Gesellschaft für Informatik e.V. (GI) GI. (Hrsg.): INFORMATIK 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 1825

Holistische Verifikation von Hybriden Quantenprogrammen durch Software Bounded Model Checking

Jonas Klamroth[1], Max Scheerer[1], Oliver Denninger[1]

Abstract:

Quantencomputer erschließen uns durch ihren überpolynomiellem Speedup neue Anwendungsfelder für schwer-berechenbare Probleme. Der Entwurf von Quantenalgorithmen ist bisher allerdings komplex und fehleranfällig. Daher ist zu erwarten, dass vorerst nur einzelne Subroutinen eines Programms auf Quantencomputern umgesetzt werden. Um die Korrektheit solcher Programme garantieren zu können, sind neue Ansätze erforderlich. In dieser Arbeit stellen wir einen Ansatz zum vollautomatischen Nachweis der Korrektheit von Programmen mit eingebetteten Quantenalgorithmen vor. Dazu bauen wir auf Bounded-Model-Checking-Verfahren auf, welche die Fehlerfreiheit hinsichtlich einer gegebenen Spezifikation beweisen können. Als Spezifikationssprache verwenden wir JML. Dabei werden die Quantenalgorithmen als Quantenschaltkreis beschrieben und in Java eingebettet. Wir zeigen die Umsetzbarkeit unseres Ansatzes an zwei etablierten Quantenalgorithmen.

Keywords: Bounded Model Checking; Quantencomputing; Verifikation; hybride Quantenprogramme

1 Einleitung

Das Potential von Quantencomputern ist nicht mehr nur theoretischer Natur, sondern es sind bereits verschiedene Quantenalgorithmen für praktische Problemstellungen bekannt: z.B. der *Grover*-Algorithmus für die unstrukturierte Suche in Datenbanken oder der *Shor*-Algorithmus zur Faktorisierung von natürlichen Zahlen. Der Shor-Algorithmus erzielt einen überpolynomiellen Speedup und ist somit deutlich effizienter als jeder bekannte klassische Algorithmus. Während der Shor-Algorithmus sich stark auf Kryptographie und somit auf IT-Sicherheit auswirkt, adressieren andere Quantenalgorithmen andere Anwendungsgebiete.

Obwohl Quantencomputer einen immensen Speedup erzielen können, kommen Quantenalgorithmen in der Praxis bisher kaum zum Einsatz. Dies ist maßgeblich auf hardwareseitige Einschränkungen zurückzuführen, die sogenannte *Noisy Intermediate-Scale Quantum* (NISQ) Computer mit sich bringen, z.B. geringe Dekohäranzzeit und hohe Gatterfehlerraten. Aus diesem Grund haben sich sogenannte *Variational Quantum Algorithms* (VQA) etabliert [Ce20], die zuverlässiger auf NISQ-Computern ausgeführt werden können. Ein VQA besteht aus einem parametrisierten Quantenschaltkreis, dessen Parameter auf einem klassische Rechnern iterativ optimiert werden, bis eine akzeptable Lösung, der durch den

[1] FZI Forschungszentrum Informatik, Haid-und-Neu-Straße 10-14, 76131 Karlsruhe, Deutschland
klamroth/scheerer/denninger@fzi.de

Quantenschaltkreis codierten Funktion, erreicht ist. Wie alle Quantenalgorithmen können VQA nur in Kombination mit klassischen Programmteilen verwendet werden, da immer zumindest eine Vor- sowie Nachverarbeitung nötig ist [LB20]. Ein System, bestehend aus klassischem sowie Quantenprogrammteil, wird als *hybrides System* bzw. *hybrides Quantenprogramm* bezeichnet.

Ein weiterer Hinderungsgrund für den Einsatz von (hybriden) Quantenalgorithmen bildet ihr inhärent hoher Grad an Komplexität. Quantencomputer nutzen quantenmechanische Effekte, wie z.B. *Quantenverschränkung*, *Superposition* oder *Messungen* von *Quantenzuständen*. Somit kann das Implementieren von Quantenprogrammen nicht mit konventionellen Programmen verglichen werden, da Quantencomputer ein völlig neues Paradigma mit sich bringen. Entsprechend schwer ist auch das Debuggen und Testen, so dass formale Verifikation eine noch bedeutendere Rolle bei der Qualitätssicherung einnimmt [Ba20].

In dieser Arbeit stellen wir einen Ansatz zum vollautomatischen Nachweis der Fehlerfreiheit von Programmen mit eingebetteten Quantenalgorithmen hinsichtlich einer gegebenen Spezifikation vor, indem die Quantenalgorithmen in Java-Code übersetzt werden. Unser Ansatz ist auf das Schaltkreismodell für Quantencomputing anwendbar. Dabei werden Algorithmen durch Schaltkreise beschrieben, die wiederum aus einer Anordnung von Gattern bestehen. Jedes Gatter manipuliert dabei ein oder mehrere Qbits. Nachfolgend stellen wir die notwendigen mathematischen Grundlagen vor. Für eine ausführlichere Einführung in Quantencomputing verweisen wir auf [NC10].

2 Grundlagen Quantencomputing

Ein Qbit kann, in Gegensatz zu einem klassischen Bit, nicht nur in den Zuständen 1 oder 0 sein, sondern auch in einer beliebigen Superposition davon. Um dies auszudrücken, werden Qbits als Elemente in $\mathbb{C}^2$ beschrieben. Qbits werden typischerweise in der Dirac-Notation wie folgt dargestellt: $\alpha\,|0\rangle + \beta\,|1\rangle$, mit $|0\rangle = (1\ 0)^T$, $|1\rangle = (0\ 1)^T$ und $\alpha, \beta \in \mathbb{C}$. Dabei gilt immer $|\alpha|^2 + |\beta|^2 = 1$. Der Zustand eines Systems mit mehreren Qbits kann als Tensorprodukt der einzelnen Zustände dargestellt werden. Dies ist im Allgemeinen umgekehrt nicht der Fall. Als Beispiel betrachte man folgenden Zustand: $\sqrt{2}^{-1}\,|00\rangle + \sqrt{2}^{-1}\,|11\rangle$. Hierfür lässt sich keine äquivalente Zerlegung in Zustände einzelner Qbits finden. Solche Zustände nennt man "verschränkt", d.h. die Manipulation eines einzelnen Qbits verändert den Zustand des gesamten Systems. Die Manipulation eines Qbits geschieht durch unitäre Transformationen. Da wir immer nur endlich viele Qbits betrachten, kann eine solche Transformation als Matrix ausgedrückt werden. Dabei gilt: Eine Matrix M ist unitär, wenn: $MM^\dagger = M^\dagger M = I$ wobei I die Identität ist und $M^\dagger$ die komplex konjugierte transponierte Matrix von M.

Um den Zustand eines Qbits beobachten zu können, muss auf einem Quantencomputer eine Messung des Qbits durchgeführt werden. Eine solche Messung hat zwei Effekte: Sie liefert als Ergebnis ein klassisches Bit (abhängig vom Zustand des Qbits) und das Qbit kollabiert. Konkret bedeutet das für ein Qbit in Zustand $|\phi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$: Die Wahrscheinlichkeit

$|0\rangle$ zu messen beträgt $|\alpha|^2$ bzw. $|\beta|^2$ für $|1\rangle$. In jeden Fall kollabiert $|\phi\rangle$ in den gemessenen Zustand. Das bedeutet insbesondere, dass eine Messung den Zustand des Systems verändert.

3 Ansatz zur Verifikation hybrider Quantenprogramme

Die Grundidee unseres Ansatzes ist die Einbettung von Quantenschaltkreisen in eine klassische Programmiersprache (in unserem Fall Java) basierend auf einer entsprechenden Übersetzung. Ein Java-Programm mit Quantensubroutinen kann somit in ein reines Java-Programm übersetzt werden und dadurch wie ein klassisches Programm spezifiziert, getestet und verifiziert werden. Der Vorteile dieses Ansatzes ist, dass durch die Einbettung des Quantenteils der Software in den klassischen Teil die Komplexität für Entwickler drastisch reduziert wird. Sowohl Spezifikation als auch Verifikation erfolgen analog zu klassischer Software. Auch die Möglichkeit Tests für den entsprechenden Quantenalgorithmus zu schreiben und Debuggingmöglichkeiten ergeben sich hierdurch. Darüber hinaus funktioniert unser Ansatz vollautomatisch, so dass kein quantenmechanisches Detailwissen nötig ist. Die Übersetzung basiert auf der Darstellung von Quantenzuständen als Arrays von Fließkommazahlen. Diese werden im Laufe des Programms entsprechend des gegebenen Quantenschaltkreises manipuliert. Zur Umsetzung müssen zwei grundlegende Operationen ermöglicht werden: unitäre Transformationen und Messungen. Transformationen lassen sich als Matrixmultiplikation abbilden und können somit direkt als Summe bzw. Produkt von Fließkommazahlen ausgedrückt werden. Es sei kurz erwähnt, dass die Anwendung eines 1-Qbit-Gatters (bspw. Hadamard-Gatter) auf einen Zustand mit mehr als einem Qbit erreicht werden kann, indem mit Hilfe des Tensorprodukts und der Identitätsmatrix eine äquivalente Operation auf alle Qbits angewendet wird.

Der spannendere Aspekt ist die Umsetzung von Messungen (in unserem Fall ausschließlich in der Standard-Basis). Messungen lassen ein Qbit in den gemessenen Zustand kollabieren. Für einen Zustand mit mehr als einem Qbit bedeutet dies, dass die Wahrscheinlichkeit für alle Zustände, in denen das Qbit nicht den gemessenen Zustand annimmt, auf 0 fällt. Betrachten wir als Beispiel den 2-QBit-Zustand: $\alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$. Messen wir in diesem Zustand eine 1 für das erste Qbit, so ergibt sich der Zustand: $\frac{1}{\sqrt{\gamma^2+\delta^2}}(\gamma|10\rangle + \delta|11\rangle)$. Der vorgestellte Faktor normalisiert den Zustand, sodass der resultierende Vektor wieder Länge 1 hat. Wir unterstützen zwei Varianten für Messungen: klassische Messung und Maximumsmessungen. Beide sind praktisch motiviert: Während die klassische Messung alle möglichen Messergebnisse für die Verifikation betrachtet, wird bei der Maximumsmessung ausschließlich das wahrscheinlichste Ergebnis der Messung weiter verfolgt. Dies hat den Grund, dass Quantenalgorithmen oftmals das gewünschte Ergebnis nicht mit Sicherheit, allerdings mit hoher Wahrscheinlichkeit liefern. Mit dieser Art der Modellierung können wir Aussagen für das wahrscheinlichste Ergebnis des Algorithmus beweisen. Umgesetzt werden beide Messvarianten durch eine Fallunterscheidung. Im klassischen Fall wird als Unterscheidungskriterium ein nichtdeterministisches Bit genutzt und somit beide Fälle betrachtet. Bei der Maximumvariante wird hingegen die Summe

der beiden Wahrscheinlichkeiten für das entsprechende Bit als Unterscheidungskriterium herangezogen.

4 Fallstudien

Um unseren Ansatz zu evaluieren haben wir zwei typische Beispiele für Quantenalgorithmen mit der vorgestellten Technik spezifiziert und verifiziert. Die Übersetzung von der Gatterdarstellung des Quantenalgorithmus in den entsprechenden Java-Code wurde mit Hilfe eines von uns entwickelten Werkzeugs automatisch ausgeführt. Als Verifikationswerkzeug wurde JJBMC [Be20] genutzt, welches (mit JBMC [Co18] als Backend) für die Verifikation von mit JML [Le08] spezifiziertem Java-Code entwickelt wurde.

Die erste Fallstudie ist der Deutsch-Algorithmus. Gegeben eine Funktion $f : \{0, 1\} \rightarrow \{0, 1\}$, entscheidet der Deutsch-Algorithmus, ob diese Funktion konstant ist oder nicht. Hierfür benötigt der Algorithmus nur eine einzige Evaluation der Funktion. Mit unserem Ansatz waren wir in der Lage vollautomatisch in weniger als einer Sekunde zu zeigen, dass die Implementierung des Algorithmus für jede möglich Funktion das erwartete Ergebnis berechnet. Als zweite Fallstudie wurde das Quantenteleportationsprotokoll betrachtet. Hierbei wird gezeigt, dass es möglich ist, einen Quantenzustand, alleine mit der Übertragung zwei klassischer Bits zwischen zwei Parteien, zu teleportieren. Auch in diesem Fall konnte mit unserem Ansatz gezeigt werden, dass durch das Protokoll der Quantenzustand tatsächlich "teleportiert"wird. Übersetzung und Verifikation erfolgten mit geringem manuellen Aufwand.

Die Fallstudien zeigten allerdings auch zwei kritische Punkte für das vorgestellte Verfahren auf. Als erstes ist anzumerken, dass wir durch den Einsatz von Fließkommazahlen anstatt reeller Zahlen eine natürlich Fehlerquelle durch Rundungsfehler in Kauf nehmen. In den vorgestellten Fallstudien haben diese Rundungsfehler keine entscheidenden Probleme verursacht. Es ist jedoch anzunehmen, dass diese sich bei größeren Quantenprogrammen potenzieren würden und somit gegebenenfalls die Aussagen der Verifikation in Frage stellen könnten. Dies führt auch zum zweiten Nachteil unseres Ansatzes: Skalierbarkeit. Die vorgestellten Fallstudien kamen mit maximal drei Qbits aus und bereits hier konnten wir exponentielle Aufwände für sowohl Übersetzung als auch Verifikation beobachten. Die Übersetzung ist aktuell als Proof-of-Concept umgesetzt und enthält keine Optimierungen. Entsprechend kann die Skalierbarkeit unseres Ansatzes aktuell noch nicht abschließend bewertet werden.

5 Verwandte Arbeiten

In der Literatur finden sich mehrere Ansätze, um beweisbare Aussagen über Quantenprogramme zu treffen. Wir unterscheiden hierbei zwischen Ansätzen, die rein theoretischer

Natur sind und solchen, die Werkzeuge und somit Automatisierungsmöglichkeiten vorstellen. Auf theoretischer Seite finden sich mehrere Logiken wie beispielsweise LQP [BS06] (eine dynamische Logik für Quantenprogramme), EEQPL [CMS06] (angelehnt an die klassische Hoare-Logik) oder eine vollständige Floyd-Hoare-Logik für eine Quanten-While-Sprache [Yi12]. Im Why3-Framework können mit Hilfe von parametrisierten Pfadsummen teilautomatisierte Beweise über Quantenschaltkreise geführt werden [Ch21]. Die Schaltkreise werden dabei in einer DSL (QBricks) beschrieben und mit Hilfe einer weiteren DSL (QBricks-spec) spezifiziert. Bemerkenswert ist hierbei, dass Schaltkreise mit variabler Größe und unbeschränkt vielen Qbits betrachtet werden können. Ein weiterer Ansatz wurde für den Theorembeweiser Coq vorgestellt [PRZ17]. Auch hier wird das Schaltkreismodell als Grundlage genutzt und die Sprache für die Beschreibung der Programme heißt Qwire. Qwire nimmt außerdem eine abstrakte Hostsprache an, mit der zusammen das Quantenprogramm ausgeführt wird. Dieser Ansatz ist allerdings deutlich weniger automatisiert als der zuvor beschriebene. Auch einen Model Checker für Quantenprogramme gibt es bereits [GNP08]. Spezifikationen werden hierbei in QCTL geschrieben. Allerdings wird hier die Integration in eine Hostsprache nicht betrachtet.

6 Zusammenfassung und weitere Arbeiten

Wir haben gezeigt, dass hybride Software spezifiziert und verifiziert werden kann, indem die Quantenalgorithmen in den klassischen Softwareteil eingebettet werden. Die Reduktion auf die Verifikation klassischer Software hat hierbei den Vorteil, dass gewohnte Sprachen und Praktiken unverändert übernommen werden können. Als weitere Arbeiten sehen wir vor allem die formale Beschreibung der Übersetzung und damit verbunden einen formalen Korrektheitsbeweis. Außerdem sollen in Zukunft andere Hostsprachen (Übersetzungen in andere Sprachen als Java) sowie andere Werkzeuge zur Verifikation untersucht werden. Zusätzlich sollen weitere Fallstudien durchgeführt werden, um den Einsatz auch für komplexere Algorithmen zu testen.

Die Arbeit erfolgte im Projekt SEQUOIA, gefördert durch das Wirtschaftsministerium Baden-Württemberg.

Literatur

[Ba20] Barbosa, L. S.: Software Engineering for 'Quantum Advantage'. In: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20). ACM, S. 427–429, 2020.

[Be20] Beckert, B.; Kirsten, M.; Klamroth, J.; Ulbrich, M.: Modular Verification of JML Contracts Using Bounded Model Checking. In: Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles, International Symposium on Leveraging Applications of Formal Methods (ISoLA 2020). Springer, S. 60–80, 2020.

[BS06] Baltag, A.; Smets, S.: LQP: The Dynamic Logic of Quantum Information. Mathematical Structures in Computer Science 16/3, S. 491–525, 2006.

[Ce20] Cerezo, M.; Arrasmith, A.; Babbush, R.; Benjamin, S. C.; Endo, S.; Fujii, K.; McClean, J. R.; Mitarai, K.; Yuan, X.; Cincio, L.; Coles, P. J.: Variational Quantum Algorithms, 2020, arXiv: 2012.09265 [quant-ph].

[Ch21] Chareton, C.; Bardin, S.; Bobot, F.; Perrelle, V.; Valiron, B.: An Automated Deductive Verification Framework for Circuit-building Quantum Programs. In: Programming Languages and Systems, European Symposium on Programming (ESOP 2021). Springer, S. 148–177, 2021.

[CMS06] Chadha, R.; Mateus, P.; Sernadas, A.: Reasoning About Imperative Quantum Programs. Electronic Notes in Theoretical Computer Science, Proceedings of the 22nd Annual Conference on Mathematical Foundations of Programming Semantics (MFPS XXII) 158/, S. 19–39, 2006.

[Co18] Cordeiro, L.; Kesseli, P.; Kroening, D.; Schrammel, P.; Trtik, M.: JBMC: A bounded model checking tool for verifying Java bytecode. In: Computer Aided Verification, International Conference on Computer Aided Verification (CAV 2018). Springer, S. 183–190, 2018.

[GNP08] Gay, S. J.; Nagarajan, R.; Papanikolaou, N.: QMC: A Model Checker for Quantum Systems. In: Computer Aided Verification, International Conference on Computer Aided Verification (CAV 2008). Springer, S. 543–547, 2008.

[LB20] Leymann, F.; Barzen, J.: The bitter truth about gate-based quantum algorithms in the NISQ era. Quantum Science and Technology 5/4, S. 044007, 2020.

[Le08] Leavens, G. T.; Poll, E.; Clifton, C.; Cheon, Y.; Ruby, C.; Cok, D.; Müller, P.; Kiniry, J.; Chalin, P.; Zimmerman, D. M. et al.: JML reference manual, 2008.

[NC10] Nielsen, M. A.; Chuang, I. L.: Quantum Computation and Quantum Information. Cambridge University Press, 2010.

[PRZ17] Paykin, J.; Rand, R.; Zdancewic, S.: QWIRE: A Core Language for Quantum Circuits. In: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017). ACM, S. 846–858, 2017.

[Yi12] Ying, M.: Floyd–Hoare Logic for Quantum Programs. ACM Transactions on Programming Languages and Systems 33/6, 19:1–19:49, 2012.