Uncomputation and Entanglement in High-level Quantum Programming Languages

Roman Wixinger [†]
Semester project, Quantum Computing Group
Chair of Scientific Computing in Computer Science, TU Munich

Winter 2020/2021

Abstract

Quantum programming languages traditionally focus on the hardware level and are therefore not really good at representing the intentions of the programmer. Explicit formulation of uncomputation, which is essential for the safe and efficient use of the qubits, makes the code unnecessarily complex. In recent work, Vechev et al. (2020) introduced Silq, a high-level language that allows for safe, automatic uncomputation just using its type system. This feature makes the code significantly shorter and more intuitive. The type system can also ensure that any program that compiles is physical.

In this project, we compared Silq's solution of handling uncomputation with other approaches and give an overview of the features of quantum languages. We have also tried to understand whether a qubit can be safely discarded by directly looking at the entanglement.

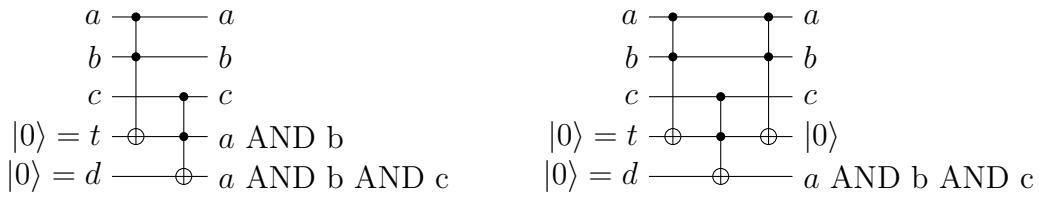
1 Introduction

The development of more powerful quantum computers enables longer and more complicated quantum algorithms to be created. Quantum programs are usually presented as pseudo code or in a real quantum programming language. Unfortunately, the latter are still unnecessarily focused on the hardware and cannot present the programmer's idea clearly.
The solution to this problem lies in the development of high-level quantum programming languages which, despite their abstraction, can ensure that they produce physical programs. This means, for example, that they do not violate the no-cloning theorem and do not allow implicit measurements.
We will now motivate the use of high-level features on the example of uncomputation. Assume we have three qubits a, b and c and we want to compute the combined AND, namely a AND b AND c and store the result in a qubit d that is initially set to zero. We can do so by storing the intermediate result of a AND b in an ancilla t[1] and then compute t AND c to get the desired result. This computation is analogue to the classical computation of x AND y AND z and the circuit is displayed in figure 1. However, in the quantum case we cannot simply forget the ancilla qubit t as we could in the classical case by deleting it. Simply taking it from consideration induces an implicit measurement, because we can deduce from the deferred measurement principle, that measuring the qubit directly would give the same probability distributions as measuring it in the end. As we could measure all non-ancilla qubits before the ancilla, measuring the ancilla

[†]Roman Wixinger, Physics student at ETH, supervised by Prof. Dr. Christian B. Mendl (TU Munich) and formally supervised by Prof. Dr. Renato Renner (ETH Zurich).

[1]The corresponding quantum gate is called Toffoli gate or controlled-controlled-not gate (CCNOT).

directly must be equivalent to not measuring it. As measuring the ancilla induces an implicit measurement on the other qubits, forgetting the ancilla is not a valid option. So instead we have to uncompute the ancilla, meaning that we reset it to zero with another unitary transformation. In this example, we simply apply another CCNOT gate with a and b as control and t as target to reset t to the zero state. The act of resetting the ancilla qubit to the zero state is called uncomputation.
Until recently, uncomputation had to be explicitly programmed. This changed when Vechev et al. (2020) introduced Silq, a high-level language that allows for safe, automatic uncomputation just using its type system. This feature makes the code significantly shorter and more intuitive. However, as with any type system, there are limitations to Silq in terms of what can be represented. In this work we also discuss this deviations and evaluate other methods for telling whether or not a qubit can be safely discarded that work directly with the entanglement.
The structure of the current work is the following: First we discuss the features of classical and quantum languages. Then we focus on the question, when a qubit can be safely discarded in the sections uncomputation and entanglement. We see how uncomputation and other quantum features are represented in quantum languages in the case studies section. We end the report with a discussion on the different approaches we have seen for handling uncomputation and ask the question, whether or not our question of discarding qubits safely is even computable.



(a) The intermediate result a AND b is stored in an ancilla qubit t.

(b) The ancilla qubit t is uncomputed after the result is duplicated to the results qubit d.

Figure 1: Example for uncomputation; Circuit for computing a AND b AND c of three qubits a, b and c [1].

2 Background

In this section we start with the classical features of programming languages and then discuss the additional features of quantum programming languages.

2.1 Classical features

Every programming language has certain types (i.e. int, float, string), expressions (i.e. if e then e_1 else e_2), annotations (i.e. const) and rules for typing judgements (i.e. $\tau + \tau = \tau$ for a type τ = int, float, string). A simple computational model for a classical programming language is the random-access machine (RAM). It has an unbounded sequence of registers as memory and a control unit that holds a program, which is a numbered list of statements. A program counter determines in each step which statement should be executed next [2].
In the classical case, an if-statement has a condition, also called control, that is either true or false. The condition cannot be something in between. There are no restrictions to the body of an if-statement. We will later see, that the quantum generalisation is not that straightforward.

Further we can have for-loops with a fixed number of iteration and while-loops where the number of iterations is to be determined during runtime. More abstract concepts like classes or functions can usually be decomposed into these elementary building blocks.

2.2 Quantum features

In the quantum case we have additional types that can be in superposition (i.e. represent qubits), expressions (i.e. measure, reverse) and annotations (i.e. mfree[2], qfree[3]).
We can extend our computational model of the RAM with quantum registers on which we can perform unitary operations and measurements. We call this machine a quantum random-access machine (qRAM) and say that a quantum program is physical if it runs on a qRAM without any errors [3].
We can generalise the classical if-statement to the quantum conditional, whose condition or control variable can be in a superposition [4]. For example, we can interpret the CNOT gate acting on qubits a, b as the quantum conditional `if` a $\{X(b)\}$ `else` $\{\}$. The control qubit can be in the state $\Psi_x = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, so that the total wavefunction becomes $\Psi_{xy} = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ when the b qubit is originally in state $\Psi_y = |0\rangle$. We will now see that there are many more possibilities for quantum conditionals than just the CNOT gate. In practise, instead of conditioning on a, we can also condition on Ua for a unitary U. As an example, we can condition on Xa to obtain an if-not-statement. In theory, we can perform the quantum conditional in an arbitrary orthonormal basis, as we can relate every two orthonormal bases with a unitary operator. For example, we can switch between the σ_z-basis $\{|0\rangle, |1\rangle\}$ and the σ_x-basis$\{|+\rangle, |-\rangle\}$ with Hadamard gates. We can also generalise the CNOT quantum conditional and control a unitary operation other than Pauli-X.
Another difference between classical and quantum computation lies in the persistence of its memory. While classical bits can be stored for almost any length of time, quantum bits suffer from quantum noise. Meaning on real hardware, they decohere within a very short time [5]. Additionally, quantum gates typically introduce larger errors than classical gates. This limits the number of gates that is available for computation and makes error correction necessary. Measurement also takes a substantial amount of time, which we have to consider in mixed programs that use partial measurement.
One surprising characteristic that we have to take into account is the deferred measurement principle. According to this principle, delaying or advancing measurements does not change the resulting probability distributions. For this reason, simply forgetting about qubits in our computation is equivalent to doing a measurement directly after we used them the last time.
One property that severely limits our possibilities in quantum computing is that all operators, except for measurements, are unitary. This means we have to write our functions in such a way that they are reversible [5]. Just to give an example of what is not possible, imagine that we have two qubits a and b and we want to store a AND b in b. The problem is that from the result (a, a AND b) we cannot deduce the initial value of b if a starts in the zero state. This means the computation is irreversible and thus cannot by implemented this way just using unitary operations. What we can do instead is storing the result of a AND b in an ancilla r that is initialised to zero [1]. We will continue our discussion on the features of quantum languages in the case studies section. There we will see different type systems, helper functions and ways to handle uncomputation.

[2] A function is mfree if it does not contain any measurement.
[3] A function is qfree if it does neither create nor destroy any quantum superposition.

3 Uncomputation

Using additional memory makes many calculations easier. For example the computation of x AND y AND z for three qubits x, y, z can be easily implemented with the circuit shown in figure 1 by using an ancilla qubit a that stores the result of x AND y [1]. Unfortunately, the number of qubits available to us is still very limited, both in real quantum hardware and in simulations.
In classical computation this issue is resolved by simply deleting scratch memory and reusing it. Doing so in quantum computation is a bad idea, because the ancilla qubits may be entangled with the other qubits. Measuring the ancillas and resetting them to zero with a Pauli $-$ X gate if necessary may disturb the state of the rest of the system.
Can we at least just forget about ancilla qubits when we no longer need them for our calculations? That is not possible either. According to the deferred measurement principle, delaying or advancing measurements does not change the resulting probability distributions [5]. This means that forgetting about the ancilla qubit until the end of the quantum circuit does not help as it is equivalent to directly measuring it.

3.1 Bennett's construction

An elegant way to reset the ancilla qubits to $|0\rangle$ is the following construction introduced by Bennett in 1973 [6, 7]. We start with some qubits in a general quantum state $|x\rangle$, ancilla qubits initialized to $|0\rangle$ and some results qubits initialized to $|0\rangle$. After we do our calculation that is represented by a unitary transformation C, we duplicate the qubits that represent the result $f(x)$. To reset the ancilla qubits to $|0\rangle$, we apply C^{-1}.

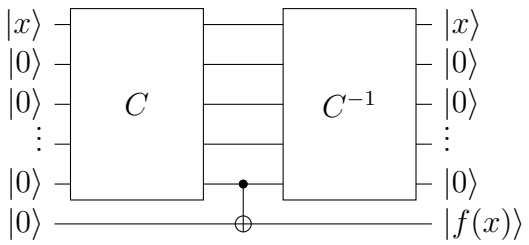


Figure 2: Circuit for Bennett's construction for uncomputing the ancilla variables [6, 7].

Here, duplicating the quantum state of a qubit is what we do when we use a CNOT gate with the qubit as a control and act on another qubit initialised to $|0\rangle$ as target.

$$\psi \otimes |0\rangle = (a\,|0\rangle + b\,|1\rangle) \otimes |0\rangle \mapsto a\,|00\rangle + b\,|11\rangle \tag{1}$$

On the other hand, cloning would be to map ψ to $\psi \otimes \psi$ for a general ψ. This is the kind of copying that is forbidden[4] by the no-cloning theorem.

$$\psi \otimes |0\rangle = (a\,|0\rangle + b\,|1\rangle) \otimes |0\rangle \mapsto \psi \otimes \psi = a^2\,|00\rangle + ab\,|01\rangle + ab\,|10\rangle + b^2\,|11\rangle \tag{2}$$

[4]The mapping that represents cloning must be unitary and thus be linear in particular. However, while the left-hand side of equation (2) is linear in a, the right-hand side is quadratic in a. This is a contradiction, to our assumption, that the mapping is linear. Note that for a, b equal to 0, 1 equation (2) reduces to equation (1) and thus we can copy classical quantum states.

In the interesting case in which C can be represented by a finite sequence of unitary gates, we can construct the inverse transformation C^{-1} by applying the adjoints of the gates in reverse order. We could naively think that this uncomputes the ancilla qubits, meaning it sets them to $|0\rangle$ in general. Unfortunately, this is not true for a general unitary C as we see in the example displayed in figure 3.

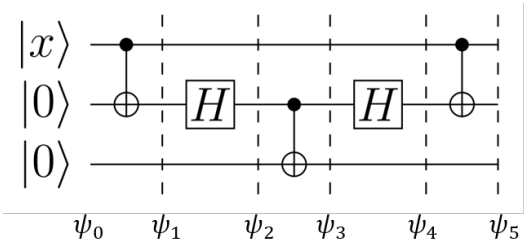


Figure 3: Circuit where Bennett's construction does not work. The intermediate quantum states are displayed in equation (3) to equation (8).

$$\psi_0 = a\,|000\rangle + b\,|100\rangle \tag{3}$$

$$\psi_1 = a\,|000\rangle + b\,|110\rangle \tag{4}$$

$$\psi_2 = \frac{a}{\sqrt{2}}(|100\rangle + |010\rangle) + \frac{b}{\sqrt{2}}(|100\rangle - |110\rangle) \tag{5}$$

$$\psi_3 = \frac{a}{\sqrt{2}}(|100\rangle + |011\rangle) + \frac{b}{\sqrt{2}}(|100\rangle - |111\rangle) \tag{6}$$

$$\psi_4 = \frac{a}{2}(|000\rangle + |010\rangle + |001\rangle - |011\rangle) + \frac{b}{2}(|100\rangle + |110\rangle - |101\rangle + |111\rangle) \tag{7}$$

$$\psi_5 = \frac{a}{2}(|000\rangle + |010\rangle + |001\rangle - |011\rangle) + \frac{b}{2}(|110\rangle + |100\rangle - |111\rangle + |101\rangle) \tag{8}$$

From equation (8), we see that the ancilla qubit is not reset to the $|0\rangle$ state, because the results qubit destroys the interference. The terms colored in red would cancel if the third qubit was not present.

3.2 Sufficient condition

Now we want to find a sufficient condition for Bennett's construction to work. Because the circuit is linear in the input state as a whole, we just have to understand what the circuit does to the basis states. In the following discussion, we introduce the notion of a classical state or function. We say that a state is classical if it is a computational basis state. We say that a function is classical if it maps computational basis states to computational basis states. Assume that the unitary transformation C in Bennett's construction is classical, then for each basis state, the result $f(x)$ is classical. When we duplicate the result $f(x)$ it remains classical, meaning it is either $|0\rangle$ or $|1\rangle$ but not in superposition. When we now apply C^{-1}, the results qubit will not destroy the interference and the uncomputation works, meaning that the ancilla is reset to zero. Because the transformation is linear, we have found the following sufficient condition for the uncomputation to work.

The unitary transformation C *does map computational basis states to computational basis states.*
We can easily construct an example to show that this condition is to strong. Consider a circuit $(C\otimes\mathbb{1})$ $(\mathbb{1}\otimes CNOT)$ $(C^{-1}\otimes\mathbb{1})$ where the condition is fulfilled. We add another qubit to it and apply two Hadamard gates, which gets us the circuit $(C\otimes\mathbb{1}\otimes H)$ $(\mathbb{1}\otimes CNOT\otimes\mathbb{1})$ $(C^{-1}\otimes\mathbb{1}\otimes H)$. This leaves the initial circuit unchanged. Clearly the uncomputation still works. However, when we consider the new circuit with $\tilde{C} = C \otimes H$, then $\tilde{C}$ does not fulfil our sufficient condition that it maps computational basis states to computational basis states.
We note that it is only the results qubit which destroys our interference when it is not in a classical state. In the last example, we do not duplicate the ancilla which we put in the $|+\rangle$ state and the uncomputation still works. Therefore, we can get a weaker sufficient condition that just makes sure that we only duplicate qubits in classical states for each computational basis state as input of the circuit. This can be formulated as
The function f(x) in Bennett's construction is classical.
Here, classical means that for every computational basis state x the function gives a value that can be understood classically; without superposition and entanglement. This brings us to a reformulation of the question, whether or not we can uncompute the ancilla qubits used in a circuit C. The new question is, whether or not the interesting part of the computation can be represented by a classical function $f(x)$. In the case where the circuit is made from gates that can be represented by classical functions, i.e. $\text{Pauli} - X$, CNOT, it is clear that also the full circuit can be represented by a classical function. This is because classical functions are just permutations of the basis states and the symmetric group, which represents these permutations, is closed under composition. But, there can be transformation that look like they introduce superposition, but actually they do not. For example the CNOT gate is just a permutation of the computational basis. But when we do the trick of switching control and target, we can find a representation of it that includes Hadamard gates. When we look at the representation in figure 4 it not apparent, that it would satisfy our condition.
In this chapter we started with the question, when it is possible to uncompute an ancilla qubit. After careful consideration of Bennett's construction we found that we can substitute this question with an approximate substitute question. Namely, is the function f(x) in Bennett's construction classical. One can think of clever rules to solve this decision problem. While the new formulation makes it easier to think about our original question, it is still true that we cannot make positive statements. As we have seen in figure 4, there are cases when non-classical gates lead to a circuit that can be represented classically. In the following chapter, we will work directly with the entanglement and see something similar.

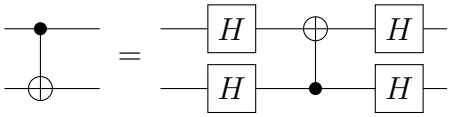


Figure 4: Representation of the CNOT gate.

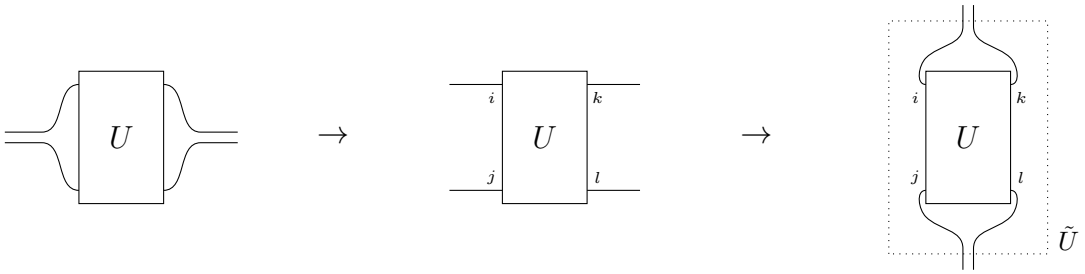
4 Entanglement

The hope for an exponential speedup of quantum computers compared to classical computers has been present since Feynman's ingenious observation; computing the time evolution of quantum systems on a classical computer requires exponentially more resources than the physical implementation [8]. Entanglement is one of the key features that separates quantum computers from classical ones. For a quantum algorithm operating on pure states, Jozsa and Linden showed that multi-partite entanglement is a necessary condition to allow for an exponential speedup of the quantum algorithm over a classical one [9]. Therefore we can view entanglement as a resource for computation. Recently, entanglement measures have sparked new interest in the research working with parameterized quantum circuits. When using such circuits in variational hybrid quantum-classical algorithms, one is often concerned that the circuit represents the solution space well. To compare circuits, Sim et al. (2019) proposed the use of the Meyer-Wallach entanglement measure to quantify the entanglement capability of the quantum circuits [10]. In our case, we are mainly interested in entanglement, because we hope to predict, whether or not some qubits may be safely discarded.
In this chapter we will start with the definition of bipartite entanglement and explain how a bipartite system can be tested for entanglement using SVD. Then we motivate discuss multipartite entanglement on the example of the W and GHZ state, giving the reader an idea of how complicated the problem becomes. After discussing an ansatz for keeping track of the entanglement, we refer to methods for quantifying multipartite entanglement that were proposed just recently. To end the chapter, we will discuss the question whether or not entanglement is predictable at all.

4.1 Bipartite entanglement

We consider two quantum systems A and B with Hilbert spaces H_A and H_B, respectively. Using the bases $\{|i\rangle_A\}_i$ and $\{|j\rangle_B\}_j$, we can write any vector in the Hilbert space $H_A \otimes H_B$ of the composite system AB as $|\Psi\rangle_{AB} = \sum_{i,j} c_{ij} |i\rangle_A |j\rangle_B$. We define a state $|\Psi\rangle_{AB}$ to be separable iff there exists states $|\Psi\rangle_A = \sum_i c_i^A |i\rangle_A$ and $|\Psi\rangle_B = \sum_j c_j^B |j\rangle_B$ so that $|\Psi\rangle_{AB} = |\Psi\rangle_A \otimes |\Psi\rangle_B$. Otherwise, we say that the state $|\Psi\rangle_{AB}$ the is entangled on the bipartite system AB.
Separability is equivalent to the existence of a matrix c with $c_{ij} = c_i^A \; c_j^B$ [11].
A useful feature of the bipartite case is that we can check for entanglement using the Schmidt decomposition, which is a reformulation of the singular value decomposition (SVD) in the context of quantum mechanics. The corresponding theorem says that for every pure state $|\Psi\rangle_{AB}$, there exists orthonormal bases $\{|i\rangle_A\}_i$ and $\{|j\rangle_B\}_j$ so that $|\Psi\rangle_{AB}$ can be written as $|\Psi\rangle_{AB} = \sum_i \lambda_i |i\rangle_A |i\rangle_B$. The Schmidt co-efficients λ_i are non-negative numbers that satisfy $\sum_i \lambda_i^2 = 1$. We define the Schmidt number to be the number of non-zero Schmidt co-efficients. If the Schmidt number is one, the Schmidt decomposition has the form of a product state and the state is separable. For a Schmidt number bigger than one, the state is entangled [5].
Next we discuss an ansatz for telling whether or not an unitary U acting on $H_A \otimes H_B$ may entangle a product state. For this discussion, we recommend the reader to have the CNOT gate in mind, in which case H_A and H_B are just single qubit Hilbert spaces. We interpret U as four legged tensor U_{ij}^{kl} where i,j are the indices of the input space and k,l the indices of the output space, standing for H_A, H_B, respectively. For example, we can write the CNOT gate as $\text{CNOT} = |00\rangle_{\text{A'B'}} \langle 00|_{\text{AB}} + |01\rangle_{\text{A'B'}} \langle 01|_{\text{AB}} + |11\rangle_{\text{A'B'}} \langle 10|_{\text{AB}} + |10\rangle_{\text{A'B'}} \langle 11|_{\text{AB}}$ and interpret the indices of each term as $|kl\rangle_{A'B'} \langle ij|_{AB}$. To make it even more explicit, we differentiate between the input systems AB and the output systems A'B'. We see that the input indices appear together

and the output indices appear together, meaning that U is split between input and output.



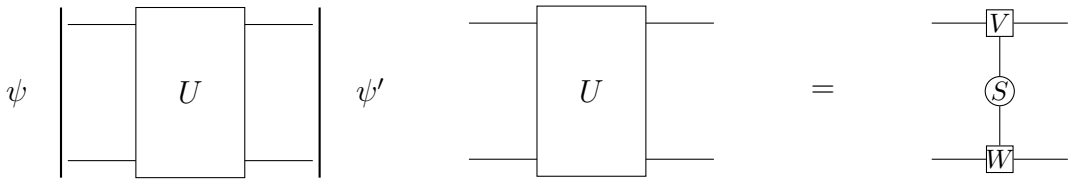
(a) Graphical representation of the new interpretation of U as $\tilde{U}$ [12].

The idea of our ansatz is now to split U between the A and the B system, which results in U_{lj}^{ki}. The upper and lower indices now represent functions on A and B, respectively. With the CNOT gate, we use the basis elements $\{\Omega_k^C\}_k = \{|0\rangle_{C'}\langle 0|_C,\ |0\rangle_{C'}\langle 1|_C,\ |1\rangle_{C'}\langle 0|_C,\ |1\rangle_{C'}\langle 1|_C\}$ for $C \in \{A, B\}$ and write the gate as in equation (9).

$$\text{CNOT} = |0\rangle_{\text{A}'}\langle 0|_{\text{A}}\left(|0\rangle_{\text{B}'}\langle 0|_{\text{B}} + |1\rangle_{\text{B}'}\langle 1|_{\text{B}}\right) + |1\rangle_{\text{A}'}\langle 1|_{\text{A}}\left(|0\rangle_{\text{B}'}\langle 1|_{\text{B}} + |1\rangle_{\text{B}'}\langle 0|_{\text{B}}\right) \tag{9}$$

$$\text{CNOT} = \Omega_0^{\text{A}}(\Omega_0^{\text{B}} + \Omega_3^{\text{B}}) + \Omega_3^{\text{A}}(\Omega_2^{\text{B}} + \Omega_3^{\text{B}}) \tag{10}$$

Now we can contract the indices of the CNOT gate in equation (10) an interpret it as a two legged tensor. One leg represents $\{\Omega_k^A\}_k$ and the other $\{\Omega_k^B\}_k$. We can now apply a SVD and see if it is possible to make the split between the two indices. In the case of the CNOT gate, we get two non-zero singular values and therefore it is not possible to write the gate as a cross product of two single qubit gates. This means that the CNOT gate may entangle the two qubits [13].



(a) A general unitary U acting on a bipartite system.

(b) Singular value decomposition of U.

Figure 6: Two representations of the same unitary U as a four legged tensor. The input ψ appears on the left since circuit diagrams are read from left to right. The horizontal lines are associated with quantum systems, and can be interpreted as tensor legs. With the CNOT gate, a leg represents a single qubit [12].

Now we have seen an ansatz, for telling whether a unitary U acting on a bipartite system may entangle the two systems if we start from a product state. For a formal discussion of the entangling power of a two-qubit gate we refer the reader to the work of Guan et al. (2014) and Shen and Chen (2018) [14, 15].

4.2 Multipartite entanglement

We will now discuss multipartite entanglement on the example of the three qubit W and GHZ state. Dür et al. showed in 2008 that three qubits can be entangled in two inequivalent ways [16]. As it is sufficient for our purpose, we restrict ourselves to a giving the reader an idea, not having in mind to summarise or represent their work. In the bipartite case, we could think of the amount of entanglement between systems A and B as the amount of information we gain about system B if we measure system A. For example the quantum state $|\psi\rangle_{AB} = \frac{|0\rangle+|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ is not entangled, because measuring A will give $|\psi\rangle_B = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and thereby not reveal anything about B. On the other hand $|\text{GHZ}\rangle_{AB} = \frac{|00\rangle+|11\rangle}{\sqrt{2}}$ will collapse the wavefunction of B to $|\psi\rangle_B = |0\rangle$ or $|\psi\rangle_B = |1\rangle$ depending on the result of measuring A. The number of possible outcomes of measuring B has gone from 2 to 1 and therefore we have gained information about B from measuring A. The two qubits were entangled.
Now let us consider a tripartite system ABC consisting of single qubit systems A, B and C. There are two maximally entangled states, namely the W state $|\text{W}\rangle = \frac{1}{\sqrt{3}}(|001\rangle + |010\rangle + |100\rangle)$ named after Wolfgang Dür and the GHZ state $|\text{GHZ}\rangle = \frac{|000\rangle+|111\rangle}{\sqrt{2}}$ named after Daniel Greenberger, Michael Horne and Anton Zeilinger. We will now argue that these states are not entangled in the same way. As in the bipartite case, we consider what happens when we measure the A qubit. In the case of the GHZ state, measuring the subsystem A also tells us in which state the subsystems B and C are, as the wavefunction is reduced to either $|\psi\rangle_{BC} = |00\rangle$ or $|\psi\rangle_{BC} = |11\rangle$ for the result 0 or 1, respectively. The number of possible outcomes for measuring the system BC has been reduced from two to one, independent of the outcome of the measurement of A. On the other hand, when we measure the subsystem A of the W state, we either get $|\psi\rangle_{BC} = |00\rangle$ if the outcome is 1 or $|\psi\rangle_{BC} = \frac{|01\rangle+|10\rangle}{\sqrt{2}}$ if the outcome is 0. That means in the case of the GHZ state, it is sufficient to measure one qubit to determine the state of all three subsystems, while in the case of the W state we need to measure two qubits, to know the state of the third one.
For completeness we finish this discussion with the main result of Dür et al. (2008). They used invertible local transformations of a multipartite system to define equivalence classes in the set of entangled states. The equivalence relation says that two states are related, iff they have the same kind of entanglement. That means iff both states can be obtained from each other by means of local operations and classical communication (LOCC) with non-zero probability. With this definition, they show for the case of a three qubit system, that there are two inequivalent kinds of genuine tripartite entanglement. The GHZ state and the W state are representatives of these. Moreover, they noticed that the loss of a particle, which can be modeled by tracing out one qubit, has a different effect on the entanglement of the remaining system. If one of the qubits of the GHz state is traced out, the remaining two qubit system is completely unentangled. On the other hand, the W state is special among the three qubit states, because it loses the least of its entanglement element with the loss of a particle [16].
We end our general discussion of multipartite entanglement by referring the reader to the work of Eisert and Briegel (2008) who introduced a way to quantify multipartite entanglement. Their entanglement monotone is called the Schmidt measure and as the name suggests, it is a generalization of the Schmidt rank earlier introduced in this chapter [17].

4.3 Predicting entanglement

In the following we first consider a rudimentary way to predict if certain subsystems are not entangled. Then we carry out an argument of Prost and Zerrari (2009) according to which the question of entanglement cannot be decided in general.
We will try to define an equivalence relation $\sim$ on the set of qubits A with $a \sim b \Leftrightarrow$ a is entangled with b, where a, b $\in$ A. Before we discuss why this fails, we mention what we tried to achieve with this relation. Any equivalence relation on a set corresponds to a partition of the set. In our case, we group the set of qubits into subsets of entangled qubits. Assuming an initial partition and considering a quantum program, we can keep track of operations which may introduce entanglement between the partitions. Even though we cannot predict at certain, if two qubits are entangled in the end, we can still predict some cases when they are certainly not. This is useful information for the programmer.
So let us check, if $\sim$ fulfils the three requirements of an equivalence relation. Clearly $\sim$ is reflexive, as measuring a will yield information about a. It is also symmetric, as the Schmidt number is independent of the ordering of the two systems. Last, we have to check that $\sim$ is transitive. If we make the connection to the classical case, that is to say classical correlation, then we already see that we might run into problems. For classical variables X, Y and Z, we can have the case that X and Y are correlated, as well as Y and Z. For example, we can imagine the case where Z is a function of X and Y, i.e. z = x + y. If we sample these three variables, then X and Y are not correlated, unless we impose a constraint on Z. So the classical correlation relation is not an equivalence relation, as it is not transitive.
In the quantum case, the entanglement of a bipartite system AB that is part of a larger system ABC is more complicated, as there are two options for measuring the entanglement. First, a necessary and sufficient condition to test the entanglement of the systems A, B for a pure state Ψ_{ABC} is to check if $\mathrm{Tr_C}(|\Psi_{\mathrm{ABC}}\rangle \langle\Psi_{\mathrm{ABC}}|)$ is not positive under the partial transpose map [18, 19]. To show that in this case the entanglement relation is not transitive, one can apply the criterion on three qubit state $|\Psi\rangle_{ABC} = \frac{1}{\sqrt{2}}(|000\rangle + |11\phi\rangle)$ and get a counter-example. Alternatively, we could measure the system C and deal with the different cases separately. For example, the GHZ state $|\mathrm{GHZ}\rangle = \frac{|000\rangle+|111\rangle}{\sqrt{2}}$ collapses to an unentangled product state if we measure one of the qubits in the computational basis. Furthermore, we can consider the Coffman-Kundu-Wootters (CKW) monogamy inequality given in equation (11) which relates the concurrence of a tripartite system. The concurrence C_{AB} is an entanglement monotone, meaning that it measures the degree of entanglement between the subsystem A and B [20].

$$C^2_{AB} + C^2_{AC} \leq C^2_{A(BC)} \tag{11}$$

The take-away message here is the following. Because there is a trade-off between A's entanglement with B and its entanglement with C, the ansatz with the entanglement relation is somehow problematic, as it cannot capture this important constraint. There might be more suitable relations, which in the best case, are even transitive. Because the entanglement relation is not transitive, we cannot use it to make positive statements on the entanglement of qubits. What we can do is to classify the gates and operations regarding whether or not they may entangle qubits, Then we can construct partitions of qubits that may be entangled after applying a quantum program by simply using one of the algorithms that was invented for finding partitions of a graph. If two qubits end up in different partitions, they are certainly not entangled. However, the opposite is not true.

4.4 Feasibility

To continue our discussion about the feasibility of predicting entanglement, we will develop an argument by Prost and Zerrari (2009) in more detail [21]. They noticed that separability is not computable, meaning that it is not possible to find a program which predicts for every quantum algorithm whether or not it entangles two systems. The idea of their argument is that such program could be used to decide the halting problem. This is the problem of determining, for an arbitrary computer program and an input, whether the program will finish running, or continue to run forever. However, Alain Turing (1936) has proven that such a program cannot exist [22]. We start our proof by contradiction by assuming that we have a program $h_q(j, a)$ which tells us for each quantum program i whether or not it will terminate for input a. We define it as given in section 4.4.

$$h_q(j,a) = \begin{cases} 1 & \text{if quantum program i produces entanglement in a,} \\ 0 & \text{otherwise.} \end{cases}$$

We now construct a classical analogue $h(i, x)$ from $h_q(j, a)$ which solves the halting problem. Let's take an arbitrary program i with an input x. We concatenate the input x with to qubits q_1 and q_2 to an input $a = (x, q_1, q_2)$. We add a last line to program j that entangles the qubits q_1 and q_2 and define the resulting problem i. Now the program $h(i, x) \coloneqq h_q(j(i), a(x, q_1, q_2)$ decides the halting problem. This is a contradiction, which is why $h_q(j, a)$ cannot exist.
To finish this chapter, we make the link to uncomputation. Our motivation to predict the entanglement of a quantum system after applying a program was that we wanted to tell if we can safely forget about certain qubits. In this chapter, we now saw that making the prediction based on considerations on the entanglement may be very complicated, when not impossible. An easier and more elegant way for this will be discussed in the next chapter.

5 Case studies

In the following section we are going to discuss how different quantum programming languages handle uncomputation. Along the way we present features that are specific to quantum programming languages. We will discuss the languages Silq [23], Q# [24], Quipper [25] and **Q**WIRE [26]. One of the first quantum languages, QCL [27], also happens to be a good example for a transparent allocation and uncomputation of ancillas and we leave it here as a reference.

5.1 Silq

In short, Silq[5] can achieve the following with its type system. All programmes that compile are physical. Ancilla variables are automatically and safely uncomputed. This leads to shorter and more readable code. Silq comes with a proof-of-concept simulator and a development environment in VS Code [23].
The type system of Silq is well described in the original paper and we present here only the options for expressions (e), types (τ), annotations (α, β) and typing judgements (Γ) in figure 7, figure 8 and figure 9, respectively [23].

[5] https://silq.ethz.ch/

$$e \ ::= \ c \mid x \mid \textbf{measure} \mid \textbf{reverse} \mid \textbf{if } e \textbf{ then } e_1 \textbf{ else } e_2 \mid e'(e_1, ..., e_n) \mid \lambda(\beta_1 x_1 : \tau_n, ..., \beta_n x_n : \tau_n).e \tag{12}$$

Figure 7: Silq-core expressions, namely constants and built-in functions, variables, measurement, reverse functions, conditionals and lambda abstraction.

$$\tau \ ::= \ \mathbb{1} \mid \mathbb{B} \mid \bigtimes_{k=1}^{n} \tau_k \mid \bigtimes_{k=1}^{k} \beta_k \tau_k! \xrightarrow{\alpha} \tau' \mid !\tau \tag{13}$$

Figure 8: Silq-core types, namely numbers, booleans, combinations of types, functions and their classical counterparts which are marked with an exclamation mark (!) [23].

$$\alpha \ \subseteq \ \{\textbf{mfree}, \textbf{qfree}\} \tag{14}$$

$$\beta \ \subseteq \ \{\textbf{const}\} \tag{15}$$

$$\Gamma \ ::= \ \beta_1 x_1 : \tau_1, ..., \beta_n x_n : \tau_n \tag{16}$$

$$\Gamma \underset{\alpha}{\vdash} e : \tau \tag{17}$$

Figure 9: Silq-core annotations and typing judgements [23].

In the following we will focus on the connection between Silq's annotations and the condition we derived for uncomputation in section 3. Functions, whose semantics can be described classically, are annotated as **qfree**. These are the functions that map basis states to basis states, such as the X-gate. On the other hand, the Z-gate and Hadamard gate are not qfree. We annotate functions as **mfree** if they do not contain measurements. Function arguments are annoted with **const** if they are unchanged by the function and are still available after the function call. This is to say that they are not consumed. Silq makes sure, that all variables are either const or that they are consumed in the function. This is necessary to prevent the following actions: A) cloning, i.e. using the same variable multiple times B) performing an implicit measurement, i.e. dropping a variable from consideration and therefore directly measuring it. When a function is both **qfree** and all arguments are **const**, then we call it **lifted** [23].
All ancilla qubits, which appear in lifted functions, can automatically be uncomputed. We can understand this if we consider that such a function has a classical result for each classical input of the const function arguments. This means that the conditions are fulfilled that we could use Bennet's construction to copy the result with CNOT. Fortunately, the condition is also sufficient for allowing uncomputation if the inputs are not classical, but in superposition. We can consider that qfree functions are linear isometries, that is, they are linear in particular. So if Bennett's construction works for each base state as a function argument individually, then it also works for the superposition.
In the section on uncomputation, we saw that the question, whether or not uncomputing is possible, boils down to telling whether or not the function $f(x)$ is classical. The problem was that telling this is non trivial, as we saw in the example with the representation of the CNOT gates in figure 4. In the type system of Silq we find a similar problem. The compiler of Silq tells whether or not a function is qfree based on the annotations of the functions which are used inside the function. For this reason, Silq does not get the constructed example given in figure 10. Unfortunately, this kind of construction where we first put the qubits into superposition

with Hadamard gates, do something, and then apply Hadamard gates is pretty common. The construction occurs, for example, in the Bernstein–Vazirani algorithm, displayed in figure 11 [28].

```
1   def CNOT(x:𝔹, y:𝔹) qfree {
2       x := H(x);
3       y := H(y);
4
5       if y {
6           x := X(x);
7       }
8
9       x := H(x);
10      y := H(y);
11
12      return (x,y);
13  }
```

cannot call 'mfree' function 'H' in 'qfree' context

Peek Problem (Alt+F8) No quick fixes available

Figure 10: The type system of Silq is too restrictive in this case. While the reader knows, that the function CNOT is qfree, Silq cannot deduce this property just using the type system.

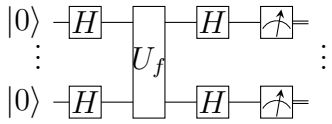


Figure 11: Bernstein–Vazirani algorithm [28].

In the design of type systems there is always a trade off between making it more precise annd making it easy to understand. To make it more precise, we can add rules and cover special cases, so the user can reach the bounds of what is possible. To make it easier we can leave out rules let the user create a workaround, for example by annotating the extra knowledge of the user or by using a coerce statement.
A special characteristic of the annotation qfree is that it depends on the basis we choose for our computations. For example the X operator[6] in the basis $\{|0\rangle, |1\rangle\}$ does the same to the basis vectors as the Z operator[7] does to the basis vectors in the basis $\{|+\rangle, |-\rangle\}$. But the X operator in the $\{|0\rangle, |1\rangle\}$ is clearly classical, whereas the Z operator is not. The phase of π, $e^{i\pi} = -1$ in front of $\mathrm{Z}|1\rangle = -|1\rangle$ cannot be described classically. This means X is qfree and Z is not. So we can argue that the annotation qfree is basis dependent. This dependence originally motivated us to look directly at the entanglement to find a basis independent view.
Last, we think about reversing functions in Silq. As all quantum operations except measurement represent linear isometries, no two unequal inputs can be mapped to the same output, as otherwise angles would not be preserved. This means the operations are injective and thus invertible on their image. We can obtain the inverse of an mfree function f in Silq by typing **reverse**(f), but we only obtain safe functions if f is also surjective and thus bijective. Otherwise, the programmer has to ensure, that **reverse**(f) is only used on the image of f. This is another useful application of the annotation mfree, which prevents the programmer from hiding measurements inside functions and then trying to reverse them. As the typing judgement would get that the function is not mfree, **reverse** would not accept it [23].

[6] $\mathrm{X}|0\rangle = |1\rangle$, $\mathrm{X}|1\rangle = |0\rangle$, $\mathrm{X}|+\rangle = |+\rangle$, $\mathrm{X}|-\rangle = -|-\rangle$
[7] $\mathrm{Z}|0\rangle = |0\rangle$, $\mathrm{Z}|1\rangle = -|1\rangle$, $\mathrm{Z}|+\rangle = |-\rangle$, $\mathrm{Z}|-\rangle = |+\rangle$

5.2 Q#

While many quantum programming language such as Quipper and QWire are focused on defining quantum circuits, Q#[8] was designed for defining algorithms that consist of both quantum and classical parts [24]. Q# comes with multiple simulators that model the quantum computing under different assumptions. While the full state simulator actually stores the full quantum state and thus can only simulate fewer than a dozen qubits, the trace simulator does not have to store the state and thus works for a few thousand qubits. Quantum programs that only use X, CNOT and multi-controlled X gates, can be simulated by the Toffoli simulator even with millions of qubits. The trace simulator and the resources estimator can be used to estimate different metrics[9].
We will quickly present the Q# types, which consists of the classical primitives, quantum primites, collections, operations, functions and user-defined types. The classical data types a Int, Double, Boolean and String. As quantum primitives there are Pauli, representing the identity and the three Pauli operators, Result, representing the result of a measurement and Qubit. Collections can be arrays and tuples and their elements can be of any Q# type. In Q# we make the distinction between routines that can affect the quantum states and ones that cannot. While operations can affect the quantum state, functions by definition do not change the quantum and only perform classical computation [24].
In Q#, qubits and ancilla qubits are both allocated with the "using"-statement. Ancilla qubits have to be returned into state $|0\rangle$ before releasing them at the end of the using block. Just ignoring the ancilla qubits may lead to unwanted behaviour, as they may still be entangled with the other qubits. In contrast to Silq, Q# requires the programmer to do the uncomputation by hand. While performing the uncomputation is usually not a difficult task, it still generates a lot of unnecessary code, which obscures the big picture of the algorithm. A feature of Q# that is also related to uncomputation is the "borrowing"-statement. Assume we have a subroutine that uses ancilla qubits temporarily and can ensure that those ancilla qubits are returned to the exact same state at the end. If we have another subroutine that runs in parallel and does not perform any actions on a part of the qubits, this qubits can be used as "dirty ancillas" and borrowed by the subroutine that needs them [24].

5.3 Quipper

Quipper is an embedded functional programming language that is geared towards a model of computation where a classical computer controls a quantum device. It is a high-level language and focused on scalability, meaning it can generate representation of quantum circuits that consist of trillions of gates [25].
Many quantum languages are based on a quantum circuit model that only consists of unitary gates and circuits. In Quipper, this circuit model a larger class of possible operations, which we will discuss in the following. Among quantum bits, the type system also contains classical bits, classical gates, classically controlled quantum gates and measurements. To represent the **scope** of ancillas and qubits in general, Quipper offers explicit **qubit initialization** and **qubit termination**. The compiler of Quipper explicitly keeps track of the scope of ancilla qubits. This

[8]The quantum language Q# is one tool of Microsoft's Quantum Development Kit, which is available at https://docs.microsoft.com/de-de/azure/quantum/. A comprehensive documentation is available at https://docs.microsoft.com/en-us/azure/quantum/user-guide/.

[9]This metrics include the run count of CNOT gates, Clifford gates, R gates, T gates and measurements. The depth, width and borrowed width can also be estimated. A documentation is available at https://docs.microsoft.com/en-us/azure/quantum/user-guide/machines/resources-estimator.

helps the compiler to know which ancillas are currently free[10] and to optimise which ancillas are used. In the hardware agnostic quantum program it does not play a role, which free ancilla we use in a subroutine. However, because there are platforms where gates can not be applied to arbitrary pairs of qubits, the choice of which ancilla to use may be a valuable degree of freedom for the compiler. Quipper also offers **assertive termination**, which is to assume that a certain qubit ends in the $|0\rangle$ or the $|1\rangle$ state. This is in contrast to ordinary termination, where a qubit is simply not considered anymore, possibly resulting in a mixed state. Note that the compiler cannot verify the assertions automatically. It is the sole responsibility of the programmer to only make correct assertions. The special feature of circuits that contain qubit initialization and assertive termination is that they do not produce mixed states. In a suitable subspace the circuits are unitary and thus reversible. This subspace is simply the subspace on which the assertions are correct. The reversibility can be used in Quipper in the following form. Any circuit that contains the same number of input and output qubits and an arbitrary number of local ancillas, can be reversed by Quipper, even if it contains qubit initialization and assertive termination. This is possible because the circuit is unitary [25].
The last feature of Quipper that we want to focus on is the automatic generation of quantum oracles[11]. The definition of such an oracle can be performed in four steps and all except the first one can be automated by Quipper. First, we express the oracle as a classical function on classical data types. Second, we translate this function to a classical circuit. Third, we transform the classical circuit to a quantum circuit. For this step we may need ancilla qubits to store intermediate values, i.e. when we compute operations such as AND. Fourth and last, we make the circuit reversible with the standard trick by replacing a function f(x) by $(x,y) \mapsto (x,y \oplus f(x))$ and uncomputing any ancillas used by the function f. While step two and three are performed with the Quipper operation **build_circuit**, which is also referred to as circuit lifting. This operation can be performed for classical functions made of millions of gates. The fourth step, namely the uncompuation, can be automated in Quipper with the function **classical_to_reversible**. This kind of helper function is characteristic for Quipper[12]. We see that uncomputation is performed explicitly with a function call. This is in contrast to Silq, where the uncomputation is not only automatic, but also implicit.

5.4 QWIRE

QWIRE[13] is a language for small quantum circuits and is used embedded in a larger, functional programming language. It has a purely linear type system to ensure type safety. In **Q**WIRE, circuits are explicitely separated from the arbitrary classical host language [26].
Here, we make the reader aware of the extension *Re***Q**WIRE. The main problem that *Re***Q**WIRE solves concerns uncomputation and it is the following. In many languages, for example Quipper, the programmer has to return the ancilla qubits to the $|0\rangle$ state before they are discarded and assert, that this has been successful. However, this is an unsafe assumption and because the programmer is not given any tools to check the assumption, it is a potential source of errors. In their original paper, Rand et al. (2019) worked out methods for verifying that ancilla qubits are discarded correctly and introduced *Re***Q**WIRE [26, 29].

[10] A free ancilla is one that is in the $|0\rangle$ state. By assumption we cannot distinguish different free ancillas.

[11] A quantum oracle is an unknown operation. We can think of it as a black box. Quantum oracles are often used as an input of another quantum algorithm, for example in Grover's algorithm, which also known as quantum search algorithm. More information can be found at https://docs.microsoft.com/en-us/azure/quantum/concepts-oracles.

[12] More examples can be found at https://hackage.haskell.org/package/quipper-core-0.8.0.1/docs/Quipper.html.

[13] Related papers and an implementation can be found at https://github.com/inQWIRE/QWIRE.

6 Discussion

In this chapter we will compare the different approaches for performing uncomputation.

6.1 Explicit vs. implicit uncomputation

We start with the advantages and disadvantage of making uncomputation explicitly by hand like in Q# or implicitly using automation like in Silq. The main advantages of the explicit formulation are that the programmer is aware of the qubit usage and that extra knowledge can be build into the uncomputation. For example, if an ancilla qubit ends in a computational basis state, and the programmer knows, that it is not used again, then the uncomputation is unnecessary. It is not entangled with the rest of the system and measuring it in the computational basis is safe. On the other hand, letting the programmer do the uncomputation by hand or with helper functions is a potential source of errors. Also, the explicit formulation does not add to the readability of the code. Uncomputation corresponds to garbage collection in classical computation and in most high-level classical languages we do make the deletion of variable explicit for this very reason.
In contrast, the implicit formulation of creating and dropping ancilla variables that Silq uses leads to shorter and more readable code. In the case of Silq, the compiler can check that the program is physical and that the ancilla variables are uncomputed correctly. This feature is very useful, because the programmer is reminded when something is not possible at the time of writing the code, making it possible to build an intuition for quantum programming. As we all agree, that quantum mechanics is not intuitive at all, making a language intuitive like Python is definitely something we should be aiming for.

6.2 Uncomputation vs. entanglement tracking

We already discussed explicit and implicit uncomputation and now we want to compare this approaches with keeping track of the entanglement. Using implicit, automatic uncomputation like Silq is certainly a beautiful approach but sometimes to restrictive. Refining the type system to reach the boundaries may not be possible with a reasonable small set of typing rules. However, one could overcome this imperfectness by letting the user annotate extra knowledge. On the other hand, trying to predict the entanglement is hard, especially when we do not restrict the gate set as it was done in the Toffoli simulator by Microsofts Quantum Development Kit. If we want to provide the user with extra knowledge, then there is certainly information that is also useful but easier to compute. In the case of entanglement, there might be even cases where the entanglement is not predictable without simulating the circuit. In the end of our conclusion we discuss some features that we would like to see in an integrated development environment (IDE) for quantum languages.

7 Conclusion

The main contributions of this work are the following. First, we gave an overview of the features of high-level quantum languages. Second, we provide a detailed discussion of uncomputation and how it is handled in existing quantum languages. Third, we searched for other methods to tell if we may forget a qubit. For this we looked at different ways to quantify entanglement.
We saw that the safest and most intuitive way to ensure that ancillas are correctly uncomputed, is the approach of Silq using its type system. We believe that the rare cases, where the type system is to restrictive, can be avoided or fixed by the programmer in useful time.

It turned out, that predicting if an ancilla can be discarded safely by looking at the entanglement seems to be harder. Simple approaches of bookkeeping cannot grasp the nature of the entanglement of the qubits. If one would like to further investigate this approach, we would suggest to search for circuit manipulations that do not change the entanglement but make the circuit easier to simulate. We think about the ZX-calculus here and the Toffoli simulator, which can simulate large circuits which are restricted to X, CNOT and multi-controlled X gates. One could also investigate the link between the quantum eraser proposed by Scully and Drühl (1982) and uncomputation [30].
In the future we would like to see IDEs and compilers that provide the programmer with extra information, similar to the autocompletion tool IntelliSense[14] by Microsoft for classical programming languages or Kite[15] for the Spyder IDE. The Silq compiler already warns the user about operations which would make the program nonphysical and when functions are annotated incorrectly. A programming assistant could further propose simplifications and inform the user about useful statistics like qubit usage. Variables that are not used anymore could be colour-coded and error correction on them could be omitted.
It remains unclear whether quantum programming will be intuitive at some point in the future. However, quantum languages that are physical by definition and programming environments that support the programmer are certainly important steps in this direction.

References

[1] Florian Neukart und Sylvester Tremmel. Qubits aus seide. *Heise Magazin*, c't 20/2020:146, 2020.

[2] Bader. Random access machine. Fundamental Algorithms class, TU Munich, 2002.

[3] E Knill. Conventions for quantum pseudocode. *Office of Scientific & Technical Information Technical Reports*, 6 1996.

[4] Alessandro Bisio, Michele Dall'Arno, and Paolo Perinotti. Quantum conditional operations. *Physical Review A*, 94(2):022340, 2016.

[5] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information. Cambridge University Press, 2002.

[6] Charles H Bennett. Logical reversibility of computation. *IBM journal of Research and Development*, 17(6):525–532, 1973.

[7] Charles H Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM journal on Computing*, 26(5):1510–1523, 1997.

[8] Richard P Feynman. Simulating physics with computers. *Int. J. Theor. Phys*, 21(6/7), 1982.

[9] Richard Jozsa and Noah Linden. On the role of entanglement in quantum-computational speed-up. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 459(2036):2011–2032, Aug 2003.

[14] A description of the features is available at https://code.visualstudio.com/docs/editor/intellisense.
[15] A description is available at https://www.kite.com/integrations/spyder/.

[10] Sukin Sim, Peter D. Johnson, and Alán Aspuru-Guzik. Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms. *Advanced Quantum Technologies*, 2(12):1900070, Oct 2019.

[11] Ingemar Bengtsson and Karol Zyczkowski. A brief introduction to multipartite entanglement. *arXiv preprint arXiv:1612.07747*, 2016.

[12] Christian Mendl. Tensor networks class (in2388), tutorial 4. 2021.

[13] Christian Mendl. Personal communication, December 2020.

[14] Zhe Guan, Huan He, Yong-Jian Han, Chuan-Feng Li, Fernando Galve, and Guang-Can Guo. Entangling power of two-qubit gates on mixed states. *Phys. Rev. A*, 89:012324, Jan 2014.

[15] Yi Shen and Lin Chen. Entangling power of two-qubit unitary operations. *Journal of Physics A: Mathematical and Theoretical*, 51(39):395303, aug 2018.

[16] Wolfgang Dür, Guifre Vidal, and J Ignacio Cirac. Three qubits can be entangled in two inequivalent ways. *Physical Review A*, 62(6):062314, 2000.

[17] Jens Eisert and Hans J. Briegel. Schmidt measure as a tool for quantifying multiparticle entanglement. *Phys. Rev. A*, 64:022306, Jul 2001.

[18] Kay Alastair. Is entanglement transitive? Quantum Computing Stack Exchange, May 2018.

[19] G. Vidal and R. F. Werner. Computable measure of entanglement. *Phys. Rev. A*, 65:032314, Feb 2002.

[20] Valerie Coffman, Joydip Kundu, and William K. Wootters. Distributed entanglement. *Phys. Rev. A*, 61:052306, Apr 2000.

[21] Frédéric Prost and Chaouki Zerrari. Reasoning about entanglement and separability in quantum higher-order functions. In *International Conference on Unconventional Computation*, pages 219–235. Springer, 2009.

[22] A. M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 01 1937.

[23] Benjamin Bichsel, Maximilian Baader, Timon Gehr, and Martin Vechev. Silq: A high-level quantum language with safe uncomputation and intuitive semantics. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2020, page 286–300, New York, NY, USA, 2020. Association for Computing Machinery.

[24] Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. Q# enabling scalable quantum computing and development with a high-level dsl. In *Proceedings of the Real World Domain Specific Languages Workshop 2018*, pages 1–10, 2018.

[25] Alexander S Green, Peter LeFanu Lumsdaine, Neil J Ross, Peter Selinger, and Benoît Valiron. Quipper: a scalable quantum programming language. In *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*, pages 333–342, 2013.

[26] Jennifer Paykin, Robert Rand, and Steve Zdancewic. Qwire: A core language for quantum circuits. *SIGPLAN Not.*, 52(1):846–858, January 2017.

[27] Bernhard Ömer. Classical concepts in quantum programming. *International Journal of Theoretical Physics*, 44(7):943–955, Jul 2005.

[28] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on computing*, 26(5):1411–1473, 1997.

[29] Robert Rand, Jennifer Paykin, Dong-Ho Lee, and Steve Zdancewic. Reqwire: Reasoning about reversible quantum circuits. *arXiv preprint arXiv:1901.10118*, 2019.

[30] Marlan O. Scully and Kai Drühl. Quantum eraser: A proposed photon correlation experiment concerning observation and "delayed choice" in quantum mechanics. *Phys. Rev. A*, 25:2208–2213, Apr 1982.