Universiteit Utrecht

FACULTY OF SCIENCE

# On Quantum Computing and Semi-Thue Systems

**Final Paper**

Course “Models of Computation”

| | |
|---|---|
| Handed in | 15/11/2011 |
| Teacher | Prof. Dr. V. van Oostrom, Dr. C. Grabmeyer |
| Student | B.F. Maier (F110163) |

# Contents

# 1 Introduction

In the 1930s, a new branch of mathematics developed, the so-called computability theory. This field of mathematics deals, besides other applications, with the question of deciding which problems are effectively calculable, meaning to decide for which problems there exists an effective procedure to solve them (say, an algorithm). Generalizing the notion "problem" and "answer", one could also formulate this to find a function which maps from one natural number (problem) to another natural number (answer). To this end, two very different yet equivalent models arose, the $\lambda$-calculus by A. Church and the notion of Turing machines by A. Turing, with which could be shown that there are decision problems for which there are no general procedures to decide whether they are solvable or not. These models are so-called models of computation.

In this paper, I will introduce two other different models of computation and compare them, besides others, to Turing machines, leaving the $\lambda$-calculus aside. Thus, the reader is expected to have a general idea of what a Turing machine is and how it operates. As an introduction for the non-prepared reader, I recommend to read [1].

The first model to be introduced is the one of semi-Thue systems, which denotes a general form of string rewriting systems. They will be explained and compared to other classical models and their origin will be shown.

Second, I will give a short presentation of quantum computing as a model of computation. It is connected to the notion of reversible Turing machines. Quantum computing may have properties superior to those of classical models, which will be indicated.

In the last chapter, both models will be connected to give indications that they are equivalent, while there will certain obstacles arise which have to be overcome.

# 2 Semi-Thue Systems and Word Problems

Semi-Thue systems denote a model of computation that is based on string rewriting. First described in 1914 by A. Thue [2], he stated certain problems for these string rewriting systems, which later have been proven to be undecidable [3], first by reduction to the halting problem of Turing/Post machines [3, 4, 5].

In the following section, I will introduce the model formally, give examples, formulate some of Thue's word problems and give indications on the equivalence of semi-Thue systems and Turing/Post machines.

## 2.1 Definitions

A semi-Thue system is a 2-tupel $(\Sigma, S)$ with

- $\Sigma$ being a finite alphabet,
- $\Sigma^*$ being the set of finite-length words over $\Sigma$ (with $^*$ being the Kleene star),
- $S \subseteq \Sigma^* \times \Sigma^*$ being a set of pairs of words in $\Sigma^*$, called the rewriting rules.

Note that the empty word $\varepsilon$ is part of the set $\Sigma^*$. To give an example of concatenations, one concatenation of two words $x, y \in \Sigma^*$ would be $xy$, the second possible one is $yx$. One rewriting step is denoted by the following procedure. Given a pair $(u, v) \in S$ and a word $xuy$, we denote one rewriting step as

$$xuy \longrightarrow_S xvy. \tag{1}$$

Finding a substring in the word for rewriting is done *non-deterministically*, meaning if there is more than one possibility to apply rules from $S$ on a word, there is no preference on which rule to apply where first. If there is a chain of words $w_1, w_2, \ldots, w_n \in \Sigma^*$, s.t.

$$w_1 \longrightarrow_S w_2 \longrightarrow_S \ldots \longrightarrow_S w_n, \tag{2}$$

we can shorten the derivation to

$$w_1 \overset{*}{\longrightarrow}_S w_n, \tag{3}$$

and say that $w_n$ is *derivable* from $w_1$ in $S$. The term *semi-Thue systems* relates to the first introduced *Thue systems*, in which the rules were symmetrical, meaning that a pair $(u, v)$ denotes possible replacing of $u$ by $v$, as well as replacing $v$ by $u$. However, since semi-Thue systems are more general than Thue-systems, I will consider only them.

## 2.2 Examples

Take a semi-Thue system $(\Sigma, S)$ defined by the alphabet $\Sigma = \{a, b, c\}$ and the rules $S = \{(ab, c), (cc, bba), (ac, ca)\}$. Given the initial word *abccc*, we have three possibilities to apply rules from $S$

$$1) \quad \underline{ab}ccc \quad \text{with the rule } (ab, c) \tag{4}$$

$$2) \quad ab\underline{cc}c \quad \text{with the rule } (cc, bba) \tag{5}$$

$$3) \quad abc\underline{cc} \quad \text{with the rule } (cc, bba) \tag{6}$$

Deciding for the second possibility, one obtains the next word *abbbac*. Now, we can apply the rule $(ab, c)$ and afterwards the rule $(ac, ca)$, yielding the word *cbbca* that cannot be altered anymore. The derivation can be written as

$$abccc \longrightarrow_S abbbac \longrightarrow_S cbbac \longrightarrow_S cbbca, \tag{7}$$

or

$$abccc \longrightarrow_S abbbac \longrightarrow_S abbbca \longrightarrow_S cbbca, \tag{8}$$

or

$$abccc \overset{*}{\longrightarrow}_S cbbca. \tag{9}$$

Thus, *cbbca* is derivable from *abccc* in $S$.

A second example denotes addition in a unitary representation of the natural numbers. First, consider the alphabet $\Sigma = \{|, +, \}$, where a word of $n$ concatenated $|$'s stands for the natural number $n$. Addition can now be defined by the rewriting rule $S = \{(| + |, ||)\}$. That means for example, given a string

$$||| + || + ||||, \tag{10}$$

which represents $3 + 2 + 4$, applying the rule finally yields $|||||||||$, which is the result 9. Note that invalid expressions as initial words like $+||$, $|||+$ or $| + +||$ will not be processed.

## 2.3 Connection to Other Classical Models

Semi-Thue systems are equivalent to all other classical models of computation in the way that every decision or computability problem formulated in another class can be reduced, i.e. reformulated, to a semi-Thue system and vice versa [6]. In this subsection I will give both formal and informal indications on connections to different models.

### 2.3.1 String Rewriting Systems

**Unrestricted grammars** are formal grammars and rewriting systems, which often use a distinction between terminal and non-terminal symbols (not necessarily), meaning that all the rules are of a form $(v, w)$, $v, w$ being words made of terminal and non-terminal symbols with $v$ not being the empty string. Furthermore, they usually start with a single starting symbol and yield words which are made of terminal symbols only. These grammars are connected to all recursively enumerable languages (languages that can be recognized by a Turing machine).

The difference to normal semi-Thue systems is the usual distinction between terminals and non-terminals, as well as the use of a starting symbol. A similarity is the use of non-determinism to apply rules.

**Lindenmayer systems** (or L-systems) were first introduced by A. Lindenmayer and represent parallel rewriting. With an alphabet $\Sigma_L$, rewriting rules $S_L$ and an initial word $w \in \Sigma_L^*$ already included in the system, all rules of a classical L-system have only one letter at the left-hand side and all rules have to be applied at once. Furthermore, there is at most one rule for every letter. If there is no rule for a letter, the letter is set to be constant. To simulate an L-system using a semi-Thue system, we can use two head symbols $H$ and $C$, which go through the current word and change letters according to the rules of the L-system. Suppose an L-System with $\Sigma_L^*$ of arbitrary finite size, rules $S_L$ and the initial word $w$. We now construct a semi-Thue system with $\Sigma_S = \Sigma_L \cup \{H, C, \#\}$. The initial word is the following concatenation.

$$H\#w\# \tag{11}$$

For every letter in $\Sigma_L$ we now have to introduce a rule which rewrites the letter and shifts the head to the next letter. The starting rule is

$$(H\#\sigma_i, \#H\sigma_i) \tag{12}$$

with $\sigma_i \in \Sigma_L$. Every rule for a constant letter $c_i \in \Sigma_L$ is

$$(Hc_i, c_iH) \tag{13}$$

Every other rule of the kind $(v_i, u)$ with $v_i \in \Sigma_L$ and $u \in \Sigma_L^*$ reads

$$(Hv_i, uH) \tag{14}$$

In the end we have to "rewind" the head to its initial position, which is done via

$$(\sigma_i H\#, \sigma_i C\#) \tag{15}$$
$$(\sigma_i C, C\sigma_i) \tag{16}$$
$$(\#C\sigma_i, H\#\sigma_i) \tag{17}$$

Every time a word reaches the form of $H\#u\#$, where $u \in \Sigma_L$, the word $u$ holds the next state of the L-system. A version which has been implemented in the exotic programming language Thue[1] [7] can be be downloaded[2].

The main difference between the two systems is that rules only contain one letter at the left-hand side, that there is only one rule per letter and that all rules are applied at once to reach the new state. Furthermore an initial word is hard coded in the L-system. However, there are variations of L-systems such as non-deterministic ones, where there may be more than one rule for each letter and rules are chosen non-deterministically. Furthermore there are context sensitive L-systems that choose rules according to the environment of a letter in the word. Both these variations can easily be modeled with semi-Thue systems.

**Markov algorithms** are deterministic string rewriting systems, where rewriting rules are gathered in an ordered list. Given an initial word, they take the first rule which is able to rewrite a substring and rewrite the leftmost occurrence of this substring. If no rule can be applied, the algorithm stops. Furthermore there is a distinction between normal rules and terminating rules. After the application of a terminating rule, the algorithm stops as well.

The main difference to semi-Thue systems is the drop of the non-determinism. There is no ambiguity about which rule to apply when. Another difference is the introduction of terminating rules. A semi-Thue system does not have any property such as halting (However, as soon as no rule can be applied on a word anymore, we can colloquially say that the derivation terminated).

### 2.3.2 Turing Machines

It is actually stunningly easy to construct a semi-Thue system which simulates the behavior of a Turing machine. The following description uses the notion of [1] pp. 23-32, and follows the example shown in [8], p. 89. Take an arbitrary Turing machine with internal states $Q$, tape alphabet $\Gamma$, inital state $q_0$ and final states $F$, whose transition function $\delta$ is given by a list of 5-tuples $(q_i, s_j, q_l, s_k, d)$ with $q_i \in Q \cup \{q_0\}$, being the current state, $s_j \in \Gamma$ the current tape symbol, $q_l \in Q \cup F$ the next state, $s_k \in \Gamma$ the next symbol, and $d \in \{L, R\}$ the direction to move the head – left ($L$) or right ($R$). Furthermore, any configuration $i$ is denoted as $w_1 q_i w_2$, where $w_1$ and $w_2$ are words from $\Gamma^*$.

Now, we can take our semi-Thue system to be of alphabet $\Sigma = Q \cup F \cup \{q_0\} \cup \Gamma$, and identify every configuration with a word from $\Sigma^*$. The position of the current state symbol $q_i$ denotes the position of the head. The right adjacent symbol $s_j$ is the current symbol to read. Now every rewriting has to be done according to the rule in $\delta$. A tuple $(q_i, s_j, q_l, s_k, L)$ yields the rewriting rule $(s_m q_i s_j, q_l s_m s_k)$, analogously a rewriting with move to the right is performed via $(q_i s_j s_m, s_k q_i s_m)$. Since there can never be final states at the left side of the rewriting rule, any rewriting is impossible after reaching a final state.

The other direction is simple, too, as we can easily construct a non-deterministic Turing machine, which implements an arbitrary semi-Thue system $(\Sigma, S)$. Take the tape alphabet to be the alphabet of the semi-Thue system with an additional blank symbol $b$. Given a semi-Thue system and an initial word, the machine non-deterministically chooses a rule to apply and a position on the tape to apply it on. It first checks whether the rule can be applied at this position. If it can, it would shift the part right from the substring to the left or to the right (according to the length of the right-hand side of the rule) as in [9] and rewrites the substring at this formerly chosen position. If it cannot, it would choose a new position, new rule, or both.

The machine would never halt (just like semi-Thue systems are generally not intended to ever halt), however, we can ask it to compare two given words $w_1$ and $w_2$ and thus check whether $w_2$ is derivable in $w_1$. For a check of this kind we could either hard code the demanded word $w_2$ into the machine or use a second tape, on which $w_2$ is encoded. A check submachine would then simply compare both tapes after each application of a rule. Eventually, it would halt if both words are the same.

[1]download at `http://people.physik.hu-berlin.de/~bfmaier/moc/thue.py` – run programs with `> python thue.py -d program_name.t`
[2]`http://people.physik.hu-berlin.de/~bfmaier/moc/lsystem.py`

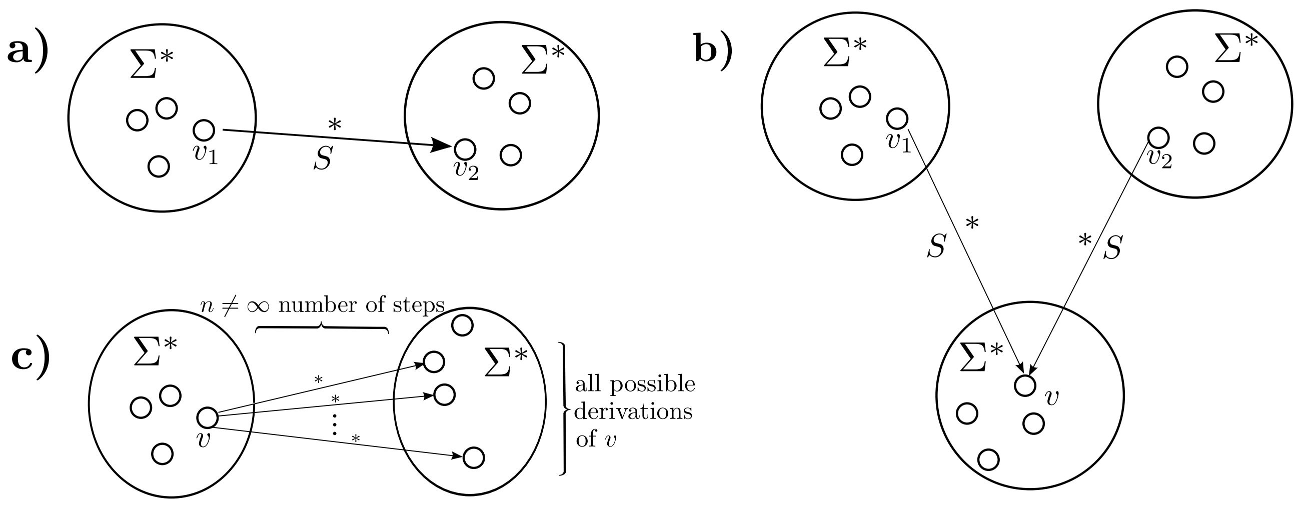

Figure 1: Illustrations of **(a)** the accessibility problem, **(b)** the common descendant problem, **(b)** the termination problem

## 2.4 Word Problems

Together with the notion of substring rewriting, Thue introduced certain decision problems connected to them, a list which later was expanded. The last mentioned application of a semi-Thue system represents one of the more simple word problems, regarding the question whether one word of $\Sigma^*$ is derivable from another word of $\Sigma^*$ in $S$. In the following, I introduce some of them.

### 2.4.1 The Accessibility Problem

Given two words $v_1, v_2 \in \Sigma^*$, is $v_2$ derivable from $v_1$ in $S$? An illustration of the problem can be seen in 1a. A proof in [4] constructs a Post machine which halts if and only if $v_2$ is derivable from $v_1$. Hence it reduces the decision problem to the halting problem of Turing/Post machines, which is undecidable for the constructed machine. Therefore the accessibility problem is undecidable.

Another proof uses the reducibility of the accessibility problem to Post's correspondence problem, which has also been proven to be undecidable.

### 2.4.2 The Common Descendant Problem

Given two words $v_1, v_2 \in \Sigma^*$, is there a $v \in \Sigma^*$, s.t. $v_1$ is derivable from $v$ and $v_2$ is derivable from $v$? An illustration of the problem can be seen in 1b. A proof in [3] constructs 3-rule semi-Thue systems with certain properties, for which the problem is shown to be undecidable. Therefore the common descendant problem is undecidable.

### 2.4.3 The Termination Problem

Given a word $v \in \Sigma^*$, does every derivation in $S$, starting with $v$, have finite length? An illustration of the problem can be seen in 1c. A proof in [3] for its undecidability uses the same structure of 3-rule semi-Thue systems.

### 2.4.4 An Additional Problem

Given a word $v_1 \in \Sigma^*$, is there a $v \in \Sigma^*$, s.t. $v_1$ is derivable from $v$ in $S$? I formulated this problem by myself and could not find any proofs for it. However, it can be handled quite easily.

From the rules of a semi-Thue system $(\Sigma, S)$, construct a semi-Thue system $(\Sigma, S')$, where all the rules in $S$ are inverted to yield $S'$. Now, given a $v_1 \in \Sigma^*$, we can construct a Turing machine which searches through all letters of $v_1$ and tries to apply every rule from $S'$ on the substring beginning at this letter. As soon as it is possible to apply a rule, the machine applies it and halts in an accepting state. We found a word $v$ from which $v_1$ can be derived. As soon as it reaches the end of the word, it will halt in a rejecting state. Thus, the machine halts in an accepting state if and only if there exists such a $v$. Since $v$ is finite, the machine will always halt and give a yes or no answer. Hence, the problem is decidable.

# 3 Quantum Computing

Semi-Thue systems and Turing machines are classical models of computation. However, there are new approaches for other models, which might have more power than the classical models. One of them is the model of quantum computing, which I want to give a short introduction to in the following.

With the challenge of Moore's law [10], stating that the average number of transistors placed on an integrated circuit will double about every two years, there are two main problems in sight for the computer industry. The first one is energy cost, which will increase with the number of transistors. The second one is the occurrence of quantum effects. Since the transistors get smaller and smaller to be placed on an integrated circuit, they will eventually reach a scale on which quantum effects can not be neglected anymore. However, in the beginning of the 1980's, it was the famous physicist R. Feynman who asked if it was possible to even *use* quantum systems for computations. He introduced quantum systems which act like classical computers by means of reversible operations [11]. Later on, people developed a more general view of quantum computing, which actually made use of quantum effects to hypothetically speed up algorithms. First, D. Deutsch introduced a universal quantum Turing machine and showed the connection to classical models of computation [12].

In the following sections I will give a short introduction to the topic, describing reversible gates as unitary quantum operators and introducing an algorithm which makes computations faster compared to the best classical solutions. I will base my notation on an introduction I published earlier [13] and that I would recommend the non-prepared reader to read in advance. However, I will recapitulate a bit of the notion due to readability. If you are familiar with the basic concepts of quantum physics, you can skip the following section.

## 3.1 Important Properties of Quantum Systems

In quantum physics, a system can be described as being in possible discrete states. A state of a quantum system is denoted by the "ket"-notation $|\cdot\rangle$. These states can be seen as orthonormal base vectors in a certain complex vector space, called "Hilbert space". We distinguish between bosonic states $|0\rangle, |1\rangle, |2\rangle ...$ and fermionic states $|0\rangle, |1\rangle$. Note that for every Hilbert space, there is a dual space, which in this case is denoted by the "bra"-notation $\langle\cdot|$. We yield that notation by using the $^\dagger$ operator, which represents transposition and complex conjugation and works as $|\phi\rangle^\dagger = \langle\phi|$ and $\langle\phi|^\dagger = |\phi\rangle$.

Now, as with every vector space with a dual space, we are able to build scalar products with the base vectors, denoted as $\langle n|m\rangle$. A scalar product is always associated with a probability, meaning that $|\langle n|m\rangle|^2$ is the probability to measure the state $|m\rangle$ to be in state $|n\rangle$ (which for the base vectors is always 0 for $m \neq n$ and 1 for $m = n$, usually denoted with Kronecker's delta $\delta^n_m$ which has the same behavior).

Furthermore we are able to build linear combinations to yield a superposition of quantum states

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle + \gamma|2\rangle, \tag{18}$$

where $\alpha\beta\gamma$ are complex numbers and called "probability amplitudes". The value $|\alpha|^2$ represents the probability that we measure the state $|\psi\rangle$ to be $|0\rangle$. Therefore, we have the additional normalization condition $|\alpha|^2 + |\beta|^2 + |\gamma|^2 = 1$. The physical interpretation of a state like $|\psi\rangle$ is that the quantum system represents all three states at the same time. The only prediction of the outcome of a measurement can be done according to probabilities. Once measured, a superposition of states collapses to the measured state.

In addition to the states, we can build operators. Operators take states as an input and transform them to other states. They are denoted by $|\cdot\rangle\langle\cdot|$ and can be associated with matrices. Important operators are e.g. the hermitian operators $H$, which are their own hermitian conjugate $H = H^\dagger$ and therefore have real eigenvalues. These are called observables, because they can be associated with values in nature. A second very important kind of operators is the one of unitary operators $U$. These have the property $U^{-1} = U^\dagger$, which means that $UU^\dagger = U^\dagger U = \mathbb{1}$, where $\mathbb{1}$ is the identity. Note that every action we perform with a unitary operator $U$ can be easily reversed by applying $U^\dagger$ on the result to yield the initial state.

Important operators for the following considerations are the fermionic ladder operators

$$a = |0\rangle\langle 1|, \qquad a^\dagger = |1\rangle\langle 0|, \tag{19}$$

where $a$ applied on $|0\rangle$ yields 0 (applied on $|1\rangle$ yields $|0\rangle$) and $a^\dagger$ applied on $|0\rangle$ yields $|1\rangle$ (applied on $|1\rangle$ yields 0). The operator $^\dagger$ acts on a series of operators $ABC$ as $(ABC)^\dagger = C^\dagger B^\dagger A^\dagger$. Another important fermionic operator is the

$$NOT = a + a^\dagger = |0\rangle\langle 1| + |1\rangle\langle 0|, \tag{20}$$

which, acting on $|0\rangle$ yields a $|1\rangle$ and acting on $|1\rangle$ yields a $|0\rangle$

$$NOT\,|0\rangle = |0\rangle\underbrace{\langle 1|0\rangle}_{=0} + |1\rangle\underbrace{\langle 0|0\rangle}_{=1} = |1\rangle, \tag{21}$$

$$NOT\,|1\rangle = |0\rangle\underbrace{\langle 1|1\rangle}_{=1} + |1\rangle\underbrace{\langle 0|1\rangle}_{=0} = |0\rangle. \tag{22}$$

The operator is unitary. The last important operator is the identity, which reads

$$\mathbb{1} = \sum_{n=n_{\min}}^{n_{\max}} |n\rangle\langle n| \tag{23}$$

and with $n_{\min} = 0$ and $n_{\max} = 1$ is $\mathbb{1} = |0\rangle\langle 0| + |1\rangle\langle 1|$ in the fermionic case.

## 3.2 Setup of a Quantum Computer

A quantum system which is able to perform computations consists of two main units.

1. **Quantum storage** – a quantum storage is a many-body storage of single-body quantum systems. Quantum systems can be in certain states – in general, every quantum system can be used for computations, bosonic, fermionic or even real number states. However, we will mostly consider the storage to be made of interacting fermionic states. Here, fermionic relates to the algebra of the group of Pauli matrices and therefore means that a single-body system can be in two states, in the following called $|0\rangle$ and $|1\rangle$. Since they can be seen as bits, we call them "quantum bits" or simply "qubits".

   Now a quantum storage (i.e. a state) of $n$ qubits can be represented as a tensor product of these qubits, thus we write

   $$|\psi\rangle = |0\rangle^{(0)} \otimes |0\rangle^{(1)} \otimes ... \otimes |0\rangle^{(n-1)} = |00...0\rangle. \tag{24}$$

   The remarkable thing about qubits and their main difference to classical bits is, that they are able to be in a superposition of states, meaning that they represent $|0\rangle$ and $|1\rangle$ at the same time. Furthermore every qubit is associated with a phase factor, called the "probability amplitude" which encodes more information such as the probability to measure a qubit to be in a certain state.

2. **Computational operations** – an operation on qubits has to be done via unitary operators, which are called "quantum gates" in quantum computing. As I stated in the introduction, reversibility of computations is a crucial property to reduce energy dissipation (more about that can be read in the paper by D. Esser). The reversibility is achieved by the use of unitary operators, since these are easy to reverse (by hermitian conjugation). Important unitary operators are for example the fermionic ladder operators $a$ and $a^\dagger$, as well as the *NOT* or a Hadamard transformation $H$, which rotates a state by $\pi/2$ in the plane spanned by $|0\rangle$ and $|1\rangle$ (i.e. yields a superposition of states when applied to a single state).

Physical realizations of these systems can have various forms. For example, it is possible to use spin systems (states "up": $|\uparrow\rangle$ and "down": $|\downarrow\rangle$), as it was done for an experimental realization of Shor's algorithm (speed up of prime number factorization). Vanderspyn et.al. used the magnetic spin resonance in certain molecules to use the atoms' nuclear spins as qubits [14]. Quantum gates for these spin systems can be realized via certain magnetic field configurations.

Another possibility to achieve two-state systems is to use the polarization of photons, which is either horizontal ($|H\rangle$) or vertical ($|V\rangle$) [15]. Quantum gates could be achieved by configuration of cavity filters.

A third possibility is to act on trapped atoms in an optical lattice [16]. Here, "ultracold atoms" are trapped in a periodical potential and can be manipulated to be in the ground state $|0\rangle$ or the first excited state $|1\rangle$. Lasers can be used to prepare quantum gates. Advantages of that method are e.g. error reduction by a good isolation from the environment.

It is important to note that there are other important things than storage and gates regarding quantum computing, such as error handling, precision increase and, as mentioned, isolation from the environment. In fact, these are the properties which make it really hard to actually implement quantum computers in the real world. However, since this is supposed to be an introductory paper on the model of computation, I will not consider them in the following. The interested reader is encouraged to read further in [15].

## 3.3 Quantum Computing as Classical Reversible Computing

A main property of classical computing is that it is logically irreversible. Take for example the *AND* operator, which takes two bits as input and yields one bit as output. Given just the output, it is not possible to determine the form of the input, because the *AND* does not act one-to-one. Due to that erasure of information, the entropy of the system increases, which leads to energy dissipation in form of heat [17]. Thus, the actual energy which is needed for computation does not come necessarily through processing but simply from the loss of information. Given a reversible system, it is possible to rewind it to its initial state, thus keeping the entropy fixed and the dissipated energy zero.

In order to act physically reversible, a computer has to be logically reversible. It has been shown that a classical computational model with reversible properties is actually possible to construct by using a Turing machine with an additional "garbage" tape [18]. The machine uses this tape to store bits which have been overwritten in order to make the machine reversible. Furthermore it is possible to achieve certain reversible logical gates which act one-to-one and substitute the former irreversible correspondents [19].

For quantum computing, reversible operations are not a necessity but due to the reasons mentioned above, it is highly recommended to include this property. With reversibility we associate unitary quantum gates, which bring along other advantages, such as preserving the norm of states or the reversible rotation of states in the hyperplane. In the following I will present certain reversible logical gates in the notion of quantum computers and an example for a computation (both following [11]).

### 3.3.1 Operators

**Not** The $NOT_q$ flips the qubit $q$. It is defined as $NOT_q = a_q^\dagger + a_q$ and is hermitian and unitary

$$NOT_q^\dagger = NOT_q, \tag{25}$$

$$NOT_q NOT_n = \underbrace{a_q^\dagger a_q + a_q a_q^\dagger}_{=\mathbb{1}} + \underbrace{a_q a_q}_{=0} + \underbrace{a_q^\dagger a_q^\dagger}_{=0} = \mathbb{1}. \tag{26}$$

**If and Controlled If** The $I\!F_{q=b}$ acting on a state $|a\rangle^{(q)}$ projects the state on the basis vector $|b\rangle$ (with $a,b \in \{0,1\}$). It is defined as

$$I\!F_{q=b} = |b\rangle\langle b|^{(q)}, \tag{27}$$

is hermitian but *not* unitary ($I\!F_{q=b} I\!F_{q=b} = I\!F_{q=b}$). The superscript denotes that the operation takes place in the Hilbert space of the $q$-th qubit. A more general version of the "if" takes two words $v,w \in \{0,1\}^*$ of the same length $\ell$ as an input and checks whether they are equal

$$I\!F_{v=w} = I\!F_{v_1=w_1} \otimes I\!F_{v_2=w_2} \otimes \ldots \otimes I\!F_{v_\ell=w_\ell}, \tag{28}$$

with $v_i$ denoting the qubit $|a\rangle^{(v_i)}$ and $w_i$ denoting the $i$-th letter of the word $w$.
The "controlled if" $CI\!F_{v=w}(U_Q)$ performs the unitary transformation $U_{Q\setminus\{|a_i\rangle^{(v_i)}\}}$ on a set of states $Q$ without any of the states $|a_i\rangle^{(v_i)}$ if and only if two words $v$ and $w$ are equal. Thus, the operator reads

$$CI\!F_{v=w}(U_Q) = \mathbb{1} + (U_{Q\setminus\{|v_i\rangle\}} - \mathbb{1})\, I\!F_{v=w} \tag{29}$$

and is unitary

$$\begin{aligned} CI\!F_{v=w}(U_Q)\ (CI\!F_{v=w}(U_Q))^\dagger &= [\mathbb{1} + (U-\mathbb{1})\, I\!F]\ \left[\mathbb{1} + (U^\dagger - \mathbb{1})\, I\!F\right] && (30)\\ &= \mathbb{1} + (U-\mathbb{1})\, I\!F + (U^\dagger-\mathbb{1})\, I\!F + (U-\mathbb{1})(U^\dagger-\mathbb{1})\, I\!F && (31)\\ &= \mathbb{1} + U\, I\!F - I\!F + U^\dagger I\!F - I\!F + UU^\dagger\, I\!F - U^\dagger\, I\!F - U\, I\!F + I\!F && (32)\\ &= \mathbb{1} && (33)\end{aligned}$$

(second way analogous).

**Controlled Not or Reversible Exclusive Or** The $CNOT_{q,u}$ takes two qubits $|a\rangle^{(q)}|b\rangle^{(u)}$ as input and flips $|b\rangle^{(u)}$ if and only if $|a\rangle^{(q)} = |1\rangle^{(q)}$. Thus, it can be defined as

$$CNOT_{q,u} = CI\!F_{q=1}(NOT_u). \tag{34}$$

Hence, its action preserves $|a\rangle^{(q)}$ while it transforms $|b\rangle^{(u)}$ to $|a\oplus b\rangle^{(u)}$, where $\oplus$ denotes the "exclusive or" operation

$$CNOT_{q,u}|a\rangle^{(q)}|b\rangle^{(u)} = |a\rangle^{(q)}|a\oplus b\rangle^{(u)}. \tag{35}$$

The "exclusive or" has the properties

- commutativity – $a \oplus b = b \oplus a$,
- associativity – $(a \oplus b) \oplus c = a \oplus (b \oplus c)$,
- existence of identity – $a \oplus 0 = a$,
- the inverse is given by the identity of the input – $a \oplus a = 0$

**Controlled Controlled Not or Toffoli Gate** The $CCNOT_{q,u,v}$ takes three qubits $|a\rangle^{(q)}|b\rangle^{(u)}|c\rangle^{(v)}$ as input and flips $|c\rangle^{(v)}$ if and only if $|a\rangle^{(q)} = |1\rangle^{(q)}$ and $|b\rangle^{(u)} = |1\rangle^{(u)}$. Thus, it can be defined as

$$CCNOT_{q,u,v} = CIF_{qu=11}(NOT_v). \tag{36}$$

Hence, its action preserves $|a\rangle^{(q)}$ and $|b\rangle^{(u)}$ while it transforms $|c\rangle^{(v)}$ to $|c \oplus (a \wedge b)\rangle^{(u)}$.

$$CCNOT_{q,u}|a\rangle^{(q)}|b\rangle^{(u)} = |a\rangle^{(q)}|a \oplus b\rangle^{(u)}. \tag{37}$$

The $CCNOT$ is a universal in the sense that it is possible to create all boolean functions with it.

### 3.3.2 Example for a Computation

Using the gates above, it is possible to perform all possible computations of a classically acting quantum computer of reversible nature. As a short example I want to introduce an operator which takes two arbitrary qubits $|a\rangle^{(q)}|b\rangle^{(u)}$ and a carry qubit $|0\rangle^{(C)}$ and performs a bit addition to the second qubit and the carry. The classical reversible operator preserves the bit $a$, projects the carry bit to $a \wedge b$ and projects the bit $b$ to $a \oplus b$. Thus, for a quantum computation of this kind, we can first use the operator $CCNOT_{q,u,C}$ and afterwards the operator $CNOT_{q,u}$, yielding the operator

$$\begin{aligned} ADD_{q,u,C}|a\rangle^{(q)}|b\rangle^{(u)} &= CNOT_{q,u}CCNOT_{q,u,C}|a\rangle^{(q)}|b\rangle^{(u)}|0\rangle^{(C)} && (38)\\ &= CNOT_{q,u}|a\rangle^{(q)}|b\rangle^{(u)}|0 \otimes (a \wedge b)\rangle^{(C)} && (39)\\ &= |a\rangle^{(q)}|a \oplus b\rangle^{(u)}|a \wedge b\rangle^{(C)}. && (40)\end{aligned}$$

The whole process is reversible by applying the inverse operator

$$ADD^{\dagger}_{q,u,C} = CCNOT^{\dagger}_{q,u,C}CNOT^{\dagger}_{q,u} = CCNOT_{q,u,C}CNOT_{q,u} \tag{41}$$

on the resulting state (note that in the last step we used that $CNOT$ and $CCNOT$ are hermitian operators – they are composed of the hermitian operators $IF$ and $NOT$).

$$\begin{aligned} ADD^{\dagger}_{q,u,C}|a\rangle^{(q)}|a \oplus b\rangle^{(u)}|a \wedge b\rangle^{(C)} &= CCNOT_{q,u,C}CNOT_{q,u}|a\rangle^{(q)}|a \oplus b\rangle^{(u)}|a \wedge b\rangle^{(C)} && (42)\\ &= CCNOT_{q,u,C}|a\rangle^{(q)}|a \oplus a \oplus b\rangle^{(u)}|a \wedge b\rangle^{(C)} && (43)\\ &= |a\rangle^{(q)}|b\rangle^{(u)}|(a \wedge b) \oplus (a \wedge b)\rangle^{(C)} && (44)\\ &= |a\rangle^{(q)}|b\rangle^{(u)}|0\rangle^{(C)} && (45)\end{aligned}$$

which is the initial state.

## 3.4 Quantum Turing Machine

In 1985, D. Deutsch showed that a quantum computer can model the behavior of a reversible Turing machine [12]. He furthermore introduced a universal quantum computer which is capable of modeling the behavior of every quantum computer up to arbitrary precision. His notion is as follows.
He introduces a "tape state" made of infinite qubits

$$|\mathbf{m}\rangle = |...m_{-2}m_{-1}m_0m_1m_2....\rangle. \tag{46}$$

Second, he uses an additional state of qubits as a finite set of $2^M$ machine states

$$|\mathbf{n}\rangle = |n_0n_1...n_{M-1}\rangle, \tag{47}$$

and an integer marker state $|x\rangle$. The marker state has to be seen as the Turing machine's head, thus can be increased or decreased according to the current state and the tape symbol $m_x$. To this end, we need the decrement and increment function in order to add and subtract numbers to the marker. We can do this by introducing ladder operators for the integer number states (which I will do explicitly here, because we will need it in the next section, too).

$$\text{annihilation:} \qquad z = \sum_{n=-\infty}^{\infty} |n\rangle\langle n+1| \tag{48}$$

$$\text{creation:} \qquad z^{\dagger} = \sum_{n=-\infty}^{\infty} |n+1\rangle\langle n| \tag{49}$$

Note, that these operators are unitary

$$zz^{\dagger} = \sum_{m,n} |n\rangle \underbrace{\langle n+1|m+1\rangle}_{=\delta_{n+1,m+1}} \langle m| = \sum_{n} |n\rangle\langle n| = \mathbb{1} \tag{50}$$

$$z^{\dagger}z = \sum_{n=-\infty}^{\infty} |n+1\rangle\langle n+1| = \sum_{n=-\infty}^{\infty} |n\rangle\langle n| = \mathbb{1}. \tag{51}$$

Thus we found the increment operator $z^{\dagger}$ as well as the decrement operator $z$. Note that the introduced operators must not be mistaken for the bosonic ladder operators.

Now every transition from a configuration $|x;n;m\rangle$ to a configuration $|x';n';m'\rangle$ has to be mapped to a unitary transformation $U$ with matrix elements

$$\langle x;\mathbf{n};\mathbf{m}|U|x';\mathbf{n}';\mathbf{m}'\rangle = \delta_{x'}^{x+1} U^{+}(\mathbf{n},m_x|\mathbf{n}',m_x') + \delta_{x'}^{x-1} U^{-}(\mathbf{n},m_x|\mathbf{n}',m_x'). \tag{52}$$

The $\delta_{x'}^{x\pm1}$ make sure that the head can only move one step left or right at a time, with $U^{+}$ being the unitary transformation for a head shift to the right and equivalently for a head shift to the left for $U^{-}$. The $U^{\pm}$ determine whether the transition from state $\mathbf{n}$ and input symbol $m_x$ to state $\mathbf{n}'$ and output symbol $m_x$ is possible or not (thus giving 1 if it is possible and 0 else). To satisfy the operations to be unitary he furthermore introduces

$$U^{\pm}(\mathbf{n},m|\mathbf{n}',m') = \frac{1}{2}\delta_{\mathbf{n}'}^{\mathbf{A}(\mathbf{n},m)}\delta_{m'}^{B(\mathbf{n},m)}\left[1 \pm C(\mathbf{n},m)\right], \tag{53}$$

with functions of range $\{0,1\}^M$ for $\mathbf{A}$, $\{0,1\}$ for $B$ and $\{-1,1\}$ for $C$. The requirement for the whole function to be unitary is that the mapping from the tuple $(\mathbf{n},m)$ to the value of the functions $\mathbf{A}$, $B$ and $C$ is bijective. This machine can model only reversible Turing machines (which is as much as modeling a normal Turing machine, as shown in [18]).

The whole machines seems a bit odd, since it contains every transition hard-coded and is therefore counter-intuitive (meaning that there is not really a read-out of tape bits, after which the machine acts, but every transition is set beforehand). However, Turing machines can be said to work exactly like that. Given a certain input, they act deterministically as if we would have hard-coded the whole process.

Generally, since coefficients of states can be complex and states itself can be of real nature, it does not seem to be possible to show the simulation of quantum computing on a classical model or even with a universal quantum computer. As indicated in [12], due to these properties, a quantum computer can act in principle more powerful than any classical computation method. However there is a solution to this problem given in the same paper. Introducing so-called “generators” (similar to those of groups), he shows that every arbitrary unitary transformation can be modeled up to arbitrary precision by successive application of the generators.

Furthermore, as claimed in [20], we can limit quantum computing on working with finite sets of “interesting” real or complex numbers, e.g. rotations in a hyperplane can with arbitrary precision be performed by a complex number from the set

$$C = \left\{ \mathrm{e}^{i\pi q} : q \in \mathbb{Q}_n, -1 \le q < 1 \right\}. \tag{54}$$

with $Q_n$ being the rational numbers of precision $n$. Since $C$ is constructed from a finite subset of the rational numbers, it is finite. The upper boundary 1 has been chosen not to have a representation in $C$, because the resulting complex number would be covered by $q = -1$ (note that $\mathrm{e}^{i\pi} = \mathrm{e}^{-i\pi}$. Another set of interesting numbers is

$$R = \{\pm\sqrt{q}, q : q \in \mathbb{Q}_n, 0 \le q \le 1\}. \tag{55}$$

The set $R$ is finite. Now the overall set of interesting numbers is

$$I = \{r \times c : r \in R, c \in C\}. \tag{56}$$

$I$ is finite. However, in most cases, it will be sufficient to work with small sets of coefficients such as $\left\{-1, -2^{-1/2}, 0, 2^{-1/2}, 1\right\}$. With these sets, it is possible to model any quantum computer up to arbitrary precision.

## 3.5 Quantum Parallelism

### 3.5.1 Superpositions of States

We just saw that a quantum computer can model classical computations. However, there is more that a quantum computer actually can do. Let me introduce the Hadamard transformation

$$H = \frac{1}{\sqrt{2}}\Big(|0\rangle\langle 0| + |0\rangle\langle 1| + |1\rangle\langle 0| - |1\rangle\langle 1|\Big), \tag{57}$$

which is hermitian and unitary. Acting on a base state, it yields a superposition of states

$$H|0\rangle = \frac{1}{\sqrt{2}}\Big(|0\rangle + |1\rangle\Big) \tag{58}$$

$$H|1\rangle = \frac{1}{\sqrt{2}}\Big(|0\rangle - |1\rangle\Big). \tag{59}$$

This represents a basic principle of quantum mechanics. A system can be in a superposition of states, representing two bits at the same time. A computation (i.e. a unitary operator) can act on this superposition, thus yielding two (or more) results at once. This phenomenon is called “quantum parallelism”. However, we can’t directly use the outcome of our computation. As an example, suppose we have the initial state

$$|x = 0\rangle = |000\rangle. \tag{60}$$

Now we can use the Hadamard transformation on every qubit to prepare a superposition of all possible states, yielding

$$|y\rangle = \frac{1}{2\sqrt{2}}\Big(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle\Big). \tag{61}$$

Let us perform the $ADD_{(0),(1),(2)}$ operation on $|y\rangle$. We obtain

$$ADD_{(0),(1),(2)}|y\rangle = \frac{1}{2\sqrt{2}}\Big(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |110\rangle + |111\rangle + |101\rangle + |100\rangle\Big). \tag{62}$$

Every possible result has been computed at one step! However, we cannot easily read-out the result for every computation. A read-out cannot be performed without a collapse of the state. But after collapsing the state, we obtained only one result and all the others are gone. We could think of copying the state to another quantum computer (or a lot of other computers), in order to read out all the states. However, as the no-cloning theorem shows, it is not possible to copy quantum states [21].

So is there a possibility to use the quantum parallelism to obtain algorithms which are superior to classical algorithms? It turns out that there is!

### 3.5.2 Grover’s Algorithm

A basic problem in computer science is the one of finding a certain member in a list of $N$ members. Suppose an ordered list, which is the best case, then we can use the binary search algorithm to obtain an algorithm of order $O(\log_2 N)$. However, if the list is unordered, there is no other classical strategy than going through the list member by member until we find the wanted item. This is an algorithm of order $O(N)$ for $N \gg 1$. However, using quantum computing, one can find a procedure of order $O(\sqrt{N})$ [22], namely Grover’s algorithm, sometimes called the Grover iteration, which I will introduce in the following. A reader who is not interested in the actual derivation of the algorithm may skip the next paragraphs.

Suppose we have an unordered list of $N$ items. We can label it with binary numbers (i.e. qubits), s.t. the state $|n\rangle$ labels the $n$-th item of the list. One of them is the wanted marked state $|\omega\rangle$. Second, we have a “quantum oracle”, which knows $|\omega\rangle$. Given a certain number state $|x\rangle$ of the list, it flips the sign of $|x\rangle$ if and only if $|x\rangle = |\omega\rangle$. It acts as identity otherwise. Therefore, we can see it as an operator which acts like

$$U_\omega|x\rangle = \begin{cases} -|\omega\rangle & x = \omega \\ |x\rangle & x \neq \omega \end{cases}. \tag{63}$$

It can be described as

$$U_\omega = \mathbb{1} - 2|\omega\rangle\langle\omega| \tag{64}$$

and is hermitian. It follows that it is unitary

$$U_\omega U_\omega = \mathbb{1} - 2|\omega\rangle\langle\omega| - 2|\omega\rangle\langle\omega| + 4|\omega\rangle\langle\omega| = \mathbb{1}. \tag{65}$$

Note that $\omega$ is still unknown. Hence not us ourselves can perform the transformation, but it has to be done by the oracle. Now let us prepare the initial state of our quantum computer to be a superposition of all items

$$|s\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle, \tag{66}$$

which can be achieved by first preparing the state to $|00...0\rangle$ and then Hadamard transforming every qubit. We are able to perform another unitary transformation

$$U_s = 2|s\rangle\langle s| - \mathbb{1}. \tag{67}$$

We consider the following thoughts. The probability to find the right state by just measuring $|s\rangle$ is $|\langle\omega|s\rangle|^2 = N^{-1}$. How can we actually measure the state $|s\rangle$? We prepare gates in the direction of the bases states $|x\rangle$. Every state is associated with an eigenvalue $\lambda_x$. Usually when we measure states like this, we obtain the outcome of the measurement by obtaining the eigenvalue and deduce the associated state (imagine, e.g., light of different wavelengths to be the eigenvalue).

Now we use the unitary transformations above to act on the prepared state $|s\rangle$,

$$\begin{aligned} U_s U_\omega &= U_s\left(\mathbb{1}|s\rangle - 2|\omega\rangle\langle s|\omega\rangle\right) && (68)\\ &= U_s\left(|s\rangle - \frac{2}{\sqrt{N}}|\omega\rangle\right) && (69)\\ &= \mathbb{1}|s\rangle - \frac{2}{\sqrt{N}}\left(\frac{2}{\sqrt{N}}|s\rangle - |\omega\rangle\right) && (70)\\ &= (1-c^2)|s\rangle + c|\omega\rangle && (71) \end{aligned}$$

with $c = 2/\sqrt{N}$. We obtained a new quantum state, where the single number states are associated with the probability amplitudes

$$\alpha_\omega = c + \frac{1}{\sqrt{N}}(1-c^2) \tag{72}$$

$$\alpha_{x\neq\omega} = \frac{1}{\sqrt{N}}(1-c^2). \tag{73}$$

Since $\alpha_\omega > \alpha_{x\neq\omega}$, the probability to measure the state to be $|\omega\rangle$ (or equally, measuring the eigenvalue $\lambda_\omega$) is now higher than measuring explicitly one of the other states (although it is still lower than measuring any of the other states). By performing the unitary transformations iteratively, we increase the probability amplitude for measuring $\omega$ with every step. A geometrical investigation of the situation shows that the unitary transformation can be associated with a rotation of an angle $\theta$ of the current state in the hyperplane spanned by $|\omega\rangle$ and $|s\rangle - |\omega\rangle$. Because the transformation is a rotation, the current state comes closer to the direction of $|\omega\rangle$ with each step of the iteration. However, after being in an optimal state, it would get rotated even further, thus decreasing the probability to measure $|\omega\rangle$. Hence, there is an optimal number of iterations to rotate the state s.t. the probability to measure $|\omega\rangle$ becomes nearly 1. This optimal number is proportional to $N^{-1/2}$. A short algebraic, as well as the geometrical derivation, can be seen in [23].

I am, however, skeptical towards this algorithm, because it needs the "oracle" which already knows the solution. How can this oracle be constructed without knowing the solution? Unluckily there is no answer to this question, the function of the oracle is always described as a black box.

## 3.6 Summary and Outlook

In this section I introduced the general notion of quantum computing and introduced quantum operators for computations in terms of a reversible classical machine. Furthermore I gave indications on the connection to classical Turing machines and continued with presenting an algorithm which is capable to speed up the solution of a problem in contrast to the solution with a classical computer.

Indeed, there are other algorithms which decrease the order with respect to classical solutions, like Shor's algorithm or the Deutsch-Jozsa algorithm. Complexity theorists introduced new classes in order to classify the speed up of quantum algorithms. Since quantum computing is expected to increase humanity's computational power, a lot of research is in progress currently. Theoretical research for developing new algorithms, as well as experimental to cope with difficulties such as decoherence and isolation. The general hope is to be able to actually build universal quantum computers one day.

# 4 Connection of the Models

In order to connect the two models described in the last two sections, it is necessary and sufficient to show in both directions that one model can simulate the other. Therefore, we will use the following sections to create a qantum computer that models arbitrary semi-Thue systems, as well as trying to create a semi-Thue system that simulates the behavior of an arbitrary quantum computer. However, the latter has to be restricted to a simulation up to a finite precision, since a real quantum computer can hypothetically work with all real numbers, whereas a classical computer cannot. There will further problems arise which will not be solved in this paper. However, indications on the solutions are given and a semi-Thue system for a certain quantum algorithm is presented. Please also note that the systems worked out in this section are purely made to show the connection between the two models of computation, meaning that they might lack mathematical elegance.

## 4.1 Simulating Word Problems on a Quantum Computer

### 4.1.1 Essentials

It is useful to begin with pointing out the main properties of semi-Thue systems and briefly state how to model them with a QC.

1. finite alphabet – We are going to remodel a finite alphabet to the binary alphabet (represented by qubits) and show that they are equivalent.
2. rewriting rules – The rules have to be modeled by unitary operators in order to be reversible. To this end we need a storage of replaced words (in the following called "garbage section", as well as a storage of zeros for the new words additionally to a storage that keeps the current word.
3. non-deterministic behavior – A random number generator will pick rewriting rules and apply them to a random position on the word storage.

### 4.1.2 General Setup

The quantum computer consists of an infinite number of qubits

$$|t_0 t_1 t_2 t_3 t_4 \ldots\rangle \tag{74}$$

which we will call the "tape storage" following the notion of Turing machines. The tape is sectioned in three parts. The first part from position 0 up to and including position $x-1$, $x \in \mathbb{Z}$ will store the current word. A current word is encoded in the alphabet $\{0,1\}$. Suppose the initial alphabet of the word problem to be of cardinality $d$. Then, every letter will be encoded in a string of bits of length

$$N = \left\lceil \frac{\log d}{\log 2} \right\rceil . \tag{75}$$

We will solve the problem of addressing by only looking up substrings beginning on positions $aN$ on the tape, with $a = 0, 1, 2, \ldots$.

The second part of the tape from position $x$ up to and including position $y-1$, $y \in \mathbb{Z}$ will store the "garbage" from substitutions. The garbage part of the tape is actually needed to keep otherwise overwritten bits. In order to rewind the whole computation in the end, thus keeping the entropy fixed, we have to keep those bits. We will see later on, why we need $x$ and $y$ to be integer numbers rather than natural numbers. The third part will store zeros from position $y$ on. In a state of computation, one can identify the tape state to

$$\overset{\text{position}}{\text{state}} : \qquad |\overset{0}{\text{word}}; \overset{x}{\text{garbage}}; \overset{y}{\text{zeros}}\rangle . \tag{76}$$

As you may have noticed, we need to store the markers $x$ and $y$ in some way. To this end, we use two storages of integer numbers, $|x\rangle$ and $|y\rangle$.

### 4.1.3 Additional Operators

**Increment and Decrement** We can define general increment and decrement operators using the operators defined in equations (48)-(49). Suppose we want to decrement $x$ by the number $n$, the operation is given by $z_x^n|x\rangle = |x-n\rangle$. The corresponding increment is $\left(z_x^\dagger\right)^n|x\rangle = |x+n\rangle$. Since $z^n$ and $\left(z^\dagger\right)^n$ are products of unitary operators, they are unitary, as well. It becomes clear why we need $x$ and $y$ to be integers: the analogue operators for natural number states would not be unitary.

**Exchange of two Qubits** The $EX_{q,u}$ exchanges the values of the qubits $|a\rangle^{(q)}$ and $|b\rangle^{(u)}$. It is defined as

$$EX_{q,u} = CNOT_{q,u} CNOT_{u,q} CNOT_{q,u} \tag{77}$$

and is unitary. As we can see, this definition satisfies the requirement

$$\begin{aligned}
EX_{q,u}|a\rangle^{(q)}|b\rangle^{(u)} &= CNOT_{q,u} CNOT_{u,q} CNOT_{q,u}\ |a\rangle^{(q)}\ |b\rangle^{(u)} && (78)\\
&= CNOT_{q,u} CNOT_{u,q}\ |a\rangle^{(q)}\ |a\oplus b\rangle^{(u)} && (79)\\
&= CNOT_{q,u}\ |a\oplus(a\oplus b)\rangle^{(q)}\ |a\oplus b\rangle^{(u)} && (80)\\
&= CNOT_{q,u}\ |0\oplus b\rangle^{(q)}\ |a\oplus b\rangle^{(u)} && (81)\\
&= CNOT_{q,u}\ |b\rangle^{(q)}\ |a\oplus b\rangle^{(u)} && (82)\\
&= |b\rangle^{(q)}\ |(a\oplus b)\oplus b\rangle^{(u)} && (83)\\
&= |b\rangle^{(q)}\ |a\rangle^{(u)}. && (84)
\end{aligned}$$

To exchange two words $v, w \in \{0,1\}^*$ of the same length $\ell$ we introduce the general exchange

$$EX_{v,w} = EX_{v_1,w_1} \otimes EX_{v_2,w_2} \otimes \ldots \otimes EX_{v_\ell,w_\ell}. \tag{85}$$

**Permute Qubits on the Tape** A permutation is nothing more than an exchange of two adjacent qubits on the tape. Thus, we can define it as

$$P_{t_n,t_{n+1}} = EX_{t_n,t_{n+1}}. \tag{86}$$

More generally, taking a word $w$ of length $\ell$ starting at position $n$ of the tape, we can shift the whole word by $m$ bits to the right using

$$P^m_{n,\ell} = \left[P_{t_{n+m+\ell-1},t_{n+m+\ell-2}} \otimes P_{t_{n+m+\ell-2},t_{n+m+\ell-3}} \otimes \ldots \otimes P_{t_{n+1},t_n}\right]^\ell \tag{87}$$

Remember that the operator on the most right acts first. The corresponding operation to shift a word $m$ bits to the left is

$$\left(P^m_{n,\ell}\right)^\dagger, \tag{88}$$

it shifts a word of length $\ell$, beginning on position $n+m$ to the position $n$. Both operations are unitary.

### 4.1.4 Rewriting Rules

To perform rewriting, we have to map the rules of a semi-Thue system to unitary operators. In general, it works as described in the following.

1. Choose a random rule $(w,v)$ from the semi-Thue system (with word lengths $\ell_w, \ell_v$).
2. Construct a set $\bigcup_{a=0}^{x\,\mathrm{div}\,N-\ell_w} \{aN\}$ of possible substring start positions and choose a position $n$ from it randomly.
3. Check all bits from position $n$ up to and including position $n+\ell_w N-1$ with $CIF(U)$.
4. If the rule can be applied, perform the rewriting encoded in $U$.

Now we have to distinct between different cases.

**Rules $(w,v)$ with $\ell_w$=$\ell_v$** First we have to prepare the word $v$ from the first position of the zero part $y$ up to and including position $y+\ell_v N-1$ by flipping the bits which are supposed to be 1 in the word $v$. Now, the word encoded in qubits $t_n$ to $t_{n+\ell_w N-1}$ can be exchanged with the word encoded in qubits $t_y$ to $t_{y+\ell_v N-1}$ via the exchange operation. The $|x\rangle$ marker stays the same, the $|y\rangle$ marker has to be increased by $\ell_v N$. The operation is unitary and stored in $U$.

**Rules $(w,v)$ with $\ell_w$>$\ell_v$** Again, first we have to prepare the word $v$ from the first position of the zero part $y$ up to and including position $y+\ell_v N-1$ by flipping the bits which are supposed to be 1 in the word $v$. Now, the first part of the word $w$, encoded in qubits $t_n$ to $t_{n+\ell_v N-1}$, has to be exchanged with the word encoded in qubits $t_y$ to $t_{y+\ell_v N-1}$. The marker $|y\rangle$ has to be increased by $\ell_v N$. The second part of the word, encoded in qubits $t_{n+\ell_v N}$ to $t_{n+\ell_w N-1}$ has to be shifted to the end of the garbage tape. The marker $|x\rangle$ has to be decreased by $N(\ell_w-\ell_v)$. The whole operation is unitary and stored in $U$.

**Rules $(w,v)$ with $\ell_w$<$\ell_v$** First we have to shift $N(\ell_v-\ell_w)$ zeros from the beginning of the zero section to the end of word $w$ on the tape section. The markers $|x\rangle$ and $|y\rangle$ both have to be increased by $N(\ell_v-\ell_w)$. Then, we can prepare the word $v$ from the first position of the zero part $y$ up to and including position $y+\ell_v N-1$ by flipping the bits which are supposed to be 1 in the word $v$. Afterwards, we can exchange the bits representing the word $w$ by the bits representing the word $v$ and increase the marker $|y\rangle$ by $N\ell_w$. The whole operation is unitary and stored in $U$.

**Word Problem** In order to check whether the machine achieved a wanted word, one can perform two measurements from outside, the first one measuring $|x\rangle$, to check if the word has the right length. If it does, one can perform a measurement in the base of $|t_0t_1...t_{x-1}\rangle$, to check if we achieved the right state. If we did, the computation is done and the word problem solved. If not, we can go on with the computation.

**Comments on Nondeterministic Behavior** Rule 2 in the rewriting process above demands the construction of a set and the random pick of a member. This can be done via measuring $|x\rangle$ and constructing the set externally, then deciding which rule to apply after the random external choice. Note that this random process only models non-determinism but is not the same (since randomness is always associated with a probability distribution, where in true non-deterministic systems, nothing is known about the choice of rules).

## 4.2 Simulating Quantum Computers Using Semi-Thue Systems

### 4.2.1 Problems of a Simulation and General Procedure

The first problem we encounter when trying to model an arbitrary quantum computer by semi-Thue systems is the ambiguity of a single state of a quantum computer. Since several qubits can be in a superposition of base states, it seems to be hard to rewrite substrings, s.t. the transition of one state to another can be represented by a single rewriting step. Furthermore, transformations such as exchange of bits are possible within one single unitary transformation, meaning that we have to rewrite different bits at different positions inside the string at the same time. In deed, I was not able to find a semi-Thue system which models an arbitrary quantum computer. Let me first introduce my thoughts and then state why it seems complicated to create such a universal semi-Thue system.

Two make rewriting of non-adjacent qubits possible, we have to mark the qubits and allow them to permute, as it was done in previous sections, e.g. $|000\rangle = |0\rangle^{(0)}|0\rangle^{(1)}|0\rangle^{(2)}$. Now for every unitary transformation $U$, we can define a head which runs through the word, exchanging adjacent qubits according to the behavior of $U$. Since the current state can be a superposition, one has to do this with every term of the superposition. The problem now comes with transformations with brackets. Imagine to have transformation like

$$|0\rangle^{(0)}|0\rangle^{(1)}|0\rangle^{(2)}|0\rangle^{(3)}|0\rangle^{(4)}|0\rangle^{(5)}|0\rangle^{(6)} \to \\ |0\rangle^{(0)}|0\rangle^{(1)}\frac{1}{\sqrt{2}}\left(|0\rangle^{(2)}+|1\rangle^{(2)}\right)\frac{1}{\sqrt{2}}\left(|0\rangle^{(3)}+|1\rangle^{(3)}\right)\frac{1}{\sqrt{2}}\left(|0\rangle^{(4)}+|1\rangle^{(4)}\right)|0\rangle^{(5)}|0\rangle^{(6)}. \quad (89)$$

We would have to find general rules which allow to get the states from the left outside the brackets inside the first brackets, then multiplying the brackets, then getting the states from the right inside the last brackets. I could not think of an elegant and easy way to construct a head letter which is capable of going through the word and multiplying brackets without hard-coding every multiplication in the rules, which would be possible but not very elegant.

The general procedure of applying an arbitrary unitary transformation on a state is done in a semi-Thue system as follows

1. permute the qubits until they are in a certain position for rewriting at once
2. go through the word and rewrite the necessary qubits in every term of the state
3. rewind
4. bring all qubits from the left of possible brackets inside the brackets
5. evaluate the multiplication of brackets
6. bring all qubits from the right inside the brackets
7. perform the last three points for every term in the state
8. rewind and delete all brackets left.

In the end, there is another problem, because quantum states can be associated with real and even complex coefficients. However, we can limit our work to certain finite sets of interesting numbers, as indicated in the last section.

### 4.2.2 Semi-Thue Systems that Model the Grover Iteration

As seen, I was not able to construct a semi-Thue system which is able to simulate an arbitrary quantum computer. However, I managed to model the Grover iteration in a semi-Thue system in two different ways. In the following I introduce the first one, while the second one is downloadable without description.

To model the Grover iteration for searching a list of size $N$, we have to recapitulate the main parts of the algorithm. First of all we a have the initial state $|s\rangle$, which is the superposition of all possible bit strings. Furthermore we have one marked state $|\omega\rangle$ and two unitary operations $U_s$ and $U_\omega$ and one important constant $c = 2/\sqrt{N}$. The unitary operators act as

$$U_s|s\rangle = |s\rangle \tag{90}$$
$$U_s|\omega\rangle = c|s\rangle - |\omega\rangle \tag{91}$$
$$U_\omega|\omega\rangle = -|\omega\rangle \tag{92}$$
$$U_\omega|s\rangle = |s\rangle - c|\omega\rangle. \tag{93}$$

As we can see, the operators act like a string rewriter already. We are going to use that in the semi-Thue system. One iteration in the algorithm is achieved by letting the unitary operator $U_sU_\omega$ act on the current state.

We use a semi-Thue system consisting of the alphabet

$$\Sigma = \{U_\omega, U_s, |s\rangle, |\omega\rangle, -, (,), c, C, B, \S, \#, I\} \tag{94}$$

The initial word is given by

$$U_sU_\omega\S|s\rangle\# \qquad . \tag{95}$$

In the following I will write $x \to y$ for a rewriting rule $(x,y)$, because there are brackets in the alphabet. The starting rule is

$$U_\omega\S \to \S U_\omega \qquad . \tag{96}$$

Now the operators $U_\omega$ works as a "head" going through the word, replacing the states.

$$U_\omega|\omega\rangle \to (-|\omega\rangle)U_\omega \tag{97}$$
$$U_\omega|s\rangle \to (|s\rangle - c|\omega\rangle)U_\omega \tag{98}$$

However, they do not act on other symbols, so we add the permutation rule

$$U_\omega\sigma \to \sigma U_\omega \tag{99}$$

with $\sigma \in \{-, c, (,)\}$. Once at the end, the head has to be rewound

$$U_\omega\# \to C\# \tag{100}$$
$$\lambda C \to C\lambda \tag{101}$$
$$U_s\S C \to B\S U_s \tag{102}$$

with $\lambda \in \Sigma - \{\S\}$. Now the second replacement can be made.

$$U_s|s\rangle \to |s\rangle U_s \tag{103}$$
$$U_s|\omega\rangle \to (c|s\rangle - |\omega\rangle)U_s \tag{104}$$
$$U_s\# \to C\# \tag{105}$$

Now the tape can be rewound again. Once at the beginning, the $C$ will be replaced.

$$B\S C \to IU_sU_\omega\S \tag{106}$$

Now the whole system is in the state $IU_sU_\omega\S w\#$, with the quantum state being the substring $w$. Every time the system is in a state $nU_sU_\omega\S w\#$, the $n$-th operation has finished and the current quantum state is $w$ ($n$ is a natural number in unitary representation). The advantage of this system is that it can run indefinitely. A version which has been implemented in the language Thue can be downloaded[3]. Because we are interested in an iteration count of $r \propto \sqrt{N}$ with an optimal natural number $r$, we could modify the last mentioned rule to

$$mB\S C \to IU_sU_\omega\S \tag{107}$$

with $m < r$ being a natural number in the unitary representation and add the rule

$$rB\S C \to \S \tag{108}$$

[3]`http://people.physik.hu-berlin.de/~bfmaier/moc/grover2.t`

with $r$ being the optimal number of iterations in the unitary representation.

However, the execution is rather slow and copying the final substring $w$ into another program will end in a slow computation as well, since it has to multiply a lot of bracket terms. Another implementation[4] uses a terminal rule. The disadvantage of this method is that one has to hard-code multiplications of $c$ up to a terminating order $c^n$.

But there is another problem with these systems. Even though the probability amplitude of the state $|s\rangle$ can be neglected after enough iterations, in the end we will only have the state $|\omega\rangle$, which is just a description of the marked state, but not actually known. To get the information which state it actually is, we would physically have to measure it in the base system to yield an eigenstate or an eigenvalue. I could not think of a way to simulate this measurement.

## 5 Summary

In this paper we achieved a deeper insight in two very different models of computation. The classical model of semi-Thue systems uses string rewriting rules to compute functions. They represent the most general version of string rewriting systems and are connected to several other rewriting systems. We saw how semi-Thue systems relate to one of the most important classical models, the Turing machine, and investigated their origin in word problems.

The second model was the one of quantum computing, a model that uses quantum operators and states to act as a reversible machine. We saw how this model is related to classical reversible machines and investigated an example of the superiority of quantum algorithms with respect to classical models, which comes through a natural parallelism and the probabilistic character of quantum mechanics.

In the end, we connected both models. First, we constructed a quantum computing system which can model any semi-Thue system. Second, we gave indications on how to model a semi-Thue system which simulates an arbitrary quantum computer. Problems were that quantum computers can theoretically act on real numbers, but we overcame this restriction, the second problem was connected to the superposition of quantum systems which could not have been solved for arbitrary problems. However, we succeeded in constructing a semi-Thue system for a certain quantum algorithm.

To sum up, quantum computing seems to be a promising model of computation. It comes with the possibility to speed up classical algorithms and saves energy through reversibility. Future research will hopefully make clear if we can actually use it for computations or if it is just a beautiful theoretical model.

## References

[1] M. Fernández, “Models of Computation (An Introduction to Computability Theory)”, Springer, 2009

[2] A. Thue, “Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln”, Skr. Vid. Kritianaia, I. Mat. Naturv. Klasse, No. 10, 1914

[3] Y. Matiyasevich, G. Sénizergues, “Decision Problems for Semi-Thue Systems with a Few Rules”, Journal Theoretical Computer Science - Insightful theory archive Volume 330 Issue 1, 2005, pp. 145-169

[4] Z. Manna “Mathematical Theory of Computation”, McGraw-Hill, New York, 1974

[5] E. Post, “Recursive Unsolvability of a Problem of Thue”, The Journal of Symbolic Logic, Vol. 12, No. 1, (1947), pp. 1-11

[6] R. McNaughton, “Semi-Thue Systems with an Inhibitor”, Journal of Automated Reasoning, Volume 26, Number 4, 2001 , pp. 409-431(23)

[7] J. Colagioia, F. van der Plancke, “The Thue Programming Language”, `http://catseye.tc/projects/thue/`, last view: June 2012

[8] R. Weber, “Computability Theory”, Student Mathematical Library, v. 62, American Mathematical Society, 1977

[9] R. Zavodnik, “Introduction to Computer Science”, Chapter 6, `http://homepages.fh-regensburg.de/~zar39030/in/node7.html`, last view: June 2012

[10] G.E. Moore, “Cramming more components onto integrated circuits”, Electronics Magazine, 1965, p. 4.

[11] R. Feynman “Quantum Mechanical Computers”, Optics News, 1985, pp. 11-20.

[12] D. Deutsch, “Quantum theory, the Church-Turing principle and the universal quantum computer”, Proceedings of the Royal Society of London A 400, 1985, pp. 97-117

[13] B. Maier, “A Very Very Very Very$^k$ Short Introduction to Quantum Mechanics for Quantum Computing$_{\text{choose arbitrary } k\in\mathbb{N} \text{ adapted to your knowledge}}$”, `http://people.physik.hu-berlin.de/~bfmaier/files/introduction_to_qm.pdf`, last view: June 2012

[4]`http://people.physik.hu-berlin.de/~bfmaier/moc/grover.t`

[14] L.M.K. Vandersypen, M. Steffen, G. BreytaâĂă, C.S. Yannoni , M.H. Sherwood, I.L. Chuang, “Experimental realization of Shor’s quantum factoring algorithm using nuclear magnetic resonance”, 2001, arXiv:quant-ph/0112176v1

[15] J. Preskill, “Lecture Notes for Physics 229: Quantum Information and Computation”, California Institute of Technology, 1998

[16] A. Saenz, P.-I. Schneider, “Quantum computation with ultracold atoms in a driven optical lattice”, Physical Review A 85, 050304(R) (2012)

[17] R. Landauer, “Irreversibility and heat generation in the computing process”, IBM Journal of Research and Development, vol. 5, 1961, pp. 183-191,

[18] C.H. Bennett, “Logical reversibility of computation”, IBM Journal of Research and Development, vol. 17, no. 6, 1973, pp. 525-532

[19] F. Edward, T. Toffoli, “Conservative logic”, International Journal of Theoretical Physics (Springer Netherlands) 21 (3), 1982, pp. 219-253

[20] L. Fortnow, “One complexity theorist’s view of quantum computing”, Theoretical Computer Science 292, 2003, pp. 597-610

[21] W.K. Wootters, W.H. Zurek, “A Single Quantum Cannot be Cloned”, Nature 299, 1982, pp. 802-803

[22] L.K. Grover, “A fast quantum mechanical algorithm for database search”, Proceedings, 28th Annual ACM Symposium on the Theory of Computing, 1996, p. 212

[23] Wikipedia Foundation, “Grover’s algorithm”, `http://en.wikipedia.org/wiki/Grover%27s_algorithm`, last view: June 2012