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
[2] Ecole Normale Supérieure de Rennes
[3] Massachusetts Institute of Technology
[4] University of Glasgow

Abstract

Quantum computing is an emerging model of computation that exploits non-classical effects like superposition and entanglement to achieve algorithmic speedups. A radical break from classical computation, the model presents new challenges for programming languages research. In this presentation, we showcase Goose: an OCaml library to model, simulate and compile low-level quantum programs. Goose is designed to support research into quantum programming languages through an emphasis on ease of use and extensibility. The library is compatible with the OpenQASM standard, and targets a variety of backends with a minimalistic circuit-based IR.

1 Introduction

- Hardware is scarce, Hardware doesn't work well
- Libraries/frameworks are all in weakly-typed Python (cf. Qiskit, Cirq)
- we should push the PL angle more
- Connecting IRs (never mind high-level languages) to the stuff you see in books: how? May not be obvious.
- challenges: accessibility, tooling needed, hardware errors and correctness matters

Quantum computing is an emerging model of computation that exploits non-classical physical effects like superposition and entanglement to achieve algorithmic speedups. Despite these performance promises and recent advances in the hardware implementation of quantum computers, widespread access remains limited. Furthermore, the current Noisy Intermediate Scale Quantum (NISQ) era systems are error-prone and can only maintain their coherence for a limited time. In addition, they typically do not have enough qubits to run meaningful algorithms and there may be further restrictions in terms of allowed gates and qubit connectivity.

As an alternative to physical quantum computers, simulation on classical computers allows researchers to study quantum algorithms without such restrictions and with predictable errors. Additionally, simulators enable researchers to artificially emulate errors to predict the behaviour of their algorithms on NISQera quantum computers.

For this reason, programming language researchers need accessible tools for compilation and classical simulation of quantum programs.

In this presentation, we showcase Goose: an extensible library to model, simulate and compile quantum programs.

that lays the foundation for further quantum programming languages research in the OCaml ecosystem. Goose is a low-level compilation framework that . . . [multiple input representations] . . . [multiple backends]

2 The Goose Library

What is Goose Goose is a friendly OCaml library for Quantum Computing. In its core, it features a simple intermediate representation to model quantum circuits. The library is built around this IR and provides a modular API to build and customize new tools such as simulators, compilers or static-analyzers for quantum programs.

Why ? Quantum Computing is not accessible to a wide audience. This can be explained by several factors. (1) Maths/Physics are scary, (2) Quantum Computers aren't accessible to non-experts, (3) Computer Science in itself is perceived as hard. Recent projects [] tried to overcome these limitations by developing more resources to make Quantum Computing more accessible to computer scientists. Goose follows the same track and try to provide a well-documented playground to start experimenting with the science of Quantum Computing and the activity of quantum programming. By choosing OCaml as an implementation language, we also hope to create interesting connections with the functional programming language community.

Goose's present and future Goose is completely open source and freely available at the address . Currently, Goose supports OpenQASM as a front-end, the *de facto* standard for interoperability between quantum computing tools. It also accepts a friendlier textual circuit descriptions. Goose also implements a customizable simulator for quantum circuits. To demonstrate the benefits of the modular design, we derive a symbolic simulator and a C compiler from the circuit simulator. These tools are distributed as a part of the library.

In the future, we could leverage Goose's modularity to connect goose with other existing projects such as VOQC, a formally verified compiler for quantum circuits that can be extracted to OCaml.

References