Release of Silq: A High-level Quantum Language

Contact Information: Prof. Martin Vechev, ETH Zurich, Switzerland, silq@inf.ethz.ch

Background. Recent efforts have improved quantum computers to the point where they can outperform classical computers on some tasks, a situation referred to as quantum supremacy. Quantum computers run quantum algorithms, typically expressed in a low-level quantum language.

Silq. We release Silq, the first high-level quantum language designed to abstract from low-level implementation details of quantum algorithms. Silq is publicly available at GitHub (https://github.com/eth-sri/silq) and licensed under the free and open-source Boost Software License 1.0.
As a key novelty, Silq helps to bridge the conceptual gap between classical and quantum languages. As a consequence, Silq (i) lowers the entrance barrier for non-expert quantum programmers, (ii) generally helps to express complicated algorithms cleanly and concisely, and (iii) facilitates a tech-transfer of 50+ years of programming and analysis techniques developed for classical computing, to the domain of quantum programming.

Comparison. While traditionally quantum algorithms were often specified in terms of circuits, quantum languages express quantum algorithms more conveniently as source code. However, existing quantum languages force the programmer to work at a low level of abstraction, still essentially specifying quantum circuits that explicitly apply quantum operations to individual quantum bits. As a consequence, implementing quantum algorithms in these languages is tedious and error-prone.
In contrast, Silq supports a descriptive view on quantum algorithms that expresses the high-level intent of the programmer. Then, the compilation of these algorithms to low-level quantum circuits becomes a second-order concern that can be handled by a specialized compiler, much like in classical programming languages.

Our experimental evaluation demonstrates that Silq programs are significantly shorter than equivalent programs in other quantum languages (on average -46% for Q#, -38% for Quipper), while using only half the number of quantum primitives. Thus, Silq programs are not only shorter, but also easier to read and write, as they require less primitives and concepts.
Most of over evaluation focused on Q#, because (i) it is one of the most widely used quantum languages, (ii) we consider it to be more high-level than Cirq or QisKit, and (iii) the Q# coding contests from 2018 and 2019 provide a large collection if Q# implementations we were able to leverage for our comparison.

Team. Silq was developed by Benjamin Bichsel, Maximilian Baader, Timon Gehr, and Prof. Martin Vechev, who are all part of the SRI Lab, a research group at ETH Zurich's Department of Computer Science.

Resources. We provide various resources to get started with Silq.

- Silq's webpage (http://silq.ethz.ch/) offers a detailed introduction of Silq's key concepts, a demonstration of Silq's benefits over Q#, concrete examples of Silq algorithms, instructions on installation, and much more.
- To facilitate programming in Silq, we provide an extension for Visual Studio Code, which provides a convenient interface for type-checking and simulating Silq code.
- Silq's parser, type checker, and simulator are publicly available at GitHub (https://github.com/eth-sri/silq) and licensed under the Boost Software License 1.0.
- We provide a detailed description of Silq's language design in our publication about Silq at the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2020).

Technical Details

Key Benefit: Automatic Uncomputation. The key benefit of Silq is that it automatically (without intervention from the programmer) handles *uncomputation* of temporary values. In the following, we explain the necessity of uncomputation, and the benefits of automating it.

Analogously to the classical setting, quantum computations often produce temporary values. However, as a key challenge specific to quantum computation, removing such values from consideration induces an *implicit measurement* collapsing the state. In turn, collapsing can result in unintended side-effects on the state due to the phenomenon of *entanglement*. Surprisingly, due to the quantum principle of *deferred measurement*, preserving values until computation ends is equivalent to measuring them immediately after their last use, and hence cannot prevent this problem.

To remove temporary values from consideration without inducing an implicit measurement, algorithms in existing languages must explicitly uncompute all temporary values, i.e., modify their state to enable ignoring them without side-effects. In particular, even dropping the result of a sub-expression like `x||y` in `(x||y)||z` requires explicitly applying gates that uncompute this value, essentially forcing the programmer to perform manual memory management.

This results in a significant gap from quantum to classical languages, where discarding temporary values typically requires no action (except for heap values not garbage-collected). This gap is a major roadblock preventing the adoption of quantum languages as the implicit side-effects resulting from uncomputation mistakes, such as silently dropping temporary values, are highly unintuitive.

In contrast, Silq is comparable to high-level languages like Rust, in that it automatically handles uncomputation in a safe way. Among other benefits, this allows us to naturally support nested expressions, as known from classical languages.

Key Technique: A Quantum Type System. Silq introduces a novel *quantum type system* which exploits a fundamental pattern in quantum algorithms, stating that uncomputation can be done safely if (i) the original evaluation of the uncomputed value can be described classically, and (ii) the variables used to evaluate it are preserved and can thus be leveraged for uncomputation.

Silq's quantum type system allows it to detect this pattern, ensuring uncomputation is possible. Moreover, Silq also detects when uncomputation is not possible. In this case, Silq reports an error—before even running the algorithm. As a consequence, all Silq algorithms are physical: they can be realized on a quantum computer, and never attempt physically impossible operations like invalid uncompuation or reversing a measurement.

Future Benefits. We expect Silq to advance various tools central to programming quantum computers. First, Silq compilers may require fewer quantum bits, e.g., by more flexibly picking the time of uncomputation. Second, static analyzers for Silq can safely assume (instead of explicitly proving) temporary values are discarded safely. Third, Silq simulators may improve performance by dropping temporary values instead of explicitly simulating uncomputation. Finally, Silq's key language features are relevant beyond Silq, as they can be incorporated into existing languages such as QWire or Q#.

Origins of Silq. Originally, we did not set out to create a quantum language—our goal was to analyse quantum algorithms. Instead, we realized that existing quantum languages cannot express quantum algorithms at a convenient level of abstraction, and decided to focus our efforts on this problem instead. Since then, we have worked on Silq for almost two years, and are proud of the result we are presenting now.