The Journal of Open Source Software

Enabling the Verification and Formalization of Hybrid Quantum-Classical Computing with OpenQASM 3.0 compatible QASM-TS 2.0

Sean Kim[*1] **and Marcus Edwards**[†2]

2 University of British Columbia **1** George Washington University

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

The unique features of the hybrid quantum-classical computing model implied by the specification of OpenQASM 3.0 motivate new approaches to quantum program verification. We implement and thoroughly test a QASM 3.0 parser in TypeScript to enable implementations of verification and validation software, compilers and more. We aim to help the community to formalize the logic of hybrid quantum-classical computing by providing tools that may help with such efforts.

Top Level Abstractions

The parser implements recursive descent parsing with support for expression precedence and type checking. At a high level, the parsing can be split into three logical sections: expression parsing, quantum-specific parsing, and classical parsing.

Expression parsing forms the foundation for both quantum and classical parsing by breaking down expressions into their constituent parts and rebuilding them according to the mathematical and operator syntax defined in the OpenQASM 3.0 official grammar (*OpenQASM 3.0 Grammar*, n.d.). Starting with basic elements like numbers, variables, and operators, the parser constructs an abstract syntax tree that reflects proper operator precedence and nesting. This process handles not only mathematical operations but also array accesses, function calls, and parameter lists, creating a structured representation that maintains the logical relationships between all parts of the expression.

Quantum operation parsing manages the core quantum computing elements of QASM, ensuring that quantum operations are syntactically correct. The parser maintains strict tracking of quantum resources through its *gates*, *standardGates*, and *customGates* sets, validating that each quantum operation references only properly defined gates and qubits. It processes quantum register declarations, custom gate definitions, gate applications (including gate modifiers), measurement operations, and timing-critical operations like barriers and delays. Each quantum operation is validated in its context.

The classical parsing processes classical variable declarations with strict type checking, function definitions with parameter and return type validation, control flow structures like conditional statements and loops, and array operations. The parser maintains separate tracking for classical and quantum resources while ensuring they can interact in well-defined ways. This enables QASM programs to express complex quantum algorithms that require classical processing while maintaining type safety and operational validity.

*skim658@gwu.edu
†msedward@student.ubc.ca

The Journal of Open Source Software

Our parser produces a strongly-typed AST that captures the full structure of QASM programs and is designed to enable subsequent semantic analysis.

Statement of Need

The OpenQASM 3.0 type system supports classical and quantum types as well as functions which can be used to specify hybrid quantum-classical programs.

There is need for formal analysis and verification of hybrid quantum classical programs and we argue that mathematical frameworks and software frameworks are needed to address this gap.

Hand-in-hand with formalization efforts are the development of community standards for quantum programming. The standardization of quantum computer programming is ongoing (Cross et al., 2022; Di Matteo et al., 2024) and relies significantly on open source software and frameworks including some released only last year (Seidel et al., 2024; Wille et al., 2024). Some community standards such as the 2022 Open Quantum Assembly (OpenQASM) 3.0 specification boast typing as well as interoperability and portability between quantum systems of different types. However, others such as the QUASAR instruction set architecture assume some backend details such as a classical co-processor to complement our Quantum Processing Unit (QPU) (Shammah et al., 2024). To what extent the classical surrounds of a quantum processing unit should be assumed or specified and at what part of the stack is an open question. This "piping" can leverage many classical programming paradigms including web technology. What we refer to here is distinct from cloud quantum computing which simply offers a web-accessible front-end to users of quantum computers. Instead, we are interested in parts of the programming model itself, such as pieces of the compile toolchain (compilers, transpilers, assemblers, noise profilers, schedulers), which are implemented using web technology. An example of this is Quantinuum's QEC decoder toolkit which uses a WebAssembly (WASM) virtual machine (WAVM) as a real-time classical compute environment for QEC decoding (*QEC Decoder Toolkit - H-Series*, n.d.). Other examples include our ports of Quantum Assembly (QASM), Quantum Macro Assembler (QMASM) and Blackbrid to TypeScript (Edwards, 2023).

An important part of standardization is verification. By our typed implementation of an OpenQASM 3.0 parser, we implement a system that infers types from QASM syntax. This opens the door to the type based formal verification of QASM code. A body of work exists regarding the verification of quantum software, which is summarized in (Fortunato et al., 2024).

The primary future direction that we see is the development of verification tools such as static analysis tools based on QASM-TS in the vein of QChecker (Zhao et al., 2023). This could be complemented by a formal type theory of OpenQASM 3.0.

Virtually every quantum computing company has provided access through a hybrid cloud. This demands that parts of the stack be implemented in web technology, and we argue that it is optimal in a sense to use technology that is designed for this environment when we find ourselves working in a hybrid quantum / classical cloud. We suggest that a closer marriage of open source efforts to the inherently web based stack supporting existing quantum computing offerings is desirable.

We note that Osaka University's open source quantum computer operating system project "Oqtopus" already depends on and makes use of Qasm-ts (Aso et al., 2024) and thank the Oqtopus team for their interest in our work.

Outcomes

Our comparative analysis focused on two promiment OpenQASM 3.0 parsers: Qiskit's Python ANLTR-based reference implementation and Qiskit's experimental Rust parser.

Performance Benchmarking

Benchmark Results			
Result	ANTLR Parser	Rust Parser	Qasm-ts
Success Rate	100% (11/11 files)	18.2% (2/11 files)	100% (11/11 files)
Average Time	8.53 ms	0.59 ms	0.90 ms
Min Time	1.93 ms	0.55 ms	0.36 ms
Max Time	30.54 ms	0.62 ms	3.68 ms

The benchmarking reveals that when just taking into account the AST generation, the Rust implementation generally offers superior raw performance, but suffers from only currently supporting a subset of the full OpenQASM 3.0 specification. The QASM-TS parser provides competitive performance for web deployment scenarios, while the ANTLR parser offers a balance of features and performance suitable for development and testing.

Acknowledgements

We would like to thank Dr. Shohini Ghose for support and helpful discussions regarding the previous version of this software package, QASM-TS 1.0, which has had use in the community by hundreds (counted by npm downloads).

References

Aso, N., Feluś, K., Gaj, A., Gokita, S., Góralczyk, S., Kakuko, N., Masumoto, N., Miyaji, K., Miyanaga, T., Mori, T., Noda, K., Tsukano, S., Ymaguchi, M., & Żybort, D. (2024). *OQTOPUS cloud*. https://github.com/oqtopus-team.github.io/oqtopus-cloud

Cross, A., Javadi-Abhari, A., Alexander, T., De Beaudrap, N., Bishop, L. S., Heidel, S., Ryan, C. A., Sivarajah, P., Smolin, J., Gambetta, J. M., & Johnson, B. R. (2022). OpenQASM 3: A broader and deeper quantum assembly language. *ACM Transactions on Quantum Computing*, *3*(3), 1–50. https://doi.org/10.1145/3505636

Di Matteo, O., Nunez-Corrales, S., Stechly, M., Reinhardt, S. P., & Mattson, T. (2024). An Abstraction Hierarchy Toward Productive Quantum Programming. In *arXiv.org*. https://arxiv.org/abs/2405.13918v1

Edwards, M. (2023). *Three Quantum Programming Language Parser Implementations for the Web*. arXiv. https://doi.org/10.48550/arXiv.2310.10802

Fortunato, D., Jiménez-Navajas, L., Campos, J., & Abreu, R. (2024). Verification and Validation of Quantum Software. In I. Exman, R. Pérez-Castillo, M. Piattini, & M. Felderer (Eds.), *Quantum Software* (pp. 93–123). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-64136-7_5

OpenQASM 3.0 Grammar. (n.d.). Retrieved November 24, 2024, from https://openqasm.com/grammar/index.html

QEC Decoder Toolkit - H-Series. (n.d.). Retrieved November 20, 2024, from https://docs.quantinuum.com/h-series/trainings/getting_started/pytket_quantinuum/qec_decoder_toolkit/Quantinuum_hseries_qec_decoder_toolkit.html?_gl=1*srcw0e*_gcl_au*MTU2MzIwMzA1NC4xNzI5NTk1NzEw

Seidel, R., Bock, S., Zander, R., Petrič, M., Steinmann, N., Tcholtchev, N., & Hauswirth, M. (2024). *Qrisp: A framework for compilable high-level programming of gate-based quantum computers*. https://arxiv.org/abs/2406.14792

Shammah, N., Roy, A. S., Almudever, C. G., Bourdeauducq, S., Butko, A., Cancelo, G., Clark, S. M., Heinsoo, J., Henriet, L., Huang, G., Jurczak, C., Kotilahti, J., Landra, A., LaRose, R., Mari, A., Nowrouzi, K., Ockeloen-Korppi, C., Prawiroatmodjo, G., Siddiqi, I., & Zeng, W. J. (2024). Open Hardware Solutions in Quantum Technology. *APL Quantum*, *1*(1), 011501. https://doi.org/10.1063/5.0180987

Wille, R., Schmid, L., Stade, Y., Echavarria, J., Schulz, M., Schulz, L., & Burgholzer, L. (2024). QDMI – Quantum Device Management Interface: A Standardized Interface for Quantum Computing Platforms. *IEEE International Conference on Quantum Computing and Engineering (QCE)*.

Zhao, P., Wu, X., Li, Z., & Zhao, J. (2023). *QChecker: Detecting Bugs in Quantum Programs via Static Analysis*. arXiv. https://doi.org/10.48550/arXiv.2304.04387