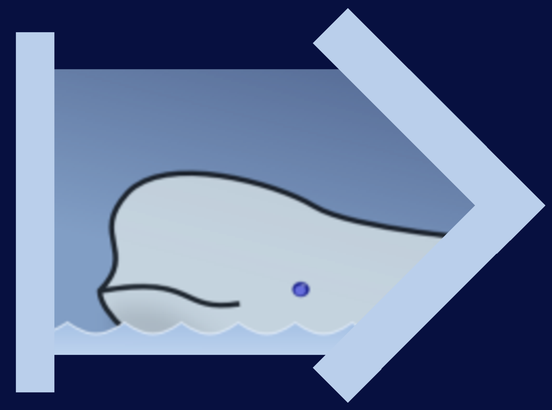
Structural Proto-Quipper: Mechanization of a Linear Quantum Programming Language in a Structural Setting



This work was funded by an NSERC USRA from UQAM and the FRQNT

Max Gross*, Ryan Kavanagh†, Brigitte Pientka*, Chuta Sano*

*McGill University

†UQAM

Problem and Motivation

- Goal: Develop a general method for mechanizing quantum programming languages that use linear logic, which manages resources like qubits.
- Key Idea: Solve the challenge of integrating linear logic into systems based on structural logic by "enforcing linearity without linearity."
- Proof of Concept: Mechanize Proto-Quipper, a quantum language for generating circuits, using Beluga, a formal reasoning tool.
- Prove two properties:
  a. Subject Reduction: Well-typed expressions retain their type after stepping.
  b. Progress: Well-typed expressions either result in a value or can step further.

Background and Related Work

- Mahmoud et al. mechanized Proto-Quipper by formalizing its semantics and proving type soundness.
- Used a linear extension of Hybrid in Coq to enforce linearity.
- This project avoids such extensions and enforces linearity within the Logical Framework itself.
- Builds on Karl Crary's technique for representing Girard's linear logic in Twelf.
- Crary defined a predicate to ensure variables are used linearly in typing judgments.
- Sano et al. applied this approach in Beluga for session typing.
- This project extends the idea to quantum programming languages as a proof-of-concept

Approach and Uniqueness

In mechanizing Proto-Quipper, two linearity predicates were introduced

```
lin : ({x:tm} oft F x A → oft F (b x) B) → type.
lin/q : ({x : qv} oft F (b x) B) → type.
```

- These predicates ensure variables, including quantum variables, are used linearly within the typing judgment.
- The predicate takes as input a function from Beluga variables and their types to output a typing judgement
- By modeling variables as Beluga variables, the predicates leverage higher-order abstract syntax, simplifying proofs.
- Achieved entirely within the Logical Framework, without extensions, unlike Mahmoud et al.'s approach.
- Similar to the method used by Sano et al. in Beluga for concurrency.

Results and Contribution

- The mechanization of Proto-Quipper using linearity predicates was successful, validating Crary's technique in a new context.
- Proofs for subject reduction and progress are still incomplete.
- These proofs exist in Beluga's computational layer, where the Curry-Howard isomorphism represents them as recursive functions.
- Further work on intermediate lemmas is required to finalize these theorems.
- Plans to explore the mechanization of Proto-Quipper-Dyn, a newer version of Proto-Quipper that incorporates dynamic lifting.

Terms of Proto-Quipper

$$
\begin{array}{rcl}
a, b, c & ::= & x \mid q \mid (t, C, a) \mid \texttt{True} \mid \texttt{False} \mid \langle a, b\rangle \mid * \mid ab \mid \lambda x.a \mid \\
& & rev \mid unbox \mid box^T \mid \texttt{if } a \texttt{ then } b \texttt{ else } c \mid \texttt{let } * = a \texttt{ in } b \mid \\
& & \texttt{let } \langle x, y\rangle = a \texttt{ in } b. \\
t, u & ::= & q \mid * \mid \langle t, u\rangle.
\end{array}
$$

Types of Proto-Quipper

$$
\begin{array}{rcl}
A, B & ::= & \textbf{qubit} \mid 1 \mid \textbf{bool} \mid A \otimes B \mid A \multimap B \mid !A \mid \mathrm{Circ}(T, U). \\
T, U & ::= & \textbf{qubit} \mid 1 \mid T \otimes U.
\end{array}
$$

Example Mechanization

$$
\frac{\Gamma_1, !\Delta; Q_1 \vdash a : !^n A \quad \Gamma_2, !\Delta; Q_2 \vdash b : !^n B}{\Gamma_1, \Gamma_2, !\Delta; Q_1, Q_2 \vdash \langle a, b\rangle : !^n (A \otimes B)} \ (\otimes_i)
$$

```
oft/⊗i : oft F a A
         → oft F b B
         → equibang AB A (A' ⊗ B') A'
         → equibang AB B (A' ⊗ B') B'
         → oft F (pair a b) AB.

lin/⊗i1 : {D :  {x : tm} oft F x _ → oft F (a x) A}
          lin D
          → lin (\x. \tx. oft/⊗i (D x tx) _ _ _ ).

lin/⊗i2 : {D :  {x : tm} oft F x _ → oft F (b x) B}
          lin D
          → lin (\x. \tx. oft/⊗i  _  (D x tx) _ _ ).
```

REFERENCES

[1] Karl Crary. 2010. Higher-order representation of substructural logics. SIGPLAN Not. 45, 9 (sep 2010), 131–142. https://doi.org/10.1145/1932681.1863565
[2] Peng Fu, Kohei Kishida, Neil J. Ross, and Peter Selinger. 2023. Proto-Quipper with Dynamic Lifting. Proc. ACM Program. Lang. 7, POPL, Article 11 (jan 2023), 26 pages. https://doi.org/10.1145/3571204
[3] Jean-Yves Girard. 1987. Linear logic. Theoretical Computer Science 50, 1 (1987), 1–101. https://doi.org/10.1016/0304-3975(87)90045-4
[4] Robert Harper. 2005. Mechanizing the meta-theory of programming languages. In Proceedings of the Tenth ACM SIGPLAN International Conference on Functional Programming (Tallinn, Estonia) (ICFP '05). Association for Computing Machinery, New York, NY, USA, 240. https://doi.org/10.1145/1086365.1086396
[5] Mohamed Yousri Mahmoud and Amy P. Felty. 2019. Formalization of Metatheory of the Quipper Quantum Programming Language in a Linear Logic. Journal of Automated Reasoning 63, 4 (2019), 967–1002. https://doi.org/10.1007/s10817-019-09527-x
[6] Brigitte Pientka and Jana Dunfield. 2010. Beluga: A Framework for Programming and Reasoning with Deductive Systems (System Description). In Automated Reasoning, Jürgen Giesl and Reiner Hähnle (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 15–21.
[7] Neil Julien Ross. 2015. Algebraic and Logical Methods in Quantum Computation. Ph. D. Dissertation. Dalhousie University.
[8] Chuta Sano, Ryan Kavanagh, and Brigitte Pientka. 2023. Mechanizing Session-Types using a Structural View: Enforcing Linearity without Linearity. Proc. ACM Program. Lang. 7, OOPSLA2, Article 235 (oct 2023), 28 pages. https://doi.org/10.1145/3622810