Goose: an OCaml environment for quantum computing

Denis Carnier, Arthur Correnson, Christopher McNally, Youssef Moawad

Goose: an OCaml environment for quantum computing

Denis Carnier[1], Arthur Correnson[2],
Christopher McNally[3], and Youssef Moawad[4]

[1] imec-DistriNet, KU Leuven, Belgium
denis.carnier@kuleuven.be
[2] CISPA Helmholtz Center for Information Security, Germany
arthur.correnson@cispa.de
[3] Massachusetts Institute of Technology, United States
mcnallyc@mit.edu
[4] University of Glasgow, United Kingdom
y.moawad.1@research.gla.ac.uk

Abstract

In this presentation, we showcase Goose: an OCaml library to model, simulate and compile low-level quantum programs. Goose is designed as a playground with an emphasis on extensibility and accessibility to non-experts. The library is compatible with the OpenQASM standard, and targets a variety of backends with a minimalistic, circuit-based IR.

1 Introduction

Quantum computing [8] is an emerging model of computation that exploits non-classical effects like *superposition* and *entanglement* to achieve algorithmic speedups. A radical break from classical computing, the model presents new challenges for programming languages research.

In particular, researchers are interested in more expressive ways to design and develop quantum algorithms, and the ability to leverage existing tooling for classical programming languages in the construction of quantum programming languages (QPLs) [1, 4, 9, 3, 6, 2].

OCaml is a mature ecosystem, widely used for programming languages research. However, to the best of our knowledge, it has a scarcity of tools for quantum programming. We therefore present Goose[1]: an OCaml library to model, simulate and compile quantum programs. The library is designed to act as a playground to experiment with quantum programming in the OCaml ecosystem.

2 Description and architecture

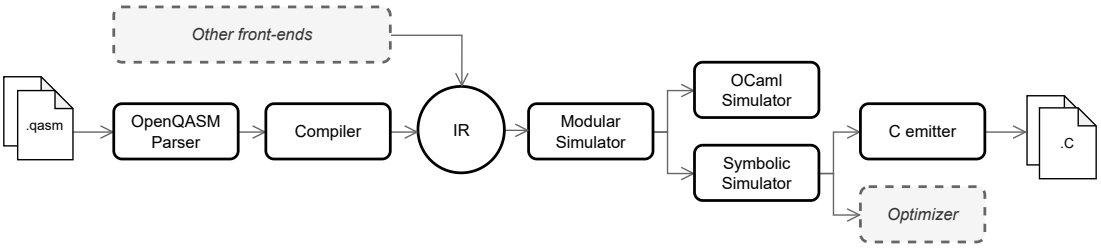

Figure 1: A high-level overview of Goose's modular pipeline.

[1] Goose is available at https://qgoose.github.io.

Figure 1 gives a high-level overview of Goose. The library is built around an intermediate representation (IR) for *quantum circuits*. This IR can be used as an entry point to, for instance, build simulators, compilers, and static analyzers for quantum programs. Goose includes a full state-based simulator to interpret the IR, designed as a functor parametrized by a linear algebra module. This design choice allows one to implement many tools simply by specializing the simulator. We demonstrate it by deriving both a symbolic simulator and an emitter to C. Further tools could be implemented, for example optimizers and compilers for other languages. To ensure compatibility with other existing tools related to quantum computing, we provide a frontend for OpenQASM 2.0, a standard low-level quantum circuit representation [5].

3 An example

In Figure 2, we provide an example of a GHZ [8] circuit entangling three qubits. Listing 3a shows the corresponding Goose IR representation. This is implemented with a list comprehension that specifies the CX gates, with an initial H gate. In Listing 3b, we display the C code that was generated based on the results of the symbolic simulator. `cadd`, `cmul`, and `csub` are C functions for complex number arithmetic. Finally, `s` and `o` are arrays for storing the input and output state, respectively.

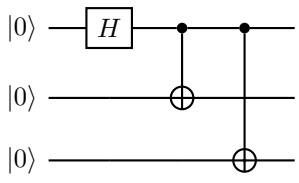

Figure 2: 3-qubit GHZ circuit.

```
let entanglement n = {
  qbits = n;
  gates = List.cons ({
    target = A 0;
    kind = H;
    controls = []
  }) (List.init (n-1)
  (fun i -> {
    target = A (i + 1);
    kind = X;
    controls = [A 0]
  }))
}
```

(a) Explicit construction of a GHZ circuit in the Goose IR.

```
cfloat *s = (cfloat*)malloc(N*sizeof(cfloat));
cfloat *o = (cfloat*)malloc(N*sizeof(cfloat));

for (int i = 0; i < N; i++)
    s[i] = (cfloat) {0.0, 0.0};

o[0] = cmul(SQRT1_2,cadd(s[0],s[1]));
o[1] = cmul(SQRT1_2,csub(s[6],s[7]));
o[2] = cmul(SQRT1_2,cadd(s[2],s[3]));
o[3] = cmul(SQRT1_2,csub(s[4],s[5]));
o[4] = cmul(SQRT1_2,cadd(s[4],s[5]));
o[5] = cmul(SQRT1_2,csub(s[2],s[3]));
o[6] = cmul(SQRT1_2,cadd(s[6],s[7]));
o[7] = cmul(SQRT1_2,csub(s[0],s[1]));
```

(b) Sample of generated C code to simulate this circuit for $n = 3$. Each line computes one amplitude in the output state: $\frac{1}{\sqrt{2}}(1, 0, 0, 0, 0, 0, 0, 1)^T$

4 Conclusion and future work

We presented Goose: an open source library for quantum computing in the OCaml ecosystem. Currently, the library's front-end supports a subset of the OpenQASM 2.0 circuit representation, a standard for interoperability between quantum computing tools. Additionally, Goose implements a generic simulator for quantum circuits. We specialized this simulator to both a symbolic simulator and a C emitter.

In the future, we want to connect Goose to other existing projects, for instance Twist [10], a QPL whose existing OCaml-based interpreter calls out to C++ quantum simulation libraries, or VOQC [7], a formally verified compiler for quantum circuits that can be extracted to OCaml.

Acknowledgements

We would like to thank the anonymous reviewers who have helped to improve this paper through their criticism and suggestions. Moreover, we especially thank Paulette Koronkevich for inspiring the name Goose.

C.M. was supported by the CQE-LPS Doc Bedard Fellowship. This research is co-funded by a PhD studentship from the College of Science and Engineering at the University of Glasgow, by the Research Fund KU Leuven, by the Flemish Research Programme Cybersecurity and by the European Union (ERC, UniversalContracts, 101040088). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] Qiskit: An open-source framework for quantum computing.

[2] Thorsten Altenkirch and Jonathan Grattage. A functional quantum programming language. *arXiv:quant-ph/0409065*, April 2005.

[3] 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 '20, pages 286–300, New York, NY, USA, June 2020. Association for Computing Machinery.

[4] Cirq Developers. Cirq, July 2018.

[5] Andrew W. Cross, Lev S. Bishop, John A. Smolin, and Jay M. Gambetta. Open Quantum Assembly Language. *arXiv:1707.03429 [quant-ph]*, July 2017.

[6] 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*, PLDI '13, pages 333–342, New York, NY, USA, June 2013. Association for Computing Machinery.

[7] Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. A verified optimizer for Quantum circuits. *Proceedings of the ACM on Programming Languages*, 5(POPL):37, January 2021.

[8] M.A. Nielsen and I.L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[9] Robert S. Smith, Michael J. Curtis, and William J. Zeng. A practical quantum instruction set architecture, 2016.

[10] Charles Yuan, Christopher McNally, and Michael Carbin. Twist: Sound reasoning for purity and entanglement in Quantum programs. *Proceedings of the ACM on Programming Languages*, 6(POPL):30:1–30:32, January 2022.