the mathematician Alan Turing published a landmark paper called *On Computable Numbers, With an Application to the Entscheidungsproblem*.[1] In his paper, Turing was aiming to solve a very significant mathematical problem, called *Entscheidungsproblem*. While doing so, he developed a quite simple mathematical formalism which is now known as *Turing machine*. Though very simply constructed, the Turing machine proved to be very powerful. In fact, Turing argued very convincingly that almost every mathematical function can be successfully computed by a Turing machine. What was even more remarkable is the fact that Turing brought up his idea at a time where almost no one knew what 20 years later computers would be.

Nowadays, many years later, computers are ubiquitous and have been continuing their triumph for decades. But although researchers and engineers all over the world keep developing and constructing computers that are still more elaborate and faster than the ones they developed and constructed before, none of these new machines is, in principle, more powerful than the Turing machine which was proposed in 1936.

This remarkable and astonishing result is captured by the Church-Turing Thesis. The thesis states what is principally computable. There are various formulations of the Thesis, for example *Every function that can be computed by the idealized human computer, which is to say, can be effectively computed, is Turing-computable*[2] or *Every Algorithm can be expressed by means of a program in some (not necessarily currently existing) Turing-equivalent programming language*[3] or *every effective computation can be carried out by a Turing machine*[4].

This bachelor thesis aims to investigate the state of the Church-Turing Thesis. In doing so, the object of study will be what one could call the original Church-Turing Thesis, as it was mentioned above in various versions. There are, however, many other so-called Church-Turing Theses that are referring to other phenomena. Consider, for example, the so-called

[1] Petzold, 2008, p. 64.
[2] Copeland, Shagrir, 2019, p. 67.
[3] Copeland, Shagrir, 2019, p. 68.
[4] Copeland, 2017.

physical Church-Turing Thesis: *Every function computed by any physical system is Turing-computable*[5] which is, again, out of the scope of this bachelor thesis.

To investigate the state of the Church-Turing Thesis, first the historical development of the thesis is described. In doing so, the intuitive concept of algorithm is explained and different approaches to formalize that concept are presented. Second, the epistemological state of the thesis is discussed. In doing so, it is asked what reasons do exist that justify the thesis. Third, new computing models are presented: quantum computing and DNA computing. It is analysed whether these new technologies are challenging the Church-Turing Thesis. In doing so, these new computing models are quite extensively presented. Fourth, this thesis is concluded by a summary of its investigation and outcomes.

2 History of the Church-Turing Thesis

The purpose of this chapter is to describe the history of the Church-Turing Thesis. First, the intuitive concept of the algorithm and computation is described. Second, it is briefly analysed how David Hilbert's Program, *a promising formalistic attempt that would use formal axiomatic systems to eliminate all the paradoxes from mathematics*[6] gave rise to various attempts to formalize the concept of the algorithm. Third, two of these attempts are presented. Fourth, it is explained how those attempts led to the formulation of the Church-Turing Thesis.

2.1 The Concept of the Algorithm and Computation

Today, it's well known that computers are somehow using algorithms to somehow compute things. But what exactly is an *algorithm*? What does it mean to *compute* something? An *informal* description of these two concepts is provided now, the *formal* description will be given later.

Even in ancient times, people were confronted with a certain class of problems. For example, they had to solve problems like this: *Calculate the greatest common divisor of 8*

[5] Copeland, Shagrir, 2019, p. 74.
[6] Robič, 2015, p. viii.

and 12 or *find every prime number less than 100*. Such problems can be represented by two sets A and B and a function $f: A \to B$, where A contains all the possible input data to the problem, B contains all the possible solutions to the problem and f maps the input data to the corresponding solution of the problem.[7] How does one solve these types of problems, or, more specifically, how does one calculate the value $f(x)$ for a certain x?

As time went on, people developed a certain type of procedures, that governed the process of solving those problems. Such procedures consist of a finite set of unambiguously formulated instructions. To solve a certain problem, one had to just mechanically follow these instructions. No such thing as intuition or ingenuity is needed to obtain the requested result.

The Euclidean algorithm is a telling example for that sort of procedure. If you take a closer look at it, you can derive certain characteristics, that describe that sort of procedure more closely:

- it is specified by a finite set of instructions
- the calculation proceeds in a finite sequence of steps, eventually halting with the answer
- it proceeds in a deterministic, completely fashion without any recourse to human intelligence or random devices.[8]

So, we can come up with a first, informal definition of the algorithm:

Definition (Algorithm Intuitively)*: An algorithm for solving a problem is a finite set of instructions, that lead the processor, in a finite number of steps, from the input data of the problem to the corresponding solution.*[9]

Here, some further explanations are required. The processor, that carries out the algorithm is either *a human, or a mechanical or electronically or any other device capable of mechanically following, interpreting and executing instructions, while using no self-reflection*

[7] Robič, 2015, p. 3.
[8] Soare, 2016, p. 3.
[9] Robič, 2015, p. 4.

or external help.[10] The instructions have to be simple and well-defined, so that the processor can follow them without referring to additional help. Each time the algorithm is carried out, a computation is done, so you can say, a computation is a single execution of the algorithm. The input data must be *reasonable*[11] in a way, that it can mapped to a corresponding solution of the problem.[12]

So far, the intuitive concept of the algorithm and computation has been explained. What is most important to remember is that even the intuitive concept of the algorithm concerns an instruction to calculate the value of a function. That's way later on, when one tries to formalize the intuitive concept, it is always about a certain class of functions that is concerned by the formalization.

2.2 The Entscheidungsproblem

The intuitive, informal understanding of the concept of the algorithm remained unchallenged for a very long time. But then, suddenly, during the 20th century scientists started to reconsider what they were thinking about the concept of the algorithm so far. They were doing so to solve a certain problem, part of the so-called Entscheidungsproblem, which is now briefly explained.

At the beginning of the 20th century, the mathematician Georg Cantor laid, by developing his set theory, the foundations of modern mathematics.[13] Unfortunately, after a while, mathematicians discovered that the set theory contained some serious logical paradoxes. A common example for such a paradox is the Russel's Paradox, discovered by the British mathematician and logician Bertrand Russel. Russel detected that in Cantor's set theory it was possible to construct a set that at the same time contains itself and doesn't contain itself. Thus, Russel found a paradox, embedded in Cantor's set theory. In general, a theory contains a paradox, if there's a logical statement where both, the statement and its negation can be deduced. This is a serious problem because then, any statement can be deduced in that theory, so the theory becomes useless.[14]

[10] Robič, 2015, p. 4.
[11] Ibid., p. 4.
[12] Ibid., p. 4.
[13] Ibid., p. 9.
[14] Ibid., p. 18.

In order to preserve the set theory as the basic foundation of mathematics, many mathematicians tried to revise the set theory in such a way that there won't be any paradoxes anymore. The most sophisticated and systematic attempt to do that was made by the famous German mathematician David Hilbert. During the 1920s Hilbert proposed *a promising formalistic attempt that would use formal axiomatic systems to eliminate all the paradoxes from mathematics*[15]. This attempt, the so-called *Hilbert Program*, *consisted of the following goals:*

(i) *find an f.a.s. [formal axiomatic system] M capable of deriving all the theorems of mathematics;*

(ii) *prove that the theory M is semantically complete;*

(iii) *prove that the theory M is consistent;*

(iv) *construct an algorithm that is a decision procedure for the theory M.*[16]

It would go far beyond the scope of this bachelor thesis to describe Hilbert's Program in detail. Instead, we will focus on the fourth goal of it, since this particular part of the program initiated the quest for a formalisation of the algorithm and eventually led to the foundation of Computability Theory.

The fourth goal of Hilbert's Program is also known as *Entscheidungsproblem*. It is now briefly described.

Let $\mathcal{F}$ be a syntactically complete and consistent theory, based on first order logic. Let F be a formula in $\mathcal{F}$. Proving either F or $\neg F$ could be very complicated and, thus, would require lots of creativity or ingenuity. However, since $\mathcal{F}$ is syntactically complete, there definitely exists a proof for either F or $\neg F$. What if there was some effective procedure, that could for each F in $\mathcal{F}$ automatically and during a finite amount of time decide, whether F or $\neg F$ is true. The procedure would be called *effective*, since it could decide for any formula F in $\mathcal{F}$ whether F or $\neg F$ is true. If $\mathcal{F}$ is a theory where such a procedure exists, then $\mathcal{F}$ is called *decidable*. A decidable theory allows you, at least in principle, to derive every theorem of

[15] Ibid., p. viii.
[16] Ibid., p. 53.

that theory without referring on creativity or ingenuity. The propositional calculus, for example, is a decidable theory.[17]

After the mathematician and logician Kurt Gödel had published the results of his famous two incompleteness theorems[18], Hilbert's Program was shattered. Although relying on formal axiomatic systems could avoid logical paradoxes, Gödel proved that this comes at a very high price: *The mathematics developed in this way suffers from semantic incompleteness and the lack of a possibility of proving its own consistency*.[19] [20]

Even though the second and third goal of Hilbert's Program turned out to be unachievable, the first and the last goal remained achievable. Especially the last goal, constructing an algorithm that is a decision procedure for the theory M, kept scientists interested. Spurred on the results of Gödel's incompleteness theorems, they tried to prove that there is no such algorithm.[21]

In order to do so, they first had to face another problem: When Hilbert was formulating the fourth goal of his program, *construct an algorithm that is a decision procedure for the theory* M, he had the intuitive concept of the algorithm in mind: *An algorithm for solving a given problem is a recipe consisting of a finite number of instructions which, if strictly followed, leads to the solution of the problem*.[22] So if one wanted to prove that there exists such an algorithm, it would be sufficient to first construct that algorithm, then check whether it meets the requested conditions (consisting of a finite number of instructions etc.) and finally show that it solves the problem. But proving that a certain algorithm doesn't exist would be much more intricate, since this would force one to show that each of the infinite possible recipes doesn't meet the wanted conditions. In order to achieve that one would have to find some characteristics that are only meet by algorithms in general. After having found such characteristics, one would finally be able to mathematically prove that none of the possible

[17] Robič, 2015, p. 51.
[18] Gödel. 1931, p. 173 ff.
[19] Robič, 2015, p. 63.
[20] Ibid., p. 63.
[21] Ibid., p. 71.
[22] Ibid., p. 70.

recipes would meet the conditions required. Such a definition would be called a model of computation.[23]

So, the scientists started thinking about what such a model of computation could possibly look like. They were guided by looking for some appropriate, well known examples. As a result, three different approaches were pursued: modelling the computation after *functions*, after *humans* and after *languages*.[24] These approaches included such diverse ideas like Turing Machines, the Lambda Calculus, Recursive Functions, Post Machines etc. Two of these ideas will now be described.

2.3 Different Approaches to Formalise the Concept of the Algorithm

Now, two different approaches to formalise the concept of the algorithm are presented: Recursive functions and Turing machines. Each concept is first briefly described on a rather informal level, then, the formal definition will be presented.

2.3.1 Recursive Functions

While describing the intuitive concept of the Algorithm, we discovered that there is a certain kind of problems that can be represented by two sets A and B and a function $f: A \to B$, where A contains all the possible input data to the problem, B contains all the possible solutions to the problem and f maps the input data to the corresponding solution of the problem. So, solving this kind of problems would mean that you would have to calculate, for some $x \in A$, the value $f(x)$. But what does it mean to calculate the value of a function $f: A \to B$? Or what does it mean, if a function is *computable*?

In order to get an answer to these questions, it's quite convenient to focus on the simplest functions you could possibly think of. These functions are called total numerical functions $f: \mathbb{N}^k \to \mathbb{N}$, where $k \geq 1$. A total numerical function f associates to every k-tuple $(x_1, \dots, x_k) \in \mathbb{N}^k$ exactly one natural number called the value of f at $(x_1, \dots, x_k)$ and denoted by $f(x_1, \dots, x_k)$.

[23] Robič, 2015, p. 71 f.
[24] Ibid., p. 72.

If you look for a definition of the total numerical functions, you have to keep in mind two requirements:

(i) Completeness Requirement: the definition should include *all* the "computable" total numerical functions, *and nothing else*.

(ii) Effectiveness Requirement: the definition should make evident, for each such function f, an *effective procedure* for computing the value $f(x_1, \dots, x_k)$. Here an **effective procedure** is defined to be any finite set of instructions written in any language that

   a. completes in a *finite* number of steps;
   b. returns *some* answer, that is, some natural number;
   c. returns the right answer, that is, the value $f(x_1, \dots, x_k)$; and
   d. it does so for *all* instances of the problem, that is, for any $(x_1, \dots, x_k) \in \mathbb{N}^k$.[25]

One approach to meet these two requirements was pursued via the theory of recursive functions. The idea that led to the definition of the recursive functions was originated from Gödel. In his second incompleteness theorem, Gödel introduced three initial functions $\zeta: \mathbb{N} \to \mathbb{N}$, $\sigma: \mathbb{N} \to \mathbb{N}$ and $\pi_i^k: \mathbb{N}^k \to \mathbb{N}$ (called *zero*, *successor* and *projection* function) and two rules of construction, called *composition* and *primitive recursion* which allow to construct new functions from the initial ones and previously constructed ones. These functions are called primitive recursive functions.[26]

Now the question was, whether the primitive recursive functions would meet the completeness and the effectiveness requirement. Let's start with the later one. For computing the value of a primitive recursive function, would there be an effective procedure to do that?

Definition: The class of the *primitive recursive functions* is the smallest class $\mathcal{C}$ of functions such that the following hold:

[25] Robič, 2015, p. 70.
[26] Ibid., p. 70.

(i) The successor function $S(x) = x + 1$ is in $\mathcal{C}$

(ii) All constant functions $M_m^n(x_1, x_2, \dots, x_n) = m$ for $n, m \in \mathbb{N}$ are in $\mathcal{C}$

(iii) All projections (or identity) functions $P_i^n(x_1, x_2, \dots, x_n) = x_i$ for $n \leq 1, 1 \leq i \leq n$ are in $\mathcal{C}$

(iv) (Composition or substitution.) If $g_1, g_2, \dots, g_m, h$ are in $\mathcal{C}$ then $f(x_1, x_2, \dots, x_n) = h\big(g_1(x_1, x_2, \dots, x_n), \dots, g_m(x_1, \dots, x_n)\big)$ is in $\mathcal{C}$ where the g_i are functions of n variables and h is a function of m variables

(v) (Primitive recursion or just recursion.) If $g, h \in \mathcal{C}$ and $n \geq 0$ then the function f defined below is in $\mathcal{C}$: $f(x_1, \dots, x_n, 0) = g(x_1, \dots, x_n)$ $f(x_1, \dots, x_n, y + 1) = h\big(x_1, \dots, x_n, f(x_1, \dots, x_n, y)\big)$ where g is a function of n variables and h a function of $n + 2$ variables.[27]

It took, however, not very long, until some researchers discovered functions that couldn't be computed by any primitive recursive function. The most famous of such functions was suggested by Ackermann and is called *Ackermann function*.[28] This problem was solved by Kleene in 1936. He added another construction rule to the ones proposed by Gödel. The new construction rule was called μ-operator and is defined below:

If $x' = x_1, \dots, x_n$, $\Theta(x', y)$ is a partial recursive function of $n + 1$ variables and we define $\Psi(x')$ to be the least y such that $\Theta(x', y) = 0$ and $\Theta(x', z)$ is defined for all $z < y$, then Ψ is a partial recursive function of n variables.[29]

Together with the new construction rule, the class of functions is called *recursive*. These class of function seemed to contain any constructible total numerical function, such that it seemed reasonable to claim the following:

Model of Computation (Gödel-Kleene Characterization)

- An *algorithm* is a construction of a recursive function.
- A *computation* is a calculation of a value of a recursive function that proceeds according to the construction of the function.

[27] Weber, 2012, p. 51.
[28] Ibid., p. 52 f.
[29] Ibid., p. 54.

- A *computable* function is a recursive function.[30]

2.3.2 The Turing Machine

In 1936, 24 years old mathematician Alan Turing presented a model of computation. In his ground-breaking paper *On computable numbers, with an application to the Entscheidungsproblem*, he first introduced a rather simple mathematical formalism, now called *Turing Machine*, and then argued why this formalism fully comprises the intuitive concept of computability.[31]

While he was developing the concept of the Turing Machine, Turing was first guided by the way humans carry out calculations on paper and then abstracted from it. He concluded that such human calculators usually read and write symbols, look at different positions on the paper and hold some information in their mind. They apply a certain number of rules to that information, which is based on what they have seen and done before and what they are currently seeing. According to these simple elements of a human calculation process, he came up with the Turing Machine, a mathematical formalism, representing these elements.[32]

A Turing Machine is some kind of an idealized computer. It basically consists of two elements:

- A (potentially) infinite tape, which is divided into sequentially arranged squares. Each square can contain exactly one symbol of a predefined alphabet.
- A head, which reads from the tape and writes on the tape (possibly replacing text already present on the tape), moves back and forth on the tape and is controlled by a certain program.

Additionally, there exist a finite alphabet of symbols, a finite amount of internal states and a finite number of rules of calculation that are applied based on the current internal state and the symbol being read.[33]

[30] Robič, 2015, p. 74.
[31] Weber, 2012, p. 43 f.
[32] Ibid., p. 43 f.
[33] Ibid., p. 44.

A Turing machine can be formally described as follows:

A deterministic Turing machine is given by a $7-\text{Tupel } M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$.
Whereas

- Z expresses the finite set of states,
- Σ expresses the set of input symbols,
- $\Gamma \supset \Sigma$ expresses the set of tape alphabet symbols
- $\delta : Z \times \Gamma \rightarrow Z \times \Gamma \times \{L, R, N\}$ expresses the transition function,
- $z_0 \in Z$ expresses the initial state,
- $\square \in \Gamma$ expresses the blank symbol and
- $E \subseteq Z$ expresses the finite set of final states[34]

Due to Turing's very intuitive approach many scientists were convinced that the Turing Machine correctly represents what was known as the intuitive concept of computability. Up to the present day, the Turing Machine plays, especially due to its simplicity, a major role in computability theory. This leads to the following:

Model of Computation (Turing's Characterization):

- An *algorithm* is a Turing program.
- A *computation* is an execution of a Turing program on a Turing machine.
- A *computable* function is a Turing-computable function.[35]

2.4 The Development of the Church-Turing Thesis

After developing the lambda Calculus, now considered as another form of formal description of the algorithm, mathematician Alonzo Church was convinced that every computable function was computable using the lambda Calculus. This was called Church-Thesis. Church tried to convince Gödel, at that time considered as major scientific figure in formal logic, of

[34] Schöning, 2008, p. 73.
[35] Robič, 2015, p. 82.

his idea, but Gödel wasn't convinced. Later on, Church could prove that the lambda calculus is equivalent to the concept of general recursiveness, a formal description of the algorithm similar to the recursive functions, which were presented above. But again, Gödel was not convinced. Then Turing published his Thesis, claiming that every computable function was computable by a Turing machine. Now, especially due to the very intuitive approach Turing had pursued, Gödel was convinced. After a while, the two theses, the one of Church and the one of Turing, were put together to the Church-Turing Thesis.[36]

3 Epistemological State of the Church-Turing Thesis

Since Kleene had written about the Church-Turing *Thesis* in his influential book *Introduction to Metamathematics*[37], it has been widely discussed what the epistemological state of the Church-Turing Thesis actually could be. Epistemological state here simply means, what qualifies the Thesis to get broadly accepted as to be "true". Now, the different forms of epistemological state that have been proposed are briefly described.

Some scientists argue, that the Church-Turing Thesis is some form of (mathematical) definition. One of the first who did this was Church himself[38]. Today, one of the most distinguished proponents of regarding the thesis as a definition is mathematician Robert I. Soare, who claims that the Church-Turing Thesis should be regarded as a *definition of a computable function*[39]. Soare argues that there have been many theses in the past, which later became definitions. He refers, among others, to the mathematical definition of continuity. There, he argues that it took quite a while until everyone regarded the initial thesis, brought up by the mathematician Weierstraß, that the epsilon-delta criterion in fact fully captures the intuitive concept of continuity, as a definition of continuity. He claims that it simply might take some time until everyone accepts something as obvious as the Church-Turing Thesis as a definition for an intuitive notion.[40]

[36] Robič, 2015, p. 87 f.
[37] Soare, 2016, p. 236.
[38] Copeland, Shagrir, 2019, p. 67.
[39] Soare, 2016, p. 12.
[40] Ibid., p. 12.

Other scientists, such as Kleene argued that the Church-Turing Thesis is valid for practical reasons. First, so far and despite of various attempts, there has never been found a counter example that disproves the Church-Turing Thesis. Second, every characterization of computability, though different in its details proved to capture exactly the same class of functions.[41]

There are also scientists who asked for a mathematical proof of the Thesis. While some of them argue that the Thesis could be viewed as *subject ultimately to either mathematical proof or mathematical refutation like open mathematical conjectures, as in the Riemann hypothesis*[42] others argue that it is impossible to *pair an informal with a precise concept*[43]. And then there are still others, who tried to prove it mathematically. One of the most recent attempts to prove the Thesis was done by the philosopher and logician Paul Kripke. He argues that every computation is some kind of mathematical deduction. Since according to a thesis brought up by Hilbert, every mathematical deduction can be expressed as a valid deduction in the language of first-order predicate logic with identity, thus, the same is true for every computation. According to Gödel's completeness theorem[44], Kripke proceeds, every computation can be formalized by a provable formula of first order logic. And since Turing proved that every provable formula of first order logic can be proved by the universal Turing machine, every computation can be carried out by a universal Turing machine.[45]

To sum it up, it remains unclear what the actual epistemological state of the Church-Turing Thesis is. According to the author's opinion, the most convincing idea would be to regard the Church-Turing Thesis as a definition for a computable function.

[41] Copeland, Shagrir, 2019, p. 69 f.
[42] Ibid., p. 67.
[43] Ibid., p. 70.
[44] Gödel, 1929.
[45] Ibid., p. 70.

4 Quantum Computing and the Church-Turing Thesis

On September 20, 2019, the Financial Times reported on a paper by Google's researchers, where they claimed to have reached *quantum supremacy*.[46] *The term quantum supremacy […] refers to a computational task that can be efficiently performed on a quantum computer beyond the capabilities of state-of-the-art classical supercomputer can efficiently implement.*[47] According to the Financial Times, this was *a landmark moment that has been hotly anticipated by researchers*,[48] even though the problem that was used to prove quantum supremacy was a highly theoretical one, specially designed to show that quantum computers are in fact more powerful than classical ones.[49]

If the claims in the Google's researchers' paper are true, then Google wins a competition among many other tech companies and proves to be a first mover in what is seen by many computer scientists as one of the most powerful technologies of the 21th century: Quantum Computing.

The purpose of this chapter is to provide a brief introduction to quantum computing and to detect whether quantum computing and quantum computation might have effects on the Church-Turing Thesis. First, we will quickly present the history of quantum computing. Second, some basic notions of quantum computing are presented. Third, it is explained how quantum computation works. Fourth, a model quantum computing is presented. Fifth, a quantum algorithm is presented. Sixth, the possible effects of quantum computing on the Church-Turing Thesis are discussed. Seventh, some future prospects of quantum computing are presented.

4.1 Brief History of Quantum Computing

Using quantum mechanical phenomena such as entanglement and superposition as a very powerful way of doing computation has been discussed since the beginnings of quantum

[46] Murgia, Waters, 2019.
[47] Hidary, 2019, p. 190 f.
[48] Murgia, Waters, 2019.
[49] Calarco, 2019.

physics. The biggest challenge, however, has always been how one could construct a stable system that allows such form of computation.[50]

Paul Benioff was one of the first scientists who did serious research on quantum computing. In 1979, he published a paper called *The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines*. There, Benioff first provided the theoretical basis of quantum computing and then raised hope that a quantum computer could actually be build.[51]

In 1981, the famous physicist Richard Feynman held a lecture called *Simulating Physics with Computers* where he argued that in order to properly simulate quantum physics, the computer itself has to take advantage of quantum mechanical phenomena.[52]

Once Feynman, a world-famous scientist, turned the spotlight on quantum computing, a lot of researchers started tackling the problems of the new field of research. One of the major contributors at that time was David Deutsch, who in 1985 presented one of the first algorithms that would run faster on a quantum computer.[53] Following Deutsch, many other scientists developed quantum algorithms. The ones presented by Daniel Simon and Peter Shor are of particular interest.[54]

Especially the algorithm developed by Shor will most certainly yield enormous practical consequences. Shor's algorithm is able to factor large numbers into two prime factors. Since factoring large numbers is regarded as a hard problem for classical computers, it is at the core of the RSA algorithm, which is widely used in encrypting highly sensitive internet communications such as online banking, online payment and so on. Shor, however, showed that factoring large numbers could easily be done on a quantum computer and, thus, quantum computers will most certainly pose a major threat to common internet cryptography.[55]

[50] Hidary, 2019, p. 11.
[51] Hidary, 2019, p. 11.
[52]Ibid., p. 12.
[53] Ibid., p. 12.
[54] Ibid., p. 14.
[55] Ibid., p. 14.

While some researchers have been very successful in finding powerful quantum algorithms, others made significant progress in overcoming the technical barriers of building a quantum computer. In 2001, for example, Yasunobu Nakamura constructed a functioning, controllable superconducting qubit (the most basic computing unit of quantum computing).[56]

Over the years, lots of other remarkable results were made, culminating in the most recent triumph, where Google claimed it has finally achieved Quantum Supremacy.[57]

4.2 Basic Notions of Quantum Computing

The purpose of this chapter is to briefly describe some basic notions of quantum computing, that distinguishes quantum computers from classical computers. Contrary to a classical computer, *a quantum computer is a device that leverages specific properties described by quantum mechanics to perform computation.*[58] Every computer is based on quantum mechanics, since it is the fundamental theory of the physical world. What makes a quantum computer special, however, is that it is making use of some specific quantum mechanical properties to carry out computations. These phenomena are called *superposition, entanglement* and *reversibility* and will now be described.

One of the basic assumptions of quantum mechanics is that a system, such as an electron, for example, naturally is in an indeterminate state. Only if one measures the system, it will be set to a deterministic state. If one can define a set of discrete states a system can be in, for example the spin of an electron (which can be either horizontal or vertical, for example), then one can describe the indeterministic state of that system before measurement as a linear combination of these discrete states. For example, if there are to discrete states a system can be in, we can denote these states $|0\rangle$[59] and $|1\rangle$, respectively, and $\frac{1}{2} * |0\rangle + \frac{1}{2} * |1\rangle$ would be a linear combination of these two states. This linear combination, which describes the indeterministic state of a system before measurement, is called *superposition*

[56] Ibid., p. 15.
[57] Rincon, 2019.
[58] Hidary, 2019, p. 3.
[59] This notation is called Dirac notation and will be explained later.

of states.[60] If one has a linear combination of two states, such as $\alpha * |0\rangle + \beta * |1\rangle$, for example, α and β are complex numbers and are called *amplitudes*. These are very important, since it was demonstrated that *the modulus squared of the amplitude of a state is the probability of that state after measurement*.[61]

Another basic quantum mechanical principle of quantum computing is called *entanglement*. It was once considered to be one of the most counterintuitive and bizarre phenomena of quantum mechanics but is now regarded as one of the cornerstones of quantum mechanics. In quantum computing, entanglement allows novel types of computation. But what is entanglement anyway?[62]

The phenomenon of entanglement was first discovered by Einstein et al. in 1935. They were able to show that, under certain circumstances, if you have two disjunct particles, measuring the state of one of the particles will affect the state of the other one. This is also a way of storing information: The information is not stored in each of the two particles but instead in a correlation of these two. One can explain this while using a book metaphor. Imagine that there are two kinds of books, one is not entangled, the other one is entangled. The one which is not entangled can be read as usual, the information is stored on each page. The other one, however, is different: Each page contains gibberish, the information can only be extracted by analysing the correlation of the pages. This leads to the following definition: *Two systems are in a special case of quantum mechanical superposition called entanglement, if the measurement of one system is correlated with the state of the other system that is stronger than correlations in the classical world. In other words, the states of the two systems are not separable.*[63]

The last principle to discuss is called *reversibility*. Reversibility concerns a certain characteristic most operators used in quantum computation share: *All operators other than for measurement have to be reversible*.[64]

[60] Hidary, 2019, p. 3 f.
[61] Ibid., p. 4.
[62] Ibid., p. 6 f.
[63] Ibid., p. 6 f.
[64] Ibid., p. 9.

This is because the mathematical operations that formalize quantum computation also are reversible, or, more specifically, invertible. What does it mean for an operator to be reversible? Being reversible means that one is able to infer the input of an operator from observing the output. For example, consider the function $f: \{0,1\} \rightarrow \{0,1\}, f(x) = 1$. Here, f is not reversible, since it is not possible to state, whether the output 1 was produced by the input 1 or 0.

Luckily, the requirement for reversible operators does not mean a restriction for quantum computing: For other purposes, physicist Rolf Landauer carried out research on reversible classical computing processes in the 1980s and was able to define universal logical gates that are nevertheless reversible. These can also be adapted to quantum computers.[65]

4.3 Quantum Computation

In this section, the basics of a quantum computation are explained.[66] A quantum computation usually involves three steps: First, a quantum bit, or a quantum register, for that matter, is set to an initial state. Then, an operation, called unitary transformation, is applied to the qubit and the quantum register, respectively. Finally, the resulting state of it is measured.

To better explain the different components of such forms of computation, first, the basic unit of quantum computing, the qubit, is presented. It is shown, how a qubit differs from a classical bit and how it can be manipulated. Second, quantum registers are explained. Again, it is explained, how a quantum register differs from a classical register and how one can perform computations on it. Finally, it is explained, what measuring a qubit and a quantum register, respectively, means.

[65] Hidary, 2019, p. 7 f.
[66] The presentation of the theory is based on Homeister, 2015.

4.3.1 The Qubit

In classical computing, the basic unit of computation is the bit, which has either a value of 0 or 1. The basic unit of computation in quantum computing is the quantum bit, which is be defined below:

Definition (Quantum Bit): A quantum bit or qubit represents states of the following form: $\alpha * |0\rangle + \beta * |1\rangle$. α and β are called amplitudes and are complex numbers such that the following holds: $|\alpha|^2 + |\beta|^2 = 1$.

Here, some further explanation is needed. In quantum mechanics, it is common to use the so-called Dirac Notation, or ket notation for denoting quantum states. Thus, the state of a classical bit is denoted $|0\rangle$ and $|1\rangle$, respectively. In contrast to a classical bit, which has a single binary value, either $|0\rangle$ or $|1\rangle$, a qubit can exist in a so-called superposition of states and is described by $\alpha * |0\rangle + \beta * |1\rangle$, where α and β are satisfying $|\alpha|^2 + |\beta|^2 = 1$. If we want to know what state a qubit is actually in, we will have to measure it. After measurement, the qubit falls to one of the basis states, thus, the superposition is destroyed.[67] What state the qubit falls to depends on the amplitudes, α and β. More specifically, after measurement, the probability of outcome $|0\rangle$ is $|\alpha|^2$and the probability of outcome $|1\rangle$ is $|\beta|^2$.[68]

Example: Let's assume that the superposition of a certain qubit is given by $\frac{1}{\sqrt{2}} * |0\rangle + \frac{1}{\sqrt{2}} * |1\rangle$. Then, after measurement, the probability of outcome $|0\rangle$ is $\frac{1}{2}$ and the probability of outcome $|1\rangle$ is $\frac{1}{2}$.

The most important mathematical tool in quantum mechanics and, thus, in quantum computing, is linear algebra[69] and we will now start to use it.

The states of a qubit can be represented as vectors in a two-dimensional complex vector space. More specifically, the superposition $\alpha * |0\rangle + \beta * |1\rangle$ of any qubit can be written as

[67] Homeister, 2015, p. 20.
[68] Ibid., p. 20.
[69] Hidary, 2019, p. xii.

$\alpha * \begin{pmatrix}1\\0\end{pmatrix} + \beta * \begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}\alpha\\\beta\end{pmatrix}$. Thus, $\begin{pmatrix}\alpha\\\beta\end{pmatrix}$ is a linear combination of the two-dimensional unit vectors and, hence, the standard basis of the above-mentioned vector space consists of $\{|0\rangle, |1\rangle\}$. This means, the superposition of any qubit can be written as a linear combination of these two basis vectors.[70]

4.3.2 Manipulating Qubits

How do we manipulate a qubit, or, more specifically, how do we change the state of a qubit? In quantum computing, one uses unitary transformations to change the state of a qubit.[71] Performing a unitary transformation means that one multiplies the state vector of a qubit by a unitary matrix.[72]

Definition: Be A a complex $n \times n$ matrix. A is called $unitary$, if the following holds: $A^{\dagger} = A^{-1}$.

Example: The identity matrix of size 2, I_2 is unitary: $I_2^{\dagger} = I_2 = I_2^{-1}$.

In quantum computing, the unitary Matrix $H = \begin{pmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}\end{pmatrix} = \frac{1}{\sqrt{2}} * \begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}$, called Hadamard matrix, is of significant importance.

Example: Let's assume that the superposition of a certain qubit is given by $|0\rangle + |1\rangle$. If we want to change its state, we have to perform a unitary transformation. Here, we chose the Hadamard matrix. Now, we have to multiply the qubit's state vector by the matrix. This leads to $\begin{pmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}\end{pmatrix} * \begin{pmatrix}1\\1\end{pmatrix} = \begin{pmatrix}\frac{1}{\sqrt{2}}\\ -\frac{1}{\sqrt{2}}\end{pmatrix} = \frac{1}{\sqrt{2}} * \begin{pmatrix}1\\0\end{pmatrix} - \frac{1}{\sqrt{2}} * \begin{pmatrix}0\\1\end{pmatrix} = \frac{1}{\sqrt{2}} * (|0\rangle - |1\rangle)$. Now, the state of the qubit has changed. After measuring, the probability of both, outcome $|0\rangle$ and outcome $|1\rangle$ is $\left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2}$.

[70] Homeister, 2015, p. 22.
[71] Homeister, 2015, p. 22.
[72] Ibid., p. 23.

To demonstrate, what practical effects dealing with just a single qubit could have, now a simple algorithm is presented.[73] The algorithm produces random numbers. There are, of course, also algorithms, running on classical computers, that produce random numbers. But these numbers are just pseudorandom numbers. In contrary to these algorithms, the algorithm, which is described here, produces truly random numbers.

1. $|x\rangle \leftarrow |0\rangle$
2. $|x\rangle \leftarrow H|x\rangle$
3. Measuring $|x\rangle$

First, the quantum bit is initialized with the classical state $|0\rangle$. Then, a unitary transformation was performed, using the Hadamard matrix. Finally, the qubit is measured. Since the result of the measurement is truly random, the algorithm in fact produces truly random numbers.

To sum it up, in quantum computing a step of calculation is made when a quantum bit changes its state. To change its state, a unitary transformation is performed, which means that the state vector, which represents the superposition of the qubit is multiplied by a corresponding unitary matrix. To get the result of the computation, one has to measure the state of the qubit afterwards.

4.3.3 Quantum Register

So far, it was explained what a qubit is and how to change its state. Now, sequences of qubits are considered. A sequence of qubits is called *quantum register*. There are two questions of peculiar interest: What is the state of a quantum register? And how can one change the state of a quantum register?

To answer the first question, we start with a definition:

[73] The algorithm is described in Homeister, Matthias, Quantum Computing verstehen, p. 26 f.

Definition: Be R an n size quantum register. The state of R is represented by $R = \sum_{i=0}^{2n-1} \alpha_i |i\rangle$ such that the following holds: $\sum_{i=0}^{2n-1} |\alpha_i|^2 = 1$. If one measures R, the probability of the outcome $|i\rangle$ will be $|\alpha_i|^2$.

Example: Let's assume, $|x\rangle|y\rangle$ would be two-dimensional quantum register. Let's further assume that the state of $|x\rangle$ is given by $\gamma_1 * |0\rangle + \gamma_2 * |1\rangle$ and the state of $|y\rangle$ is given by $\beta_1 * |0\rangle + \beta_2 * |1\rangle$. Then, the following holds:

$$|x\rangle|y\rangle = (\gamma_1 * |0\rangle + \gamma_2 * |1\rangle) * (\beta_1 * |0\rangle + \beta_2 * |1\rangle) =$$
$$\gamma_1\beta_1 * |0\rangle|0\rangle + \gamma_1\beta_2 * |0\rangle|1\rangle + \gamma_2\beta_1 * |1\rangle|0\rangle + \gamma_2\beta_2 * |1\rangle|0\rangle =$$
$$\alpha_{00} * |00\rangle + \alpha_{01} * |01\rangle + \alpha_{10} * |10\rangle + \alpha_{10} * |11\rangle =$$
$$\alpha_0 * |0\rangle + \alpha_1 * |1\rangle + \alpha_2 * |2\rangle + \alpha_3 * |3\rangle.$$

So, in order to answer the first question: To examine the state of a quantum register, one has to multiply the states of its single qubits.

Now, the second question is answered. Just as the state of a single qubit, one can also represent the state of a quantum register by vectors. Here, the states of an n size quantum register can be regarded as vectors of a 2^n-dimensional vector space. The basis of that vector space is given by $\{|0 \dots 0\rangle, |0 \dots 1\rangle, \dots, |1 \dots 1\rangle\}$ which can be written more compactly as $\{|0\rangle, |1\rangle, \dots, |2^n - 1\rangle\}$.[74]

Example: Be $n = 2$. Then the basis of the vector space is given by $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ and can be represented as $|00\rangle = \begin{pmatrix}1\\0\\0\\0\end{pmatrix}, |01\rangle = \begin{pmatrix}0\\1\\0\\0\end{pmatrix}, |10\rangle = \begin{pmatrix}0\\0\\1\\0\end{pmatrix}, |11\rangle = \begin{pmatrix}0\\0\\0\\1\end{pmatrix}$.

Now, changing the state of a quantum register is quite similar to changing the state of a single qubit. Again, one has to perform a unitary transformation. In this case, however, to change the state of a quantum register of size n, the matrix, that represents the unitary transformation is a $2^{n\times n}$-dimensional matrix.[75]

[74] Homeister, 2015, p. 29.
[75] Ibid., p. 30.

Example: Let's assume that the superposition of a quantum register of size 2 is given by $\alpha_0 * |0\rangle + \alpha_1 * |1\rangle + \alpha_2 * |2\rangle + \alpha_3 * |3\rangle$. Let's further assume that a unitary transformation is given by the matrix $A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$. The register's superposition can be represented by the vector $v = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix}$. Then, the following holds: $A * v = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} * \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_2 \end{pmatrix}$.

So, to *basically* answer the second question: One changes the state of a quantum register by simply multiplying its state vector by a unitary matrix. For practical reasons, however, it could be very complex to perform a unitary transformation that is represented by a $2^{n \times n}$ matrix. Instead, it would be much simpler, changing the state of a quantum register bitwise. For that reason, one considers a special form of unitary transitions, called local unitary transformations.[76]

Definition: A unitary transformation is called local, if it affects not more than three single qubits.

The fact, that three qubits is an upper bound for calling a unitary transformation local, is due to practical reasons.[77] Actually, it has been proven, that it would have been enough to affect not more than two single qubits.[78]

Since it is, due to practical reasons, very important to decompose a unitary transformation into smaller, e.g. local transformations, the question arises how such a decomposition can be formalized. In order to do so, one has to use a mathematical structure, called tensor product. In the following section, the tensor product is briefly presented. The introduction, however, will be rather informal, since the tensor product is quite a complicated mathematical structure.

[76] Homeister, 2015, p. 31.
[77] There's a very useful quantum gate, called the Toffoli gate, which processes three qubits.
[78] Homeister, 2015, p. 31.

There are two reasons why the tensor product is useful: First, one can show how the state of a quantum register depends on the state of its qubits. Second, one can describe the single computation steps of a unitary transformation.

Let V and W be vector spaces. Let $e_0, \dots, e_{m-1}$ be a basis of V and $f_0, \dots, f_{n-1}$ be a basis of W. The tensor product of V and W is denoted by $V \otimes W$ and is itself an $m \times n$-dimensional vector space. The basic vectors of that vector space are denoted by

$$\begin{pmatrix} e_0 \otimes f_0 & \cdots & e_0 \otimes f_{n-1} \\ \vdots & \ddots & \vdots \\ e_{m-1} \otimes f_0 & \cdots & e_{m-1} \otimes f_{n-1} \end{pmatrix}.$$

$e_i \otimes f_j$ can be interpreted as follows: Let v be a vector in V and w be a vector in W. Then, $v = \sum_{i=0}^{m-1} \alpha_i e_i$ and $w = \sum_{j=0}^{n-1} \beta_j f_j$ and $v \otimes w = (\sum_{i=0}^{m-1} \alpha_i e_i) \otimes (\sum_{j=0}^{n-1} \beta_j f_j)$
$= \sum_{i}^{m-1} \sum_{j}^{n-1} \alpha_i \beta_j (e_0 \otimes f_0)$.

So, for example, for $n = 2$, the following holds: $\begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} \otimes \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} = \begin{pmatrix} \alpha_0\beta_0 \\ \alpha_0\beta_1 \\ \alpha_1\beta_0 \\ \alpha_1\beta_1 \end{pmatrix}$.

To show how the state of a quantum register depends on the state, one can consider the following example:

Let the superposition of two qubits $|\varphi\rangle$ and $|\phi\rangle$ be given by $|\varphi\rangle = \alpha_1 * |0\rangle + \alpha_2 * |1\rangle$ and $|\phi\rangle = \beta_1 * |0\rangle + \beta_2 * |1\rangle$, respectively. Then, the state of the quantum register $|\varphi\rangle|\phi\rangle$ can be expressed using the tensor product of $|\varphi\rangle$ and $|\phi\rangle$: $|\varphi\rangle \otimes |\phi\rangle = \alpha_1\beta_1 * |00\rangle + \alpha_1\beta_2 * |01\rangle + \alpha_2\beta_1 * |10\rangle + \alpha_2\beta_1 * |11\rangle$.

An additional reason to use the tensor product is that it allows to decompose a unitary transformation. This will now be further explained.

The tensor product can also be defined for matrices:

Let $A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$ and $B = \begin{pmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mn} \end{pmatrix}$ be Matrices. The tensor product of A and B is given by $A \otimes B = \begin{pmatrix} a_{11} * B & \cdots & a_{1n} * B \\ \vdots & \ddots & \vdots \\ a_{m1} * B & \cdots & a_{mn} * B \end{pmatrix}$.

Example: Let I_2 be the identity matrix of size n. Then $I_2 \otimes I_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

The following definition expresses the relation between the tensor product and unitary transformations:

Definition: Let $|x_1\rangle, \dots, |x_n\rangle$ be single quantum bits. Let $|x_1 \dots x_1\rangle$ be a quantum register of size n. The following two operations are equivalent:

1) Applying unitary transformations, represented by matrices $A_1, \dots, A_n$ to the single qubits (A_i to $|x_i\rangle$).
2) Applying the unitary transformation $A_1 \otimes \dots \otimes A_n$ to the quantum register.

Example: Let $|x\rangle|y\rangle$ be a quantum register of size 2. After applying the Hadamard transformation H to the first bit, one gets: $\frac{1}{\sqrt{2}} * (|0\rangle + |1\rangle)|1\rangle = \frac{1}{\sqrt{2}} * |01\rangle + \frac{1}{\sqrt{2}} * |11\rangle$. After applying the Hadamard transformation H to the second bit, one gets: $\frac{1}{\sqrt{2}} * |0\rangle \left(\frac{1}{\sqrt{2}} * (|0\rangle - |1\rangle)\right) + \frac{1}{\sqrt{2}} * |1\rangle \left(\frac{1}{\sqrt{2}} * (|0\rangle - |1\rangle)\right) = \frac{1}{\sqrt{2}} * (|00\rangle - |01\rangle + |10\rangle - |11\rangle) = H \otimes H$.

4.3.4 Measurement of Quantum registers

The last step of a quantum computation is measuring the state of the qubit and the quantum register, respectively, which was manipulated.

It has already been explained that if one wants to know what state a qubit is in, we will have to measure it. So, for example, if the superposition of a certain qubit is given by $\alpha * |0\rangle + \beta * |1\rangle$ then, after measurement, the probability of outcome $|0\rangle$ is $|\alpha|^2$and the probability of outcome $|1\rangle$ is $|\beta|^2$. The fact that there are the classical states involved, is, however, just a special case. One can also measure it in respect of other states. This requires choosing a different basis, though, and this basis is required to be orthogonal.[79]

Example: Let's first define the quantum states $|+\rangle = \frac{1}{\sqrt{2}} * (|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}} * (|0\rangle - |1\rangle)$. Let's further assume that the set $\{|+\rangle, |-\rangle\}$ defines a basis of a two-dimensional complex vector space. This basis obviously is orthogonal. So, assuming that the superposition of a qubit is given by $\gamma * |+\rangle + \delta * |-\rangle$, the probability of outcome $|+\rangle$ is $|y|^2$ and the probability of outcome $|-\rangle$ is $|\delta|^2$.

Of course, one can not only measure the state of a single qubit, but also the state of a quantum register. The following definition shows how this is done:

Definition: Let R be an n size quantum register. Let its state be $|\phi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle$. Then, a measurement is done, with respect to the orthonormal basis $|0'\rangle, |1'\rangle \dots, |(2^n - 1)'\rangle$. If $|\phi\rangle$, with respect to the that basis, can be represented as $\sum_{i=0}^{2^n-1} \alpha_i{}' |i'\rangle$, the probability that R has the state $|i'\rangle$ after measurement, is $|\alpha_i{}'|^2$.

4.4 A Model of Quantum Computing

There are a lot of different quantum computing models ranging from practically applicable ones such as quantum circuits to rather theoretical ones such as a quantum Turing machine introduced by Deutsch.[80] [81] However, each of these models have been proven to be equivalent to each other.

[79] Homeister, 2015, p. 44 f.
[80] Copeland, 2017.
[81] Homeister, Matthias, Quantum Computing verstehen, p. 123.

A very common and widely used model of quantum computing is based on quantum circuits. Quantum circuits are very similar to classical circuits, although the differ in a very important way. A classical circuit consists of wires and gates. Each wire represents a bit, whereas gates represent bit-manipulating operators. There is a set of very commonly used gates like the AND-, OR, and NOT-gate. A classical computer that is based on these gates is called Turing-complete or universal. In fact, it has been proven that there are some gates, for example the NAND-gate, a combination of the NOT- and the AND-gate that can simulate every other gate.[82]

Like classical circuits, quantum circuits consist of quantum wires and quantum gates. Each quantum wire represents a qubit and each quantum gate is performing a unitary transformation. A set of quantum gates builds a quantum circuit; thus, a quantum register is represented by a quantum circuit. Just as there is a set of classical gates that ensures Turing-completeness, there is also a set of quantum gates that is universal, although, unlike in classical computing, there doesn't exist a single gate such as the NAND-gate, for example, that provides Turing-completeness.[83]

Compared to classical circuits, quantum circuits are restricted in one way: They have to be reversible. A computation is reversible, if one can infer the input of the computation from simply regarding its output. This is because quantum gates perform unitary transformations. Since unitary transformations are represented by unitary matrices, which are invertible by definition, quantum gates have to be reversible.[84] Classical gates aren't restricted to reversibility. Consider the OR gate, for example. Both $0 \vee 1$ and $1 \vee 0$ result in 1. It's not possible to deduce the input from simply regarding the output.

A quantum's gate requirement for being reversible isn't a serious restriction, though: First, it can be shown that each reversible classical gate can be considered as a quantum gate. Second, it can also be shown that each classical gate can efficiently be rearranged to a reversible one.[85]

[82] Hidary, 2019, p. 30 f.
[83] Ibid., p. 31.
[84] Homeister, 2015, p. 85.
[85] Ibid., p. 87 ff.

4.5 Deutsch's Algorithm

To give an example, how quantum computing would practically work, we will discuss an algorithm developed by physicist David Deutsch. In 1985, Deutsch published a very influential paper on quantum computation. In his paper, Deutsch not only outlined theoretically ideas of quantum computation, such as a quantum Turing machine, but also presented a quantum algorithm, the first to show that a quantum computer could outclass a classical computer.[86]

First, the problem that the algorithm should solve will be stated. After that, the algorithm will be presented. Finally, each step of the algorithm will be explained.

Let's consider a function $f_i: \{0,1\} \rightarrow \{0,1\}$, for $i \in \{0, \dots, 3\}$. Let's further assume that the following holds:

- $f_0(0) = 0, f_0(0) = 0$
- $f_1(0) = 1, f_1(1) = 1$
- $f_2(0) = 0, f_2(1) = 1$
- $f_3(0) = 1, f_3(1) = 0$

f_0 and f_1 are called $constant$, f_2 and f_3 are called $balanced$.

Here's the problem: If one wants to know whether f_i is constant or balanced, a classical computer will have to check the outcome of both, $f_i(0)$ and $f_i(1)$. This is because, for example, if $f_i(0) = 1, i$ could either be equal to 1 or 3. A quantum computer, however, can solve the problem by checking f_i just once.

Before one can start, however, one has to adjust the function f, such that it is applicable to quantum computation. More specifically, since quantum computing uses unitary transformations, f has to be reversible. In order to a reversible version of f, one defines U_f: $|x, y\rangle \rightarrow |x, y \oplus f(x)\rangle$.

[86] Bernhardt, 2019, p. 145.

To solve the problem, the algorithm has to be carried out as follow:

1. A 2-bit-wide quantum register is initialized: $|x\rangle|y\rangle \leftarrow |0\rangle|1\rangle$
2. The Hadamard transformation is applied to each qubit: $|x\rangle|y\rangle \leftarrow H|x\rangle H|y\rangle$
3. f is carried out: $|x\rangle|y\rangle \leftarrow U_f|0\rangle|1\rangle$
4. Again, the Hadamard transformation is applied to each qubit: $|x\rangle|y\rangle \leftarrow H|x\rangle H|y\rangle$
5. The register is measured: If the value of $|x\rangle|y\rangle$ is equal to $|0\rangle|1\rangle$, f is $constant$. Otherwise, f is $balanced$.

Now, each step of the algorithm will be explained.

1. The quantum register is initialized with the basic states, $|0\rangle$ and $|1\rangle$, respectively.
2. Application of the bitwise Hadamard transformation: $H|x\rangle H|y\rangle = \frac{1}{\sqrt{2}} * (|0\rangle + |1\rangle) * \frac{1}{\sqrt{2}} * (|0\rangle - |1\rangle) = \frac{1}{2} * |0\rangle|0\rangle - \frac{1}{2} * |0\rangle|1\rangle + \frac{1}{2} * |1\rangle|0\rangle - \frac{1}{2} * |1\rangle|1\rangle$. As a result, one gets a superposition of the quantum register, which will be denoted by $|\phi_2\rangle$.
3. U_f is applied to ϕ_2. This results in $\frac{1}{2} * (|0\rangle|0 \oplus f(0)\rangle - |0\rangle|1 \oplus f(0)\rangle + |1\rangle|0 \oplus f(1)\rangle - |1\rangle|1 \oplus f(1)\rangle) = \frac{1}{2} * \big((|0\rangle * (|f(0)\rangle - |1 \oplus f(0)\rangle)) + |1\rangle * |f(1)\rangle - (|1 \oplus f(1)\rangle)\big)$. As a result, one gets a superposition of the function values of U_f, which will be denoted by$|\phi_3\rangle$. Additionally, one has to consider that the following equation holds: $|f(x)\rangle - |1 \oplus f(x) = (-1)^{f(x)}(|0\rangle - |1\rangle)$. Then, one can also write: $|\phi_3\rangle = \frac{1}{2} * ((-1)^{f(0)}|0\rangle * (|0\rangle - |1\rangle) + ((-1)^{f(1)}|1\rangle * (|0\rangle - |1\rangle)) = \frac{1}{2} * ((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle) * (|0\rangle - |1\rangle)$.
4. Now, two options are possible:
    a. Let's assume that f is constant. Then, the following holds: $(-1)^{f(0)} = (-1)^{f(1)}$ and, thus, the state of $|x\rangle$ is either $\frac{1}{\sqrt{2}} * (|0\rangle + |1\rangle)$ or $-\frac{1}{\sqrt{2}} * (|0\rangle + |1\rangle)$. Applying the Hadamard transformation to $\frac{1}{\sqrt{2}} * (|0\rangle + |1\rangle)$ and $-\frac{1}{\sqrt{2}} * (|0\rangle + |1\rangle)$, respectively, results in $|0\rangle$ and $-|0\rangle$, respectively. The state of $|y\rangle$, on the other hand, is $\frac{1}{\sqrt{2}} * (|0\rangle - |1\rangle)$. Applying the Hadamard transformation to $|y\rangle$ results in $|1\rangle$. Overall, the entire register now is in the following state: $|\phi_4{}^{constant}\rangle = \pm|0\rangle|1\rangle$.

b. Let's now assume that f is balanced. Then, the state of $|x\rangle$ is either $\frac{1}{\sqrt{2}} * (|0\rangle - |1\rangle)$ or $-\frac{1}{\sqrt{2}} * (|0\rangle - |1\rangle)$. Applying the Hadamard transformation to $|x\rangle$ results in $|1\rangle$. Applying the Hadamard transformation to $|y\rangle$ again results in $|1\rangle$. Overall, the entire register now is in the following state: $|\phi_4{}^{balanced}\rangle = \pm|0\rangle|1\rangle$.

5. Finally, after measurement, one gets the output $|1\rangle|1\rangle$ if f is balanced and $|0\rangle|1\rangle$ if f is constant. So, as a result, one had to measure only once, and the algorithm solved the problem.

4.6 Quantum Computing and the Church-Turing Thesis

One of the purposes of this chapter is to analyse if quantum computing poses a threat for the Church-Turing Thesis. More specifically, the question is whether there is a quantum computing model that is more powerful than a Turing machine. The answer is that there isn't one. As was explained, there are different models of quantum computing and they have been proven to be equivalent to each other. Since each of these models is universal, that is, Turing-complete, quantum computing doesn't challenge the Church-Turing Thesis. There are, however, many advantages, that quantum computers have compared to classical computer. Some of these advantages will be explained now.

4.7 Future Prospects

On the one hand, many scientists argue that quantum computing will only have very slight impacts on society in general. On the other hand, some scientists believe that quantum computing will similarly develop than classical computing did in the 20th century: At the beginning of the development of classical computers, people believed that, apart from some very special application, there won't ever be a broad usage of computers. And they have been proven seriously wrong by billions of smartphones being extensively used every day. So, the question whether quantum computing will have a serious impact on society remains unclear.[87]

[87] Bernhard, 2019, p. 171 f.

However, there are some important fields, where quantum computing will most certainly have a great impact on. For two of them, encryption and chemistry, this will be briefly explained.

RSA (Rivest-Shamir-Adleman) encryption is an algorithm-based encryption method that is widely used to encrypt data that is sent from one computer to the other. Two major applications of the RSA encryption method are Internet banking and electronic purchases using credit cards. The algorithm is, roughly speaking, based on prime factor decomposition. Factoring a number that is a product of two large prime numbers is seen as a hard problem for classical computers, since, *nobody has discovered a classical algorithm that can factor a product of two large primes in polynomial time*.[88] So, decrypting data which was secured by using the RSA encryption method would take an criminal eavesdropper much too long, if he used a classical computer. In 1994, however, Peter Shor presented the now so-called Shor's algorithm, which uses a quantum computer to factor a number that is a product of two large prime numbers much faster than a classical computer ever could. This could lead to a massive threat to the RSA based data encryption in general and, thus, to a very frequent way sensitive private data is secured at the moment.[89]

Chemistry could be another highly promising application of quantum computing. All of chemistry is based on quantum mechanical laws which can be mathematically described. The resulting mathematical equations, however, are far too complex to get exactly solved. So, in practice, chemists use approximations and, thus, sometimes ignore some very fine details. This has been working reasonably well, especially in computational chemistry, but there are circumstances, where these fine details do matter. Quantum computers could be used to exactly simulate the quantum mechanical basis of chemistry and, thus, lead to new insights. A practical example of how this could be useful was demonstrated by the University of Chicago. There, scientists are detecting the process of photosynthesize, a quantum mechanically based process. As soon as this process is properly analysed, the effects of photosynthesize could be used in photovoltaic cells, for example.[90]

[88] Bernhardt, 2019, p. 173 f.
[89] Ibid., p. 171 ff.
[90] Ibid., p. 181 f.

5 DNA Computing

On March 21, 2019, the scientific journal *Nature* published an article on *Demonstration of End-to-End automation of DNA Data Storage*[91]. There, a team led by scientists from the University of Washington and from Microsoft Research presented the first *fully automated end to end DNA data storage device*[92]. Using this device, the scientists were able to first store and then read the word *HELLO* while using DNA strands.[93] According to them, this would be a major step towards using DNA Data Storage across a broad front.[94]

Compared to the Google researchers' claim having achieved quantum supremacy (see chapter 4), this result may seem less landmarking. It is, however, strong evidence for the existence of another very vivid research approach concerning alternative computational models: DNA computing. The reason, why DNA-Computing appears so promising is because of some certain characteristics of DNA strands:

- DNA strands are very small and compact (It's possible to store 1 bit of information per cubic nanometre)
- DNA strands are, under certain conditions, very long-lasting, which means they are a very persistent storage media
- They can be easily duplicated a million times, which allows a redundant, decentralized and thus, very reliable form of data storage
- They are vectored, which allows modelling it through strings
- They can be, at least up to a certain point, repaired
- They can be manipulated by a broad spectrum of molecular biological processes
- They are easily recyclable and re-usable[95]

The purpose of this chapter is to provide a very brief introduction to DNA computing and to detect whether DNA computing could somehow affect the Church Turing Thesis. First, the

[91] Takashi, Bichlien, Strauss, Ceze, 2019.
[92] Ibid.
[93] Ibid.
[94] Ibid.
[95] Hinze, Sturm, 2004, p. 107.

history of DNA computing is quickly presented. Second, the foundations of DNA computing are briefly explained. Third, while discussing the Adleman experiment, a DNA computing algorithm is presented. Fourth, while briefly presenting DNA computing models, it is discussed whether DNA computing can affect the Church Turing Thesis. Fifth, some future prospects of DNA computing are provided.

5.1 History of DNA Computing

How nature encodes, stores and shares (genetic) information has puzzled philosophers and scientists for many thousand years. It was not until the twentieth century that researchers succeed in finding an answer: The introduction of the double-helix model by James D. Watson and Francis H. Crick in 1953[96] revealed the spatial structure of DNA. This model has initiated and still ongoing progress in molecular biology. Around the time when Watson and Crick were investigating the spatial structure of DNA, other scientists began to reflect on using DNA to carry out computations. In 1959 Richard Feynman, the famous Nobel Prize-winning physicist, held an influential talk suggesting to process data through of biological molecules.[97]

The beginning of the development of DNA-Computing was firstly initiated in 1985 by a rather theoretical paper by Bennet and Landauer on fundamental physical limits of computation[98], and later in 1994 practically through an experiment conducted by Leonhard Adleman and now called the Adleman Experiment. Especially the latter made great stir, since the Adleman Experiment solved, for the first time ever, a NP-complete problem in polynomial time. However, the Adleman Experiement merely shifts the complexity from time to space since it requires exponential space). The Adleman Experiment was the first evidence that DNA computing is actually feasible.[99]

Since Adleman has published his results, DNA computing has remained an active and fruitful area of research. Many scientists are now trying to build DNA computers that, in principle, should be Turing complete.[100]

[96] Hinze, Sturm, 2004, p. 15.
[97] Ibid., p.
[98] Hidary, 2019, p. 8.
[99] Ibid., p. 8 f.
[100] Ibid., p. 13.

5.2 Foundations of DNA Computing

DNA (deoxyribonucleic acid)[101] provides the genetic information that is needed for the development and proper functioning of every living organism. The 3-dimensional structure of DNA consists of two helical strands that are coiled around an axis building a double helix which is crucial for the process of heredity, that is, passing genetic information from one generation to the next.[102]

First and foremost, DNA serves as an information storage device. From a chemical perspective, DNA is made of two large polymers. Each polymer consists of several basic units, called nucleotides. Each nucleotide consists of a sugar, called deoxyribose, a phosphate and one of four different bases: Adenine (A), Thymine (T), Guanine (G) and Cytosine (C). A sequence of these four bases encodes information based on an alphabet, called genetic code. The information stored in a DNA strand is decrypted by reading this particular code.[103]

A nucleotide's base can be attached to another nucleotide's base. Such pairs of bases, however, cannot be built arbitrarily. Instead, only the following pairs are possible: Adenine can only be attached to Thymine and vice versa; Guanine can only be attached to Cytosine and vice versa. This means, one can only connect two different DNA strands with each other, if they are complementary.[104]

Example: If one DNA strand consist of the following bases: *AACTGCA*, its complementary strand would be *TTGACGT*.

Two complementary single-stranded DNA can form a double-stranded DNA, eventually building a double helix structure.[105]

[101] El-Seoud, Ghoniemy, 2017, p. 75.
[102] Ibid., p. 75.
[103] Ibid., p. 75.
[104] Ibid., p. 76.
[105] Ibid., p. 76.

One can now use a DNA strand's sequence of bases to encode information. Similar to common silicon-based computing devices, where information is encoded by using an alphabet which consist of 0 and *1*, one can now use an alphabet which consists of *A*, *C*, *G* and *T*, representing the four different DNA bases.[106]

Additionally, to perform actual computations, one needs ways to manipulate the strings of DNA bases. Fortunately, there exist a lot of molecular biological processes that allow manipulating these strings. These processes and their effects, respectively, are now briefly described.

- Synthesizing: One can synthesize a single- or double stranded-DNA of polynomial length.
- Mixing: Two test tubes, each containing DNA strands, can be combined into a third one.
- Annealing: Two single-stranded DNA sequences can be linked to each other.
- Melting: A double-stranded DNA sequence can be separated into two single-stranded DNA sequences.
- Amplifying: By using a biomolecular process called polymerase chain reaction, one can make copies of DNA strands.
- Separating: By using a technique, called gel electrophoresis, a single-stranded DNA sequence can be divided into smaller pieces, each containing a subset of the total amount of nucleotides of the sequence.
- Extracting: After separating, a certain subsequence of nucleotides can be extracted.
- Cutting: By using a certain class of enzymes, called restrictions endonucleases, one is able to detect a certain short sequence of DNA. Any double-stranded DNA containing this sequence will be cut at that location.
- Ligating: One can paste DNA strands with compatible sticky ends.
- Substituting: One can substitute, insert or delete DNA sequences.
- Marking: Allows to combine two complementary single-stranded DNA sequences to one double-stranded DNA sequence.

[106] Ibid., p. 76.

- Destroying: Allows to destroy the combination of two single-stranded DNA sequences.
- Detecting and Reading: Allows to first detect the contents of a tube and then read the actual solution.[107]

These operations allow to actually manipulate the sequences of DNA bases. At the beginning of such a DNA computing process, the DNA sequences are kept in a test tube. A test tube can contain any finite amount of DNA strands. An DNA computing operation, that is, a certain biomolecular process, is then applied to each of the DNA strands simultaneously. After the application, the content of the test tube is modified. DNA computing operations are applied by changing certain parameters, such as pH number or temperature.[108]

5.3 The Adleman Experiment

To explain how computing with DNA could actually work, the Adleman Experiment is briefly described. In 1994, computer scientist Leonard Adleman published a landmarking paper where he described how to solve an instance of an NP-complete problem in polynomial time while using DNA computing.[109] More specifically, Adleman was aiming to solve the Directed Hamiltonian Path problem (HP), a variant of the 'travelling salesman problem' (TSP).[110] The TSP asks the following question: *Given n cities, find a path visiting each and every city only once starting and ending at given point.*[111] For N cities, there are $(N-1)!/2$ possible paths.[112] For ordinary computers, this problem is hard to solve, that is, it is an example of an NP problem.[113] Adleman was trying to solve a certain instance of HP $(N=7)$.

The DNA computing algorithm Adleman used carries out four steps: First, all possible routes are generated. Second, the paths that start with the proper city and end with the final city, are selected. Third, the paths with the correct number of cities are selected. Fourth, the

[107] Abbaszadeh Sori, 2014, p. 229 f.
[108] Hinze, Sturm, 2004, p. 108.
[109] Hinze, Sturm, 2004, p. 8.
[110] El-Seoud, Ghoniemy, 2017, p. 77.
[111] Ibid., p. 77.
[112] Ibid., p. 77.
[113] More specifically, TSP is even an NP-complete problem.

paths that contain each city only once, are selected.[114] These four steps are now described. The description follows the one in El-Seoud, Ghoniemy, 2017.

1. **Generate all possible routes:** *Encode city names in short DNA sequences, and then encode the paths by connecting the city sequences for which routes exist. DNA can simply be treated as a string of data. For example, for the case when number of cities* $N = 5$*, each city can be represented by a string of six bases as follows:* $L - GCTACG\ C - CTAGTA\ D - TCGTAC\ M - CTACGG\ N - ATGCCG$

   The entire path can be encoded by simply stringing together these DNA sequences that represent specific cities. For example, the route $L - C - D - M - N$ *would simply be* $GCTACGCTAGTATCGTACCTACGGATGCCG$*, or equivalently it could be represented in double stranded form with its complement sequence. Synthesizing short single stranded DNA is now a routine process, so encoding the city names is straightforward. The molecules can be made by a machine called a DNA synthesizer. Paths can then be produced from the city encodings by linking them together in proper order. To accomplish this, we can take advantage of the fact that DNA hybridizes with its complimentary sequence. For example, you can encode the routes between cities by encoding the compliments of the second half (last three letters) of the departure city and the first half (first three letters) of the arrival city. For example the route between* $M\ (CTACGG)$ *and* $N\ (ATGCCG)$ *can be made by taking the second half of the coding for* $M\ (CGG)$ *and the first half of the coding for* $N\ (ATG)$*. This gives* $CGGATG$*. By taking the complement of this you get,* $GCCTAC$*, which not only uniquely represents the route from* M *to* N*, but will connect the DNA representing* M *and* N *by hybridizing itself to the second half of the code representing* $M\ (\ldots CGG)$ *and the first half of the code representing* $N\ (ATG\ldots)$*.*

   Random paths can be made by mixing city encodings with the route encodings. Finally, the DNA strands can be connected together by an enzyme called ligase. What

[114] El-Seoud, Ghoniemy, 2017, p. 79.

we are left with are strands of DNA representing paths with a random number of cities and random set of routes.

Using an excess of DNA encodings, we can be confident that we have all possible combinations routes between cities including the correct one.

2. **Select paths that start with the proper city and end with the final city:** *Selectively copy and amplify only the section of the DNA that starts with L and ends with N by using the Polymerase Chain Reaction. To accomplish this, we can use technique called Polymerase Chain Reaction (PCR), which allows us to produce many copies of a specific sequence of DNA. PCR is an iterative process that cycles through a series of copying events using an enzyme called polymerase. Polymerase will copy a section of single stranded DNA starting at the position of a primer. A primer is a short piece of DNA complimentary to one end of a section of the DNA that we are interested in. By selecting primers at the start and end of the section of DNA we want to amplify, the polymerase using this as a template for replication, amplifies the DNA between these primers, doubling the amount of DNA containing this sequence. After many paths of PCR, the DNA we are working on is amplified exponentially. So, to amplify the paths that start and stop with our cities of interest, L and N, we use primers that are complimentary to L and N. We then end up with a test tube full of double stranded DNA of various lengths (i.e. with various number of cities), encoding only paths that start with L and end with N.*
3. **Select the paths with the correct number of cities:** *Sort the DNA by length and select the DNA whose length corresponds to 5 cities. To accomplish this Adleman used a technique called Gel Electrophoresis(GE), which is a common procedure used to resolve the size of DNA. The basic principle behind Gel Electrophoresis is to force DNA through a gel matrix by using an electric field. DNA is a negatively charged molecule under most conditions, so if placed in an electric field it will be attracted to the positive potential. Since the charge density of DNA is constant (charge per length) long pieces of DNA move as fast as short pieces when suspended in a fluid. The gel is made up of a polymer that forms a meshwork of linked strands. As the DNA is forced to thread its way through the tiny spaces between these strands, it slows down at different rates depending on its size i.e. length. This means that after running a gel*

we end up with a series of DNA bands, with each band corresponding to a certain length. The band of interest can then simply cut out to isolate DNA of a specific length. In this case the DNA that was 30 *base pairs long (*5 *cities with each city encoded with* 6 *base pairs of DNA) would be isolated. In the end the test tube will contain all paths that each starts at L and ends at N, and which have a total of exactly* 5 *cities encoded.*

4. **Select the paths that contain each city only once:** *Successively filter the DNA molecules by city, one city at a time. Since the DNA we start with contains five cities, we will be left with strands that encode each city once.*

   DNA containing a specific sequence can be purified from a sample of mixed DNA by a technique called affinity purification. This is accomplished by attaching the compliment of the sequence in question to a substance like a magnetic bead. The beads are then mixed with the DNA. DNA, which contains the sequence we are after then hybridizes with the complement sequence on the beads. These beads can then be retrieved and the DNA isolated.

   So, we now affinity purify five times, using a different city complement for each run. For example, for the first run we use L'-beads (the complement of L) to filter out DNA sequences which contain the encoding for L. (which should be all the DNA because of step 3*). The next run we use D'-beads, and then C'-beads, M'-beads, and finally N'-beads. The order isn't important. If a path is missing a city, then it will not be filtered out during one of the runs and will be removed from the candidate pool. What we are left with are the paths that start in L, visit each city once, and end in N. This is exactly what we are looking for. If the answer exists, it would be retrieved at this step.*[115]

To sum it up, Adleman, while referring only to an instance of HP, hasn't solved the problem in general. Moreover, the complexity of the problem hasn't simply disappeared while using DNA computing techniques instead of ordinary computing. With Adlemans method, the reduced time complexity comes at a price: Now, the amount of DNA that would be needed to solve the problem, increases exponentially. One guesses, that, for $N = 200$, the amount

[115] El-Seoud, Ghoniemy, 2017, p. 79 ff.

of DNA strands required would be overwhelming.[116] But despite of these restrictions, Adleman nevertheless achieved a remarkable result: For the first time, he showed that DNA computing was possible, while making it obvious what huge potential in terms of data storage and computational speed could be achieved by using DNA as a data structure.[117]

5.4 DNA Computing Models and the Church-Turing Thesis

There are many different DNA computing models, such as filtering models, insertion-deletion systems, splicing systems and even a programming language, called DNA Pascal. Each of these models were designed to be a model of a universal DNA computer, that can be used to efficiently solve computational problems for scientific or economic purposes. In doing so, the model should be as consistent as possible with the commonly used biomolecular processes to manipulate DNA sequences. However, until now, the different models failed to do so, mostly due to their level of abstractness, which can't be implemented by dealing with commonly used biomolecular processes.[118]

Describing these models in detail would go beyond the scope of this bachelor thesis. Instead, it is sufficient to notice that each of the different models was designed to be computationally universal, thus Turing complete. This leads to the following result: Since every model claiming to properly represent the entire amount of DNA computing processes is itself Turing complete, DNA computing does not affect the Church-Turing Thesis.

5.5 Future Prospects

When it comes to the possible future outcomes of DNA computing, one of the most important questions is, whether one can eventually overcome the enormous technical obstacles of the practical realization of DNA computing. Such obstacles concern, for example, being able to effectively control the effects of the great amount of side effects that usually appear while using biochemical operations.[119] It could either be that a *precise enough characterization of DNA chemistry and physics is not possible for building a DNA*

[116] Ibid., p. 82.
[117] Ibid., p. 77 f.
[118] Hinze, Sturm, 2004, p. 210.
[119] Ibid., p. 9.

computer[120] or that *continued exploration of current* […] *techniques may eventually result in a new body of methods that are specifically adapted to computing with DNA*[121].

Another problem prohibiting the research field of DNA computing from making progress similar to the field of quantum computing is the lack of promising applications of DNA computing. Although it is guessed that DNA computing could by successfully applied to problems in molecular biology, a so-called '*killer app*'[122] hasn't been found yet.

One possible application of DNA computing could result in a form of *hybrid machines*[123]. That is, one could connect common silicon-based computers with a DNA computer. The silicon ones would be used to carry out ordinary computations while the DNA processor would be applied to particular problems where its specific advantages could be deployed.[124]

6 Summary and Conclusion

This report has first recalled the history of the Church-Turing Thesis. The main result here is that the thesis was, in some sense, a side-effect of various attempts to solve the Entscheidungsproblem: In order to solve this problem, different formal models were developed to formalise the intuitive concept of the algorithm, or, more specifically, the concept of an effectively calculable function. Among other reasons, the equivalence of these different models resulted in the Church-Turing Thesis.

Second, the epistemological state of the Church-Turing Thesis was investigated. The main result here is that there exist different opinions whether the thesis is mathematically provable or not, whether it is considered true for practical reasons or whether it follows from the definition of the concept of the algorithm.

[120] Abbaszadeh Sori, 2014, p. 231.
[121] Ibid., p. 231.
[122] Abbaszadeh Sori, 2014, p. 231.
[123] Ibid., p. 231.
[124] Ibid., p. 231.

Third, new technologies, that is, quantum computing and DNA computing, were presented. The main result here is that neither quantum computing nor DNA computing are posing a threat to the Church-Turing Thesis. This is because of the models of computation that are currently used to describe the functional principles of quantum computing and DNA computing, respectively. Each of these models can be successfully simulated by a Turing machine, thus, the class of functions computable by quantum computers and DNA computers, respectively, is Turing-complete.

To sum it up, the current state of the Church-Turing Thesis is as follows: Even though it remains divisive, *why* the Church-Turing Thesis is true, there are no doubts that it is actually true. More particularly, even newest computing technologies do not affect this.

One could, however, ask whether there might be some future technologies, not yet developed or even not yet thought about, that could challenge the Church-Turing Thesis. There is, for example, the purely theoretical idea of hypercomputing: *Hypercomputers compute (in a broad sense of 'compute') functions or numbers – or, more generally, solve problems or carry out information processing tasks – that lie beyond the reach of the universal Turing machine.*[125] But this would not affect the *original* Church-Turing Thesis. The thesis claims that every effective computation can be carried out by a Turing machine. The sticking point here concerns the notion of *effective computation*. A computation is called *effective*, if some very specific conditions hold (for example the computation requires no insight or intuition or ingenuity of the human who is performing the computation). The Church-Turing Thesis simply states that the class of functions, that fulfil the effectiveness-requirement, is Turing-complete, that is, each of these functions can be carried out by a Turing machine. It does not, however, consider something that is called *Maximality thesis*[126]: *All functions that can be generated by machines (working in accordance with a finite program of instructions) are computable by effective methods.*[127] Whether or not *this* thesis is valid, is *an open empirical question*[128], but it is a different thesis compared to the initial Church-Turing Thesis that was concerned in this bachelor thesis. So, if only the "original"

[125] Copeland, 2017.
[126] Ibid.
[127] Ibid.
[128] Ibid.

thesis is taken into account, even future technologies will not harm the Church-Turing Thesis.

7 References

1. Bernhardt, Chris. (2019). *Quantum Computing for Everyone*. Cambridge. United States of America.

2. Copeland, Jack, Shagrir, Oron. (2019). *The Church Churing Thesis: Logical Limit or Breachable Barrier?*. Last downloaded 2019-11-11. http://delivery.acm.org/10.1145/3200000/3198448/p66-copeland.pdf?ip=79.219.253.206&id=3198448&acc=OA&key=4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E8BE65BB018D793D0&__acm__=1573473041_9e86fe7808a32bfa96ad03ae7d160cba

3. Copeland, Jack. (2017). *The Church Turing Thesis*. Last downloaded 2019-11-11. https://plato.stanford.edu/entries/church-turing/

4. El-Seour, Samir A., Ghoniemy, Samy. (2017). *DNA Computing: Challenges and Application*. International Journal of interactive Mobile Technologies. April 2017. Also available via https://online-journals.org/index.php/i-jim/article/view/6564 Last downloaded 2019-11-11.

5. Gödel, Kurt. *Über formal unentscheidbare Sätze der "Principia Mathematica" und verwandter Systeme I.* Monatsheft für Mathematik und Physik, volume 38, pages 173-198, 1931. In German, reprinted an English translation.

6. Gödel, Kurt. *Kurt Gödel - Collected Works, volume I - Publications 1929-1936.* Oxford University Press, Oxford, UK, and New York, USA, 1986.

7. Gödel, Kurt. (1929). *Über die Vollständigkeit des Logikkalküls. Doctoral dissertation.* University of Vienna.

8. Gödel, Kurt. (1986) - *Collected Works. I: Publications 1929–1936.* S. Feferman, S. Kleene, G. Moore, R. Solovay, and J. van Heijenoort (editors), Oxford: Oxford University Press.

9. Hidary, Jack D. (2019). *Quantum Computing: An Applied Approach*. Cham. Switzerland.

10. Homeister, Matthias. (2015). *Quantum Computing verstehen*. Wiesbaden. Germany.

11. Petzold, Charles. (2008). *The Annotated Turing*. Indianapolis. United States of America.

12. Hinze, Thomas, Sturm, Monika. (2004). *Rechnen mit DNA: Eine Einführung in Theorie und Praxis*. München. Deutschland.

13. Murgia, Madhumita, Waters, Richard. (2019, 20th of September). *Google claims to have reached quantum supremacy*. Last downloaded 2019-11-11. https://www.ft.com/content/b9bb4e54-dbc1-11e9-8f9b-77216ebe1f17

14. Rincon, Paul. (2019). *Google claims 'quantum supremacy' for computer*. Last downloaded 2019-11-11. https://www.bbc.com/news/science-environment-50154993

15. Robič, Borut. (2015). *The Foundations of Computability Theory*. Heidelberg. Germany.

16. Schöning, Uwe. (2014). *Theoretische Informatik – kurz gefasst*. Heidelberg. Germany.

17. Soare, Robert I. (2016). *Turing Computability*. Heidelberg. Germany.

18. Sorir, Amir Abbaszadeh. (2014). DNA Computer; Present and Future. *International Journal of Engineering Research and Applications, 6(2)*. Also available via https://core.ac.uk/download/pdf/26663028.pdf Last downloaded 2019-11-11.

19. Takashi, Christopher N., Nguyen, Bichlien H., Strauss Karin, Ceze Louis. (2019). *Demonstration of End-to-End Automation of DNA Data Storage*. Last downloaded 2019-11-11.
20. https://www.nature.com/articles/s41598-019-41228-8

21. Weber, Rebecca. (2012). *Computability Theory*. United States of America.

8 Statement of Originality

I hereby confirm that I have written the accompanying thesis by myself, without contributions from any sources other than those cited in the text and acknowledgements. This applies also to all graphics, drawings, maps and images included in the thesis.

_____ _____

Place and date Signature