Static Resource Analysis of Hybrid Programs with Unbounded Loops

Abstract. While quantum hardware remains limited, hybrid quantum-classical algorithms with complex control structures, including unbounded loops, are emerging, posing new challenges for program analysis: the accurate estimation of resource consumption of a given program is needed. In this work, we answer this challenge with the first static analysis solution for reasoning about the resources — termination or cost — of hybrid quantum programs with unbounded loops. Towards that end, we introduce *hybrid path-sums*, a symbolic representation of possible executions of a quantum program. A generic strategy for determining termination and expected resource consumption via loop invariants is also proposed. Finally, a prototype of this solution is implemented in Haskell. This work is the first step toward the design of a complete static resource analysis tool for hybrid quantum programs with unbounded loops, essential for the development of real-world quantum computing. This extended abstract illustrates fully developed results without expanding on the technicalities.

Keywords: Quantum Computing · Hybrid Programs · Path-Sums · Static Analysis · Resource Analysis · Termination Analysis · Symbolic Execution

1 Introduction

1.1 Context and Motivations

Research in quantum computing has traditionally been focused on the low-level unitary circuit model [21,20,14] whereby a 'program' is a one-shot run of a finite and fixed sequence of quantum gates. Recently, a new emphasis is given to *hybrid* programs where the control flow can depend on measurement outcomes and where unbounded loops are allowed [26,11]. This is motivated by, among others, variational algorithms [17] and repeat-until-success algorithms used, for instance, in post-selection and error correction settings [9].

It should not be overlooked that the development of hybrid languages, which are highly complex by nature, comes with its own set of constraints on the verification of the correctness of such programs, e.g., their physicality or their termination. This is precisely where formal methods and verification come into play [10,28,22]. Due to the probabilistic nature of hybrid programs and because, unlike low-level quantum models, they usually contain general (unbounded) recursion, one of the fundamental aspects to be studied concerns their *resource-awareness*. This 'resource' can be anything from termination to costs in time, gate count, or more. Performing static analyses to verify resource properties of hybrid programs (instead of just circuits) is a relatively recent issue that has yet to be explored in depth.

1.2 Contributions

We tackle this question with a resource analysis framework for the inference of guarantees on the costs of hybrid quantum programs with unbounded recursion in the introduced language Hybricks. Hybricks is in the style of the Qbricks [8] environment extending it to the hybrid case with measurement, classical control, and general recursion (hence the name).

General recursion has been a long-standing challenge in the analysis of quantum programs already in terms of semantics [4,5,29], let alone static resource analysis. In fact, in the setting of general recursion, programs might not terminate; this implies that the complexity properties we can guarantee for hybrid quantum programs are fairly subtle properties which echo, to some extent, the properties of probabilistic programming, such as almost-sure termination [7] (i.e., termination with probability 1). These properties also go well beyond the scope of properties that can be addressed by quantum *circuits*, which terminate by construction. Our work also provides guarantees on probabilistic properties of the program including the expected number of gates used, expected runtime, almost-sure termination, etc. The analysis of these properties relies on a novel symbolic representation of the quantum-classical state, called *hybrid path-sums*, which extends the path-sums formalism [1] where a state is described as a symbolic sum of basis vectors to the hybrid setting. This representation allows us to maintain compactness in the analysis of otherwise exponentially large state spaces. Concretely, our contributions are as follows.

1. The introduction of Hybricks, a hybrid variant of the open source environment Qbricks with potentially unbounded `while` loops. It is equipped with a denotational semantics defined on Classical-Quantum states (*CQ states*), a common representation of hybrid states in terms of density operators and super-operators [27,12].
2. The extension of path-sums to *hybrid path-sums* or `hps` capable of representing CQ states, and limits thereof, compactly, exactly, and symbolically. Hybrid path-sums are interpretable as pure vectors in higher-dimensional Fock spaces or as CQ states.
3. A symbolic operational semantics for the language in terms of transformations of hybrid path-sums which is sound with respect to the denotational semantics (Theorem 2), allowing the extraction of properties of the program via symbolic execution rather than explicit computation on CQ states (Corollary 1).
4. A heuristic for analyzing programs with while loops (Theorem 1), which is broadly applicable and illustrated through the running example of the quantum coin-toss (Fig. 1).
5. A prototype implementation `IHPS` of a Hybricks parser and `hps`-based symbolic operational semantics engine, with future extensions underway for mechanized proofs of `hps` equivalences and invariant conservation. To the best of our knowledge, this is the first semi-automated tool for analyzing the resources consumed by hybrid quantum programs.

2 Illustration of the Approach

We chose to explain our contributions directly through a running example. First, we introduce an imperative language Hybricks for hybrid programming, which includes classical and quantum variables, quantum and classical operations, computational basis non-destructive measurements, classical control, and unbounded loops.

$$\begin{aligned} \mathtt{p} ::= {} & \textbf{skip} \mid \textbf{qubit}\ \mathtt{q} \mid \textbf{bit}\ \mathtt{c} \mid \textbf{int}\ \mathtt{x} \\ & \mid \mathtt{U}(\mathtt{q},\ldots,\mathtt{q}) \mid \mathtt{x} := \mathtt{i} \mid \mathtt{c} := \mathtt{b} \mid \mathtt{c} := \textbf{measure}\ \mathtt{q} \\ & \mid \mathtt{p}\ ;\ \mathtt{p} \mid \textbf{if}\ \mathtt{b}\ \textbf{then}\ \mathtt{p}\ \textbf{else}\ \mathtt{p}\ \textbf{end} \mid \textbf{while}\ \mathtt{b}\ \textbf{do}\ \mathtt{p}\ \textbf{done} \end{aligned}$$

Hybricks has a denotational semantics defined on CQ states, partial density operators ρ over the space of both quantum and classical variables which are dephased on the classical variables; i.e., $\langle \boldsymbol{m}_1 | \rho | \boldsymbol{m}_2 \rangle = 0$ for $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$ basis state differing on the classical parts. This semantics is relatively standard, explained, e.g., in Feng and Ying's quantum Hoare logic [12].

Among the most ubiquitous patterns in hybrid programs is the *repeat-until-success* pattern [6,19,15] where a program which probabilistically produces a (checkable) desired outcome is repeated until such outcome is obtained. For clarity, we illustrate our approach on a simple such pattern: the repeated coin-toss described in Fig. 1, but examples with richer quantum features and nested loops are also developed in Appendices A and B

In CoinToss, a qubit `q` is initialized to $|0\rangle$ and a classical bit `c` as well as an integer counter `x` are both initialized to 0 (line 1). Then, a *coin-toss* is performed consisting of a Hadamard gate which sets `q` in a uniform superposition of $|0\rangle$ and $|1\rangle$ (line 2) followed by a measurement of `q` (line 3) storing the result 0 or 1 in `c`, with each outcome having probability $\frac{1}{2}$. This process is repeated until the measurement results in a 1 (line 4), with the number of iterations performed (i.e., the number of fails before success) being counted in `x` (line 7). By the end of the program, `x` contains the count of iterations/tosses before success, a representative of runtime in this simple program. This program is almost-surely terminating [7] and the mean value of `x` is 1 (i.e. $1 + 1 = 2$ tosses). Consequently, its resource-aware properties can be studied even in the absence of (strict) termination.

```
1 qubit q; bit c; int x;
2 H(q);
3 c := measure q;
4 while (¬c) do
5     H(q);
6     c := measure q;
7     x := x + 1
8 done
```

Fig. 1: The CoinToss program

The analysis of the program relies largely on *hybrid path-sums*, our proposed novel symbolic representation of the quantum-classical state. In essence, a hybrid path-sum is a symbolic tuple $\sum_{\boldsymbol{a}} \langle p, n \cdot |m_1\rangle_{\mathtt{q}} [m_2]_{\mathtt{c}} \rangle$ of expressions p, n, m_1, and m_2 over tuples $\boldsymbol{a} = (a_1, \ldots, a_k)$ of boolean or integer *path variables*. This expression describes a sum of *paths* (complex-weighted basis states) of the form $n(\boldsymbol{v}) \cdot e^{2\pi i \cdot p(\boldsymbol{v})} |m_1(\boldsymbol{v})\rangle_{\mathtt{q}}$ where $\boldsymbol{v} = (v_1, \ldots, v_n)$ ranges over the instantiations $v_i \in \mathbb{N}$ or $v_i \in \mathbb{B}$ of integer or boolean path variables a_i respectively, and $t(\boldsymbol{v})$ is the evaluation of a term t with each a_i assigned to v_i.

$$|\psi\rangle = \sum_{\substack{1 \le i \le k \\ v_i \in \mathbb{N} \text{ or } v_i \in \mathbb{B}}} n(\boldsymbol{v}) e^{2\pi i \cdot p(\boldsymbol{v})} |m_1(\boldsymbol{v})\rangle_{\mathtt{q}}$$

In addition to the quantum states given by $|m_1\rangle_{\mathtt{q}}$, hybrid path-sums, as their name suggests, also handle classical data $[m_2]_{\mathtt{c}}$. This allows them to symbolically and compactly represent the branching structure of the execution, both in terms of quantum superpositions and classical probabilistic branching. For instance, a qubit `q` in state $|+\rangle$ is described using the path-sum $\sum_c \langle 0, (1/\sqrt{2}) \cdot |c\rangle_{\mathtt{q}} \rangle$ with the boolean variable c encoding the quantum branching in the superposition $|+\rangle$ of the basis states $|0\rangle$ and $|1\rangle$. Measuring qubit `q` (non-destructively) is then encoded as copying the symbolic expression c in $|c\rangle_{\mathtt{q}}$ into a classical bit `c` as $[c]_{\mathtt{c}}$ in the path-sum $\sum_c \langle 0, (1/\sqrt{2}) \cdot |c\rangle_{\mathtt{q}} [c]_{\mathtt{c}} \rangle$. The interpretation is then that each measurement outcome $\nu \in \{0, 1\}$ corresponds to a filtration of the sum $\sum_{c \in \{0,1\}, c=\nu} \frac{1}{\sqrt{2}} |c\rangle_{\mathtt{q}}$ into a vector whose squared norm is the probability of obtaining the outcome ν and which, when re-normalized, is the resulting quantum state if ν was obtained.

More formally, `hps` are given by a grammar of expressions involving booleans, integers, dyadics ($\frac{k}{2^n} \mid k, n \in \mathbb{Z}$), constructible reals (the closure of $\mathbb{Q}$ under $\sqrt{\cdot}$, $+$, $\cdot$, $/$). Here, we only give the high-level `hps` constructors:

$$h ::= \langle p, n \cdot m \rangle \mid h + h \mid h \otimes h \mid h \oplus h \mid \sum_a h \mid \bigotimes_a h \mid \lim_x h$$

Hybrid path-sums support additions, tensor products, and direct sums ($\oplus$) which adds the two `hps` as probabilistic branches only, without interference. Most importantly, symbolic sums and tensor products can be considered over formal boolean or integer variables a, and, to model unbounded loops, `hps` support symbolic limits over integer variables x. An `hps` h can be interpreted as a CQ state, denoted $\mathrm{cq}(h)$ inductively such that the constructors above correspond to the appropriate operations on CQ states. Notably, limits are interpreted using Hilbert norms in infinite-dimensional spaces, and require therefore some non-trivial functional analysis machinery that we developed, but omit here for clarity and brevity.

We perform symbolic execution of programs (the operational semantics) as transformations of `hps`. For instance, a quantum bit-flip X transforms m_1 into $m_1 \oplus 1$ by performing a XOR, while the Hadamard H introduces a new path variable c to sum over corresponding to the fork of the basis states into the superpositions $|+\rangle$ and $|-\rangle$. We encode this as a big-step operational semantics $(\mathtt{p}, h) \Downarrow h'$ reading `p` *run on a state* h *results in the state* h'.

$$\Big(\mathtt{X(q)}, \Big\langle p, n \cdot |m_1\rangle_{\mathtt{q}}[m_2]_{\mathtt{c}}\Big\rangle\Big) \Downarrow \Big\langle p, n \cdot |m_1 \oplus 1\rangle_{\mathtt{q}}[m_2]_{\mathtt{c}}\Big\rangle$$
$$\Big(\mathtt{H(q)}, \Big\langle p, n \cdot |m_1\rangle_{\mathtt{q}}[m_2]_{\mathtt{c}}\Big\rangle\Big) \Downarrow \sum_c \Big\langle p + \frac{m_1 c}{2}, \frac{n}{\sqrt{2}} \cdot |c\rangle_{\mathtt{q}}[m_2]_{\mathtt{c}}\Big\rangle$$

With such a symbolic description of all possible states of the computer at a given point, many formal verification tasks (including functional correctness, termination, and resource consumption) can be reformulated as equivalence checks between hybrid path-sums, checkable with an equational theory under development.

In particular, for the CoinToss program, we can find a functional specification as a loop invariant $h_{\text{CT-INV}}[x]$ with a free variable x describing the state of the system after x iterations. This symbolic representation encodes all possible branches of execution (i.e., halting by iteration x or still needing to continue).

$$h_{\text{CT-INV}}[x] \stackrel{\text{def}}{=} \sum_{y=0}^{x} \Big\langle 0, \frac{1}{\sqrt{2^{y+1}}} \cdot |1\rangle_{\mathtt{q}}[1]_{\mathtt{c}}[y]_{\mathtt{x}}\Big\rangle + \Big\langle 0, \frac{1}{\sqrt{2^{x+1}}} \cdot |0\rangle_{\mathtt{q}}[0]_{\mathtt{c}}[x]_{\mathtt{x}}\Big\rangle$$

Finally, the invariant is then used in a rule for `while` to obtain the limiting state $h_{\text{CT-}\infty}$.

$$\frac{(\textbf{if } \mathtt{b} \textbf{ then } \mathtt{p} \textbf{ else skip end}, h[x]) \Downarrow h[x+1/x]}{(\textbf{while } \mathtt{b} \textbf{ do } \mathtt{p} \textbf{ done}, h[0/x]) \Downarrow \lim_x (1 \oplus \mathtt{b}) * h} \textsc{While}$$

$$h_{\text{CT-}\infty} \stackrel{\text{def}}{=} \sum_x \Big\langle 0, \frac{1}{\sqrt{2^{x+1}}} \cdot |1\rangle_{\mathtt{q}}[1]_{\mathtt{c}}[x]_{\mathtt{x}}\Big\rangle$$

In practice, both the form and the correctness of the invariant can be obtained by a generic heuristic strategy based on the one-sided branching structure of loops.

Theorem 1 (Heuristic for loops). *If there exist* `hps` *terms* $h^{\mathtt{b}}$, $h^{\neg\mathtt{b}}$, *and* $h_{\mathtt{next}}$ *with the same signature* $\mathtt{s}$ *such that all the following hold:*

$$(\mathtt{p}, h^{\mathtt{b}}) \Downarrow h_{\mathtt{next}} \quad ; \quad \begin{matrix} h^{\mathtt{b}} \equiv \mathtt{b} * h^{\mathtt{b}} \\ h^{\neg\mathtt{b}} \equiv \neg\mathtt{b} * h^{\neg\mathtt{b}} \end{matrix} \quad ; \quad \begin{matrix} h^{\mathtt{b}}[x+1/x] \equiv \mathtt{b} * h_{\mathtt{next}} \\ h^{\neg\mathtt{b}}[x+1/x] \equiv \neg\mathtt{b} * h_{\mathtt{next}} \end{matrix}$$

then, for $\mathtt{x} \in \mathtt{A} \setminus \mathtt{s}$, *we have:*

$$\big(\textbf{int } \mathtt{x}; \textbf{while } \mathtt{b} \textbf{ do } \mathtt{p}; \mathtt{x} := \mathtt{x} + 1 \textbf{ done}, (h^{\mathtt{b}} \oplus h^{\neg\mathtt{b}})[0/x]\big) \Downarrow \sum_x h^{\neg\mathtt{b}} \otimes [x]_{\mathtt{x}}$$

The symbolic semantics illustrated here is sound with respect to the denotational semantics, allowing us to extract quantitative properties such as expectation values through symbolic execution rather than explicit computation on CQ states.

Theorem 2 (Soundness).

$$\forall(\mathtt{p}, h_1) \in \mathtt{Conf}, \forall h_2 \in \mathtt{HPS}, (\mathtt{p}, h_1) \Downarrow h_2 \implies [\![\mathtt{p}]\!]\,(\mathrm{cq}(h_1)) = \mathrm{cq}(h_2)$$

Corollary 1. *For all* $(\mathtt{p}, h_1) \in \mathtt{Conf}$, $h_2 \in \mathtt{HPS}$, *and observable* A *over* $\mathcal{H}(\mathfrak{s}(h_2))$,

$$(\mathtt{p}, h_1) \Downarrow h_2 \implies \mathbb{E}[A \mid [\![\mathtt{p}]\!]\,(\mathrm{cq}(h_1))] = \mathbb{E}[A \mid \mathrm{cq}(h_2)]$$

In this form, we are able to extract the entire probability distribution of the number of iterations before halting: $\mathbb{P}(\mathtt{x} = x) = \frac{1}{2^{x+1}}$, from which we can deduce both the almost-sure termination (AST) of the program, and the expected value of $\mathtt{x}$, which is $\mathbb{E}[\mathtt{x}] = \sum x \cdot \frac{1}{2^{x+1}} = 1$. This type of analysis can be generalized to other types of resources. For example, calculating the average number of gates of a certain type executed (e.g., expensive T [23,18] or multi-qubit gates [16]).

3 Implementation

Our solution to resource estimation with hybrid path-sums is accompanied by a prototype implementation `IHPS` of a HYBRICKS in Haskell consisting of a parser, a symbolic operational semantics engine, and a TEX pretty-printer for visualizing the resulting `hps`.

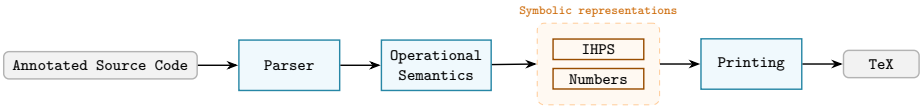

Fig. 2: Overview of the pipeline of `IHPS` and its main components.

Concretely, running `IHPS` on the source code of CoinToss produces the (TEX code of the) following `hps` (note: $\uparrow(\text{true}) = 1$ and $\uparrow(\text{false}) = 0$):

$$\lim_{x\in\mathbb{N}} \left(\sum_{y\in\mathbb{N}} \left(\left\langle 0, \frac{\uparrow(y \leq x) \cdot 1}{\sqrt{2^{y+1}}} \cdot |1\rangle_{\mathtt{q}}[1]_{\mathtt{c}}[y]_{\mathtt{x}} \right\rangle \right) \right)$$

4 Conclusion

In this abstract, we have presented, to the best of our knowledge, the first symbolic semantics of hybrid programs with general recursion and the first static resource analysis approach for such programs. This approach is scalable, maintaining effectiveness in the quantum-advantage regime where explicit computation on CQ states would be infeasible by definition. The approach benefits largely from a strong rewrite system for `hps` to simplify them, especially for proving invariants. For future work, we intend to enrich existing rewrite systems for standard path-sums [1,2,25,24] to the hybrid case, and to implement those rewrite systems in semi-automated/semi-interactive tools on top of our current prototype implementation `IHPS`.

References

1. Amy, M.: Towards large-scale functional verification of universal quantum circuits. In: Proceedings 15th International Conference on Quantum Physics and Logic, QPL 2018, Halifax, Canada, 3-7th June 2018. EPTCS, vol. 287, pp. 1–21 (2018). https://doi.org/10.4204/EPTCS.287.1 1.2, 4
2. Amy, M.: Complete equational theories for the sum-over-paths with unbalanced amplitudes. arXiv preprint arXiv:2306.16369 (2023) 4
3. Andrés-Martínez, P., Heunen, C.: Weakly measured while loops: peeking at quantum states. Quantum Science and Technology **7**(2), 025007 (2022) B
4. Assolini, N., Di Pierro, A.: A denotational semantics for quantum loops. arXiv preprint arXiv:2506.23320 (2025) 1.2
5. Barsse, K., Péchoux, R., Perdrix, S.: Quantum control and general recursion beyond the unitary case (7 2025), http://arxiv.org/abs/2507.10466v3 1.2
6. Bocharov, A., Roetteler, M., Svore, K.M.: Efficient synthesis of universal repeat-until-success quantum circuits. Phys. Rev. Lett. **114**, 080502 (Feb 2015). https://doi.org/10.1103/PhysRevLett.114.080502, https://link.aps.org/doi/10.1103/PhysRevLett.114.080502 2
7. Bournez, O., Garnier, F.: Proving positive almost-sure termination. In: RTA. pp. 323–337. Springer (2005) 1.2, 2
8. Cerezo, M., Arrasmith, A., Babbush, R., Benjamin, S.C., Endo, S., Fujii, K., McClean, J.R., Mitarai, K., Yuan, X., Cincio, L., et al.: Variational quantum algorithms. Nature Reviews Physics **3**(9), 625–644 (2021) 1.2
9. Chareton, C., Bardin, S., Bobot, F., Perrelle, V., Valiron, B.: An automated deductive verification framework for circuit-building quantum programs. In: Yoshida, N. (ed.) ESOP 2021. Lecture Notes in Computer Science, vol. 12648, pp. 148–177. Springer (2021). https://doi.org/10.1007/978-3-030-72019-3_6 1.1
10. Chareton, C., Lee, D., Valiron, B., Vilmart, R., Bardin, S., Xu, Z.: Formal methods for quantum algorithms. In: Akleylek, S., Dundua, B. (eds.) Handbook of Formal Analysis and Verification in Cryptography, pp. 319–422. CRC Press (2023). https://doi.org/10.1201/9781003090052-7 1.1
11. Dave, K., Lemonnier, L., Péchoux, R., Zamdzhiev, V.: Combining quantum and classical control: syntax, semantics and adequacy. In: Abdulla, P.A., Kesner, D. (eds.) FoSSaCS 2025. Lecture Notes in Computer Science, vol. 15691, pp. 155–175. Springer (2025). https://doi.org/10.1007/978-3-031-90897-2_8 1.1
12. Feng, Y., Ying, M.: Quantum Hoare logic with classical variables. ACM Transactions on Quantum Computing **2**(4), 1–43 (2021) 1, 2
13. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing. p. 212–219. STOC '96, Association for Computing Machinery, New York, NY, USA (1996). https://doi.org/10.1145/237814.237866 B
14. Knill, E.: Conventions for quantum pseudocode. arXiv preprint arXiv:2211.02559 (2022) 1.1
15. Lim, Y.L., Beige, A., Kwek, L.C.: Repeat-until-success linear optics distributed quantum computing. Phys. Rev. Lett. **95**, 030505 (Jul 2005). https://doi.org/10.1103/PhysRevLett.95.030505, https://link.aps.org/doi/10.1103/PhysRevLett.95.030505 2
16. Litinski, D.: A game of surface codes: Large-scale quantum computing with lattice surgery (Aug 2018). https://doi.org/10.22331/q-2019-03-05-128, http://arxiv.org/abs/1808.02892v3, quantum 3, 128 (2019) 2
17. McClean, J.R., Romero, J., Babbush, R., Aspuru-Guzik, A.: The theory of variational hybrid quantum-classical algorithms. New Journal of Physics **18**(2), 023023 (2016) 1.1
18. Meuli, G., Soeken, M., Campbell, E., Roetteler, M., de Micheli, G.: The role of multiplicative complexity in compiling low t-count oracle circuits. In: 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). pp. 1–8 (2019). https://doi.org/10.1109/ICCAD45719.2019.8942093 2

19. Paetznick, A., Svore, K.M.: Repeat-until-success: Non-deterministic decomposition of single-qubit unitaries. arXiv preprint arXiv:1311.1074 (2013) 2
20. Selinger, P.: Towards a quantum programming language. Math. Struct. Comput. Sci. **14**(4), 527–586 (2004). https://doi.org/10.1017/S0960129504004256 1.1
21. Selinger, P., Valiron, B.: A lambda calculus for quantum computation with classical control. Math. Struct. Comput. Sci. **16**(3), 527–552 (2006). https://doi.org/10.1017/S0960129506005238 1.1
22. Unruh, D.: Quantum relational Hoare logic. Proceedings of the ACM on Programming Languages **3**(POPL), 1–31 (2019) 1.1
23. Vandaele, V.: Lower T-count with faster algorithms. Quantum **9**, 1860 (Sep 2025). https://doi.org/10.22331/q-2025-09-16-1860, https://doi.org/10.22331/q-2025-09-16-1860 2
24. Vilmart, R.: Completeness of sum-over-paths for toffoli-hadamard and the dyadic fragments of quantum computation (5 2022), http://arxiv.org/abs/2205.02600v2 4
25. Vilmart, R.: Rewriting and completeness of sum-over-paths in dyadic fragments of quantum computing. Log. Methods Comput. Sci. **20**(1) (2024). https://doi.org/10.46298/LMCS-20(1:20)2024 4
26. Voichick, F., Li, L., Rand, R., Hicks, M.: Qunity: A unified language for quantum and classical computing. Proc. ACM Program. Lang. **7**(POPL), 921–951 (2023). https://doi.org/10.1145/3571225 1.1
27. Wilde, M.: Quantum information theory. Cambridge university press (2013) 1
28. Ying, M.: A practical quantum Hoare logic with classical variables, I. CoRR **abs/2412.09869** (2024). https://doi.org/10.48550/ARXIV.2412.09869 1.1
29. Ying, M., Feng, Y.: Quantum loop programs. Acta Informatica **47**, 221–250 (6 2010). https://doi.org/10.1007/s00236-010-0117-4, https://doi.org/10.1007/s00236-010-0117-4 1.2

A Nested while loops

With the symbolic representation introduced, it is possible to analyse the behavior of complex `while` loops symbolically without necessarily having to calculate difficult limits over the reals or complex numbers. In fact, as long as a sequence of complex numbers $(n[x]e^{2\pi ip[x]})_{x\in\mathbb{N}}$ can be expressed in closed form using the syntax of `Norm` and `Phase`, its limit can be expressed symbolically as well by injecting it into an `hps` as such $\lim_x \langle p, n\cdot\emptyset\rangle$, due to the fact that scalars are themselves vectors in $\mathcal{F}(\emptyset)$.

```
1  bit c1; bit c2;
2  int x1; int x2;
3  qubit q1; qubit q2;
4  while(¬c2) do
5      H(q2);
6      c1 := 0;
7      while(¬c1) do
8          H(q1);
9          c1 := measure q1;
10         R(q2, r);
11         x1 := x1 + 1;
12     done;
13     H(q2);
14     c2 := measure q2;
15     x2 := x2 + 1;
16     x1 := 0;
17 done
```

Fig. 3: The NestedWhile program.

Consider, for example, the nested loop in the NestedWhile program in Fig. 3. Let InnerLoop be the section of the program between lines 7 and 12, and OuterLoop be the section between lines 4 and 17. This program is designed specifically to test the capabilities of our symbolic representation in the context of nested, communicating `while` loops. The InnerLoop program is very similar to the CoinToss program we have been using as a running example which tosses qubit $\mathtt{q}_1$, except that it also applies a rotation (line 10) `R(qi, r)` to a different qubit $\mathtt{q}_2$ in a Hadamard basis state $|+\rangle$ or $|-\rangle$ at each iteration of the loop. This means that the final phase applied to $\mathtt{q}_2$ after exiting the InnerLoop is then dependent on the number of iterations k of the inner loop that were performed before the loop exited. This, in turn, influences the probability distribution in the measurement of the second qubit $\mathtt{q}_2$ in the OuterLoop program. In fact, each iteration of the OuterLoop roughly corresponds to applying $H\cdot R(r)^k\cdot H$ to $\mathtt{q}_2$ before measuring it, thereby making the probability of measuring 0 or 1 in $\mathtt{c}_2$ dependent on k.

By a similar analysis to CoinToss applied this time to InnerLoop, we find that, starting at line 7, from a state

$$h_{\mathtt{inner},0} \stackrel{\text{def}}{=} \sum_c \left\langle 0, 1\cdot|0\rangle_{\mathtt{q}_1}|c\rangle_{\mathtt{q}_2}[0]_{\mathtt{c}_1}[0]_{\mathtt{c}_2}[0]_{\mathtt{x}_1}[0]_{\mathtt{x}_2}\right\rangle,$$

we reach, by line 12, the state

$$h_{\mathtt{inner},\infty} \stackrel{\text{def}}{=} \sum_x\sum_c \left\langle \frac{xc}{2^r}, \frac{1}{\sqrt{2^{x+1}}}\cdot|1\rangle_{\mathtt{q}_1}|c\rangle_{\mathtt{q}_2}[1]_{\mathtt{c}_1}[0]_{\mathtt{c}_2}[x]_{\mathtt{x}_1}[0]_{\mathtt{x}_2}\right\rangle.$$

Then, by line 17, we have reached the state

$$h_{17} \stackrel{\text{def}}{=} \sum_{c'}\sum_x\sum_c \left\langle \frac{xc}{2^r}+\frac{cc'}{2}, \frac{1}{\sqrt{2^{x+1}}}\cdot|1\rangle_{\mathtt{q}_1}|c'\rangle_{\mathtt{q}_2}[1]_{\mathtt{c}_1}[c']_{\mathtt{c}_2}[0]_{\mathtt{x}_1}[1]_{\mathtt{x}_2}(k)_{\mathbb{Z}}\right\rangle.$$

with c' being introduced by an application of H after the InnerLoop and x being moved out of $\mathtt{x}_1$ and into the history $\{x\}_{\mathbb{Z}}$ by the resetting of $\mathtt{x}_1$ to 0 at line 16. Finally, the fact that the first iteration of the OuterLoop has been performed is marked by the incrementation of $\mathtt{x}_2$ to 1 at line 15 from 0 to 1.

At this point, we wish to apply the strategy (Theorem 1) for analysing the OuterLoop program by separating h_{17} into two parts $h^{\mathtt{b}}[1/x]$ and $h^{\neg\mathtt{b}}[1/x]$, according to the values of c' inhabiting $\mathtt{c}_2$ while extracting the closed form $h^{\mathtt{b}}[x]$ and $h^{\neg\mathtt{b}}[x]$; however, we are struck by an issue: each iteration of the OuterLoop appears to introduce a new path variable x, meaning that the size of the `hps` itself is dependent of the number of iterations of the OuterLoop. Keeping it at that, we would be unable to find a closed form. However, we can separate the probabilistic analysis of the inner loop from that of the outer loop as such:

$$h_{17} \equiv \sum_{c'} \left(\sum_{c} \sum_{k} \left\langle \frac{xc}{2^r} + \frac{cc'}{2}, \frac{1}{\sqrt{2^{x+1}}} \cdot (k)_{\mathbb{Z}} \right\rangle \otimes \left\langle 0, 1 \cdot |1\rangle_{\mathtt{q}_1} |c'\rangle_{\mathtt{q}_2} [1]_{\mathtt{c}_1} [c']_{\mathtt{c}_2} [0]_{\mathtt{x}_1} [1]_{\mathtt{x}_2} \right\rangle \right)$$

Then, we can define two `hps` α and β with null signatures (i.e. interpretable into scalars) as such:

$$\alpha \stackrel{\text{def}}{\equiv} \sum_{c} \sum_{x} \left\langle \frac{xc}{2^r}, \frac{1}{\sqrt{2^{x+1}}} \cdot \{x\}_{\mathbb{Z}} \right\rangle$$
$$\beta \stackrel{\text{def}}{\equiv} \sum_{c} \sum_{x} \left\langle \frac{xc}{2^r} + \frac{c}{2}, \frac{1}{\sqrt{2^{x+1}}} \cdot \{x\}_{\mathbb{Z}} \right\rangle$$

Indeed, since $\mathfrak{s}(\alpha) = \mathfrak{s}(\beta) = \emptyset$, both α and β are interpreted in CQ states as no more than the scalar probability of obtaining the measurement outcomes 0 and 1 in $\mathtt{c}_2$ respectively:

$$\mathrm{cq}(\alpha) = \sum_{x} \frac{1}{2^{x+1}} \left| \sum_{c} e^{2\pi i \frac{xc}{2^r}} \right|^2$$
$$\mathrm{cq}(\beta) = \sum_{x} \frac{1}{2^{x+1}} \left| \sum_{c} e^{2\pi i \left(\frac{xc}{2^r} + \frac{c}{2}\right)} \right|^2$$

Those probabilities are highly non-trivial, and yet, we can represent them symbolically and work however we wish with them, potentially rewriting them for easier computation, sending them to a computer algebra system, or even deciding to approximate them numerically at a later stage. In any case, this allows us to express the loop invariant of the OuterLoop program as parametrized by the number of iterations x' as such:

$$h^{\mathtt{b}}_{\mathtt{outer}}[x'] \stackrel{\text{def}}{\equiv} \alpha\beta^{x'} \otimes \left\langle 0, 1 \cdot |1\rangle_{\mathtt{q}_1} |1\rangle_{\mathtt{q}_2} [1]_{\mathtt{c}_1} [1]_{\mathtt{c}_2} [0]_{\mathtt{x}_1} [x'+1]_{\mathtt{x}_2} \right\rangle$$
$$h^{\neg\mathtt{b}}_{\mathtt{outer}}[x'] \stackrel{\text{def}}{\equiv} \beta^{x'+1} \otimes \left\langle 0, 1 \cdot |1\rangle_{\mathtt{q}_1} |0\rangle_{\mathtt{q}_2} [1]_{\mathtt{c}_1} [0]_{\mathtt{c}_2} [0]_{\mathtt{x}_1} [x'+1]_{\mathtt{x}_2} \right\rangle$$

where

$$\alpha^{x'} \stackrel{\text{def}}{\equiv} \bigotimes_{x''} (\langle 0, \uparrow(l \leq x') \cdot \emptyset \rangle \otimes \alpha + \langle 0, \uparrow(1 \oplus (x'' \leq x')) \cdot \emptyset \rangle)$$

By applying Theorem 1 to the this pair of `hps`, we obtain the following limiting `hps` for NestedWhile:

$$(\textsc{NestedWhile}, \langle 0, 1 \cdot \emptyset \rangle) \Downarrow \sum_{x'} \left(\alpha^{x'} \otimes \left\langle 0, 1 \cdot |1\rangle_{\mathtt{q}_1} |1\rangle_{\mathtt{q}_2} [1]_{\mathtt{c}_1} [1]_{\mathtt{c}_2} [0]_{\mathtt{x}_1} [x']_{\mathtt{x}_2} \right\rangle \right)$$

Which allows us to deduce the probability distribution of the number of iterations of the OuterLoop, despite the nesting and the communication of the two loops:

$$\begin{aligned}\mathbb{P}(x_2 = x') &= \|\alpha\|^2 \cdot \|\beta^{x'}\|^2 \\ &= \left(\sum_x \frac{1}{2^{x+1}} \left|\sum_c e^{2\pi i \frac{xc}{2^r}}\right|^2\right) \cdot \left(\sum_x \frac{1}{2^{x+1}} \left|\sum_c e^{2\pi i (\frac{xc}{2^r} + \frac{c}{2})}\right|^2\right)^{x'}\end{aligned}$$

Once again, we can also extract the probability of termination as

$$\mathbb{P}(\text{termination}) = \|\alpha\|^2 \cdot \sum_{x' \in \mathbb{N}} \|\beta\|^{2x'}$$

and, by the fact that we were able to represent it symbolically, we have the chance to apply intelligent computation on $\mathbb{P}(\text{termination})$ noting that it is a geometric series with ratio $\|\beta\|^2 < 1$; therefore, $\mathbb{P}(\text{termination}) = \frac{\|\alpha\|^2}{1-\|\beta\|^2} = 1$.

While this example is admittedly a bit ad-hoc, it is designed to illustrate what can be done by the `hps` symbolic representation, and how symbolic execution can be performed and composed in the context of nested `while` loops, all without necessarily having to decide on difficult questions of convergence and the computation of limits.

B Weak measurement

Due to the probabilistic nature of quantum computing, it is very often the case that the repeat-until-success schema appear in quantum algorithms. Consider, for instance, the Grover search algorithm [13], which, given a predicate Q on the set of n-bit strings such that $Q(\omega) = 1$ for a unique ω, finds ω by preparing a quantum state with high probability of being measured as $|\omega\rangle$ (expensive step) and measuring it. However, if the measurement returns a string $|\omega'\rangle \neq |\omega\rangle$, the only hope is to restart from scratch.

An alternative is to replace the measurement step with the so-called *weak κ-measurement* [3] which trades a decrease in the success probability from p to κp for the possibility not to lose the state entirely when measurement fails. In the case of failure, instead of losing all the amplitude of the $|\omega\rangle$ state, the weak measurement only drops it by a factor of $\sqrt{1-\kappa}$. The framework introduced is able to express κ-measurement and show that κ is a genuine measure of strength in the following sense: it takes $\frac{1}{\kappa}$ weak measurements on average to measure positively.

This is also the occasion to illustrate oracle-based analysis in the framework. Instead of implementing the state preparation algorithm and the weak measurement algorithm concretely, we can instead define them as the oracles `Prepare` which prepares a state h_0 on a signature $\mathfrak{s}_0$ to be measured and `WeakMeas-`κ which performs the weak measurement storing the result in a new bit `c`. We can use these oracles by giving them the simple symbolic semantics based on their specifications, as follows.

First, we assert that, for all $\mathfrak{s} \subseteq \mathtt{A}$ such that $\mathfrak{s} \cap (\mathfrak{s}_0 \cup \{\mathtt{c}\}) = \emptyset$, we have

$$\begin{gathered}\mathfrak{s} \vdash \texttt{Prepare} : \mathfrak{s} \sqcup \mathfrak{s}_0 \\ \mathfrak{s} \sqcup \mathfrak{s}_0 \sqcup \{\mathtt{c}\} \vdash \texttt{WeakMeas-}\kappa : \mathfrak{s} \sqcup \mathfrak{s}_0 \sqcup \{\mathtt{c}\}\end{gathered}$$

Then, we provide the following symbolic semantics for those oracles:

$$\begin{aligned}(\texttt{Prepare}, h) &\Downarrow h \otimes h_0 \\ (\texttt{WeakMeas-}\kappa, h) &\Downarrow \sqrt{\kappa} \cdot Q * h_\top + \sqrt{1-\kappa} \cdot Q * h_\bot + (\neg Q) * h_\bot\end{aligned}$$

Here, the semantics of `Prepare` is straightforward: it simply ignores what's in h, and prepares a new state h_0 along-side it. As for `WeakMeas-`κ, it results in two branches: either the measurement succeeds, in which case we project the state h onto the subspace satisfying the predicate Q (denoted $Q*-$), or it fails, in which case we keep the failing part of the state ($\neg Q$) and reduce the amplitude of the success part by $\sqrt{1-\kappa}$. The result of the measurement is stored in the classical bit `c` (specifically, $h_\top$ is the h marked with success ($\mathtt{c} := 1, h) \Downarrow h_\top$, and $h_\bot$ is the h marked with failure ($\mathtt{c} := 0, h) \Downarrow h_\bot$). Finally, the correct care is taken in reducing the amplitude of each branch. We have skimmed over some technical details here; check Remark 1 for more details.

With these oracles constructed, we can now write a program designed to express the (conditional) average number of weak measurements necessary to obtain a positive measurement as follows.

```
1 Prepare;
2 int x; bit c;
3 while (¬c) do
4     WeakMeas-κ;
5     x := x + 1;
6 done
```

Given an initial state h_0, and a classical predicate Q, we can split h_0 into orthogonal parts h_{good} and h_{bad} satisfying or not the predicate Q respectively. The invariant of weak measurement is then given by $h_i^b = h_{\text{bad}} \otimes [0]_{\mathtt{c}} \otimes [i]_{\mathtt{x}} + \sqrt{(1-\kappa)^i}\, h_{\text{good}} \otimes [0]_{\mathtt{c}} \otimes [i]_{\mathtt{x}}$ and $h_i^{\neg b} = \sqrt{(1-\kappa)^{i-1}\kappa}\, h_{\text{good}} \otimes [1]_{\mathtt{c}} \otimes [i]_{\mathtt{x}}$ with the limit state being $h_\infty = \sum_{x=1}^{\infty} \sqrt{(1-\kappa)^{x-1}\kappa}\, h_{\text{good}} \otimes [1]_{\mathtt{c}}[x]_{\mathtt{x}}$.

At that point, we conclude that the weak κ-measurement will succeed with probability $\sum_{x=1}^{\infty} (1-\kappa)^{x-1}\kappa |h_{\text{good}}|^2 = |h_{\text{good}}|^2$ and fail (never terminating) with probability $|h_{\text{bad}}|^2$. More importantly, conditional on success, the expected number of weak measurement necessary is $\frac{1}{\kappa}$. That is, κ does indeed express the strength of the measurement: $\kappa = 1$ is a strong measurement, and as κ decreases, the measurement is buffered over more and more iterations.

Remark 1. To be fully precise, Q is not itself an oracle here as boolean expressions, as is, do not support oracles. We can either assume that we know Q, or envision a language with oracles for boolean expressions (and perhaps all types). We don't find this crucial to this article in particular, but may be a feature of an implementation of the framework. Similarly, for simplicity, κ, as used in `Norm` expressions, must be expressed as a constructible number; there are no variables of the constructible type. This, in itself, is not a serious limitation as we can write as an abstract rational $\kappa = \frac{x_1}{x_2}$, considering that the rationals are dense in the reals. However, once again, this detail, which would have been needlessly cumbersome in the main text, is a possible feature of an implementation.