Runtime Analysis of Quantum Programs

A Formal Approach

FEDERICO OLMEDO, Department of Computer Science, University of Chile & IMFD, Chile
ALEJANDRO DÍAZ-CARO, Universidad Nacional de Quilmes & ICC (CONICET-UBA), Argentina

In this abstract we study the resource consumption of quantum programs. Specifically, we focus on the expected runtime of programs and, inspired by recent methods for probabilistic programs, we develop a calculus *à la* weakest precondition to formally and systematically derive the (exact) expected runtime of quantum programs. Notably, the calculus admits a notion of loop runtime invariant that can be readily used to derive upper bounds of their runtime. Finally, we show the applicability of our calculus analyzing the runtime of (a simplified version of) the BB84 quantum key distribution protocol.

1 INTRODUCTION

For the past decades, quantum programs have been intensively investigated. Research efforts have ranged from language design and formal semantics to termination analysis and verification techniques [13]. Nevertheless, the analysis of resource consumption has received very little attention.

The most relevant work we are aware of in this direction is the Quipper system [9], which compiles quantum programs described in a high-level language into low-level logical circuits, and estimates the size of the resulting circuits, in terms of their number of gates and qbits. The ScaffCC [3] compilation framework offers similar possibilities for programs in Scaffold, a high-level imperative language based on C. However, both these systems offer very restricted support for programs with a recursive control flow. Concretely, they can only encode loops that are (statically) bounded by a *finite* parameter known at compilation time, leaving prominent quantum algorithms such as the BB84 key distribution protocol [1] or the Simon's algorithm [11] out of their scope.

Another related line of research lies in the field of implicit computational complexity. Dal Lago et al. [2] have developed a lambda calculus that captures some polynomial time quantum complexity classes. As such, the lambda calculus can only establish *asymptotic* guarantees about the behaviour of programs of a particular (polytime) class.

We believe that all players in the quantum programming community can benefit from more effective tools to estimate the resource consumption of programs. Indeed, one of the fundamental appeals of quantum computing is the so-called *quantum speedup*, that is, the possibility that a quantum computer might efficiently solve problems that are intractable in the classical world. Having appropriate methods to formally assess this speedup is thus of utter importance.

The goal of this work is to provide a first step in this direction. We specifically focus on the *runtime analysis* of programs and, inspired by recent methods for probabilistic programs [4, 5, 7], we develop a calculus *à la* weakest precondition to formally and systematically derive the (exact) runtime of quantum programs (§ 3). Notably, the calculus admits a notion of loop *runtime invariant* that can be readily used to derive upper bounds of their runtime. In comparison to previous works, our calculus can handle programs with arbitrary loops and is flexible enough to accomodate different runtime models. We show the applicability of the calculus analyzing the runtime of an algorithm based on the BB84 quantum key distribution protocol [1] (§ 4).

We hope that this work serves as starting point for further developments on the resource consumption of quantum programs; we briefly discuss some promising directions in § 5.[1]

[1]Liu et al. [6] have independently developed similar ideas to ours to reason about the expected runtime of quantum programs. While there already exist some differences between the two approaches, we leave a thorough comparison as future work since [6] became available after the preparation of the current work.

2 PROGRAMMING MODEL

Quantum computations are governed by the four postules of the quantum mechanics:

1) **State space:** the state of a system is given by a density matrix[2] acting on a Hilbert space, referred to as the system *state space*;
2) **Evolution:** if in a time lapse a system transitions from state ρ to state ρ', then $\rho' = U\rho U^\dagger$ for some unitary operator U;
3) **Measurement:** a measurement over a system is modeled by a set $\mathcal{M} = \{M_m\}_{m\in M}$ of operators satisfying the normalization condition $\sum_{m\in M} {M_m}^\dagger M_m = I$. The measurement returns an outcome from set M and modifies the system state according this outcome: Outcome $m \in M$ occurs with probability $\mathsf{Pr}_\rho[\mathcal{M}{=}m] \triangleq \mathrm{tr}({M_m}^\dagger M_m \rho)$ and upon this outcome the system transitions from state ρ to state $\rho|_{\mathcal{M}=m} \triangleq (M_m \rho {M_m}^\dagger)/\mathsf{Pr}_\rho[\mathcal{M}{=}m]$;
4) **Composition:** the state space of a system composed of several subsystems is the tensor product of the state spaces of its components.

To describe quantum programs we use a core imperative language [8], coined qGCL. Programs are defined over a set of quantum variables (ranged over by q). Variable types are interpreted as Hilbert spaces. Here, we consider only variables of type Bool (interpreted as the 2-dimensional Hilbert space $\mathbb{C}^2$ with basis $\{|0\rangle, |1\rangle\}$) and Int (interpreted as the infinite-dimensional Hilbert space $\mathbb{C}^\omega$ with basis $\{|i\rangle\}_{i\in\mathbb{Z}}$). Programs in qGCL adhere to the following syntax:

$$c \;::=\; \mathsf{skip} \;\mid\; q \coloneqq |b\rangle \;\mid\; \overline{q} \coloneqq U\,\overline{q} \;\mid\; c_1; c_2 \;\mid\; \square\, \mathcal{M}[\overline{q}] \,=\, \overline{m \to c_m} \;\mid\; \mathsf{while}\,(\mathcal{M}[\overline{q}] = 1)\,\mathsf{do}\,c$$

Most language constructs have similar meaning to their classical counterpart. skip corresponds to a no-operation. $q \coloneqq |b\rangle$ initializes variable q with density operator $|b\rangle\langle b|$, where $|b\rangle$ lies in the basis of its type interpretation. $\overline{q} \coloneqq U\,\overline{q}$ updates the set of variables $\overline{q}$ according to the unitary operator U. $c_1; c_2$ represents the sequential composition of programs c_1 and c_2. $\square\, \mathcal{M}[\overline{q}] = \overline{m \to c_m}$ represents the quantum counterpart of the traditional case statement; it performs a measurement $\mathcal{M}$ on variables $\overline{q}$, and according to the observed outcome, execution continues with the corresponding branch. Finally, while $(\mathcal{M}[\overline{q}] = 1)$ do c represents a loop, guarded by a binary measurement $\mathcal{M}$ on $\overline{q}$. Outcome 0 represents the loop termination and outcome 1 a further loop iteration.

The Hilbert state space $\mathcal{H}_c$ of a program c is given by (the class of unit vectors in) the tensor product $\bigotimes_{q\in Var(c)} \mathcal{H}_q$ of the Hilbert spaces associated to each of the variables in c. To properly capture the semantics of c in case it is non-terminating[3], we need to generalize the notion of density matrix to that of *partial* density matrix[4] [10]. Then, we interpret c as a transformer $[\![c]\!] \colon \mathcal{P}_{\mathcal{H}_c} \to \mathcal{P}_{\mathcal{H}_c}$ of partial density matrices, where $\mathcal{P}_{\mathcal{H}_c}$ denotes the set of all partial density matrices over c state space $\mathcal{H}_c$. Transformer $[\![c]\!]$ is defined by induction on the structure of c as follows:

$$\begin{array}{lcl}
[\![\mathsf{skip}]\!](\rho) & = & \rho \\
[\![q \coloneqq |b\rangle]\!](\rho) & = & \rho[q \mapsto |b\rangle] \\
[\![\overline{q} \coloneqq U\,\overline{q}]\!](\rho) & = & U^{\uparrow q} \rho {U^{\uparrow q}}^\dagger \\
[\![c_1; c_2]\!](\rho) & = & [\![c_1]\!]([\![c_2]\!](\rho)) \\
[\![\square\, \mathcal{M}[\overline{q}] \,=\, \overline{m \to c_m}]\!](\rho) & = & \sum_m \mathsf{Pr}_\rho[\mathcal{M}{=}m] \cdot [\![c_m]\!](\rho|_{\mathcal{M}=m}) \\
[\![\mathsf{while}\,(\mathcal{M}[\overline{q}] = 1)\,\mathsf{do}\,c]\!](\rho) & = & lfp\big(\Phi^{\langle \mathcal{M}, c\rangle}\big)
\end{array}$$

[2] A *density matrix* is a square positive matrix with trace 1.
[3] By *non-terminating* programs we mean programs that terminate with probability less than 1.
[4] *Partial* density matrices generalize density matrices by allowing traces less or equal than 1.

In the second rule, the resulting state $\rho[q \mapsto |b\rangle]$ is defined as $\sum_{i\in\{0,1\}} (|b\rangle\langle i|)^{\uparrow q}\rho(|i\rangle\langle b|)^{\uparrow q}$ if $type(q) = \mathsf{Bool}$ and as $\sum_{i\in\mathbb{Z}} (|b\rangle\langle i|)^{\uparrow q}\rho(|i\rangle\langle b|)^{\uparrow q}$ if $type(q) = \mathsf{Int}$, where $A^{\uparrow q}$ is the canonical extension of the matrix A acting on q to the dimension of ρ. In the third rule, U denotes an unitary operator over $\bigotimes_{q\in\overline{q}} \mathcal{H}_q$. Finally, in the last rule $\Phi^{\langle \mathcal{M},c\rangle}(X) \triangleq \lambda\rho'.\ \mathsf{Pr}_{\rho'}[\mathcal{M}{=}0]\cdot \rho'|_{\mathcal{M}=0} + X\big(\mathsf{Pr}_{\rho'}[\mathcal{M}{=}1]\cdot [\![c]\!](\rho'|_{\mathcal{M}=1})\big)$.

3 PROGRAM RUNTIMES

Runtime model. Observe that the presence of measurements endows programs with a probabilistic behaviour: programs admit multiple executions, each occurring with a given probability. Here, we focus on the *expected* or average runtime of programs, which refers to the weighted sum of the runtime of their individual executions, where each execution is weighted according to its probability. In turn, to model the runtime of an individual program execution we asume that a skip statement consumes 1 unit of time and parametrize the runtime specification of the quantum operations by means of a function $\mathcal{T}[\cdot]$: a variable initialization with vector $|b\rangle$ takes $\mathcal{T}[|b\rangle]$ units of time, a state update induced by a unitary operator U takes $\mathcal{T}[U]$ units of time and a measurement $\mathcal{M}$ (together with the modification it induces on the program state) takes $\mathcal{T}[\mathcal{M}]$ units of time.

Runtime transformer ert. To formally capture the expected runtime of programs we use a continuation passing style, materialized by transformer ert. If c is a program with state space $\mathcal{H}_c$ and we let $\mathbb{T} = \mathcal{P}_{\mathcal{H}_c} \to \mathbb{R}^{\infty}_{\geq 0}$, then $\mathsf{ert}[c]\colon \mathbb{T} \to \mathbb{T}$, and acts as follows: Assume that $t\colon \mathbb{T}$ represents the runtime of the program following c, i.e. its continuation. Then $\mathsf{ert}[c](t)\colon \mathbb{T}$ represents the runtime of c plus its continuation. Here, both $\mathsf{ert}[c](t)$ and t have type $\mathbb{T}$ (rather than simply $\mathbb{R}^{\infty}_{\geq 0}$) because the runtime of programs (in particular, of c and its continuation) depends on the particular partial density matrix in $\mathcal{P}_{\mathcal{H}_c}$ from which their execution is started. Finally, observe that to recover the runtime of a plain program c, it suffices to set the runtime of its continuation to 0. Symbolically, $\mathsf{ert}[c](\lambda\rho'.\,0)(\rho)$ gives the runtime of c when executed from initial partial density matrix ρ.

The fundamental appeal of this continuation-based approach to model the runtime of programs is that transforer ert admits a simple and elegant definition by induction on the program structure:

$$
\begin{array}{lcl}
\mathsf{ert}[\mathsf{skip}](t) & = & \lambda\rho.\ 1 + t(\rho) \\
\mathsf{ert}[q \coloneqq |b\rangle](t) & = & \lambda\rho.\ \mathcal{T}[|b\rangle] + t\big(\rho[q \mapsto |b\rangle]\big) \\
\mathsf{ert}[\overline{q} \coloneqq U\ \overline{q}](t) & = & \lambda\rho.\ \mathcal{T}[U] + t\Big(U^{\uparrow q}\rho U^{\uparrow q\,\dagger}\Big) \\
\mathsf{ert}[c_1; c_2](t) & = & \mathsf{ert}[c_1](\mathsf{ert}[c_2](t)) \\
\mathsf{ert}[\square\ \mathcal{M}[\overline{q}] \ = \ \overline{m \to c_m}](t) & = & \lambda\rho.\ \mathcal{T}[\mathcal{M}] + \sum_m \mathsf{Pr}_\rho[\mathcal{M}{=}m]\cdot \mathsf{ert}[c_m](t)\big(\rho|_{\mathcal{M}=m}\big) \\
\mathsf{ert}[\mathsf{while}\ (\mathcal{M}[\overline{q}] = 1)\ \mathsf{do}\ c](t) & = & \mathit{lfp}\big(F_t^{\langle \mathcal{M}[\overline{q}],c\rangle}\big)
\end{array}
$$

Here, $F_t^{\langle \mathcal{M}[\overline{q}],c\rangle}(t') \triangleq \lambda\rho.\ \mathcal{T}[\mathcal{M}] + \mathsf{Pr}_\rho[\mathcal{M}{=}1]\cdot \mathsf{ert}[c](t')(\rho|_{\mathcal{M}=1}) + \mathsf{Pr}_\rho[\mathcal{M}{=}0]\cdot t(\rho|_{\mathcal{M}=0})$ and its least fixed point $\mathit{lfp}\big(F_t^{\langle \mathcal{M}[\overline{q}],c\rangle}\big)$ is taken w.r.t. the pointwise order over $\mathbb{T}$, i.e. $t_1 \preceq t_2$ if $t_1(\rho) \leq t_2(\rho)$ for every ρ.

Invariant-based reasoning. Reasoning about the runtime of loop-free programs is rather straightforward following the rules above. On the contrary, reasoning about the runtime of loopy programs requieres determining the least fixed point of transformers, which is not a simple task. Nevertheless, if we are interested in establishing upper bounds —rather than exact values— for the runtime of loopy programs, we can employ an invariant-based argument. Concretely,

$$
F_t^{\langle \mathcal{M},c\rangle}(I) \preceq I \quad \Longrightarrow \quad \mathsf{ert}[\mathsf{while}\ (\mathcal{M} = 1)\ \mathsf{do}\ c](t) \preceq I\,. \tag{1}
$$

The results follows from a direct application of Park's Theorem [12], exploiting ert continuity.

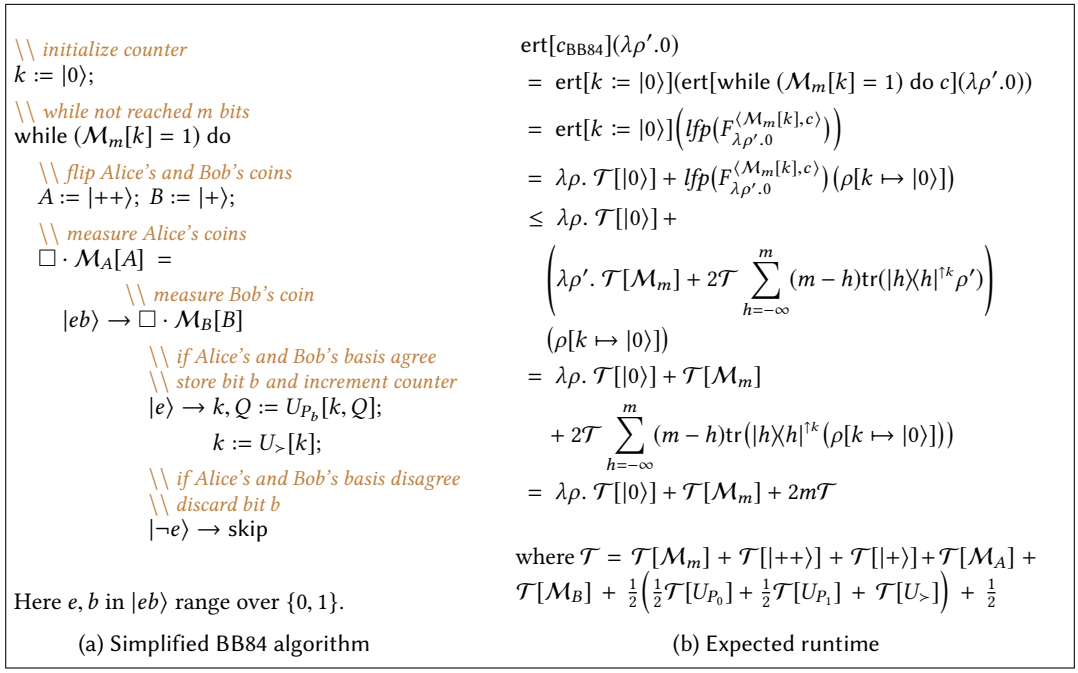

Fig. 1. Program c_{BB84} encoding (a simplified version of) the BB84 key distribution algorithm together with the derivation of its expected runtime.

4 CASE STUDY

We demonstrate the applicability of our approach by formally analyzing the runtime of a simplified version of the BB84 quantum key distribution algorithm [1]. BB84 is a protocol to securely create and distribute a shared (i.e. symmetric) key between two parties, say Alice and Bob. Assume the key consists of m bits. To begin with, Alice sends m encoded bits to Bob. To determine each of these bits, Alice flips two quantum coins; the first coin determines whether the encoded bit will be 0 or 1; the second coin determines whether she will encode it using basis $\{|0\rangle, |1\rangle\}$ or $\{|+\rangle, |-\rangle\}$. Then Bob continues by measuring each of the received (encoded) bits. For each of them, he flips a coin to determine the basis (either $\{|0\rangle, |1\rangle\}$ or $\{|+\rangle, |-\rangle\}$) he will use for the measurement. Finally, Alice and Bob publish the basis they employed to respectively encode and measure each bit. Bits whose respective basis coincide are kept as part of the resulting key; the remaining bits are discarded. The process continues until completing the m bits.

Here we abstract the basis exchange step. We assume that for each bit, Alice and Bob flip their coins, and immediately determine whether the basis used by Alice and Bob coincide, keeping or discarding the bit at hand accordingly. The qGCL program c_{BB84} representing this algorithm is depicted on Figure 1a. Variable k (over space $\mathbb{C}^\omega$) keeps track of the number of completed key bits; variable A and B (over space $\mathbb{C}^4$ and $\mathbb{C}^2$, respectively) represent the coin flips of Alice and Bob, respectively. Finally, variable Q stores (the successive bits of) the key. Set $\mathcal{M}_m = \{M_{\geq}, M_{<}\}$ measures counter k, distinguishing whether it reached m or not ($M_{\geq} = \sum_{i=m}^{\infty} |i\rangle\langle i|$ and $M_{<} = I - M_{\geq}$). Set $\mathcal{M}_A = \{|00\rangle\langle 00|, |01\rangle\langle 01|, |10\rangle\langle 10|, |11\rangle\langle 11|\}$ measures Alice's coin outcomes yielding the encoded bit and the employed basis. Finally, set $\mathcal{M}_B = \{|0\rangle\langle 0|, |1\rangle\langle 1|\}$ measures Bob's coin outcome, yielding his employed basis. Unitary operator U_{P_b} acts on variables k, Q setting the k-th

bit of Q to $|b\rangle$ (and leaving variable k untouched). Finally, unitary operator $U_{\succ}$ acts on variable k by updating its value from $|i\rangle$ to $|i+1\rangle$.

The runtime analysis of the program is depicted in Figure 1b. The crux of the analysis is proving that runtime $\lambda\rho'.\, \mathcal{T}[\mathcal{M}_m]+2\mathcal{T}\sum_{h=-\infty}^{m}(m-h)\cdot\mathrm{tr}(|h\rangle\langle h|^{\uparrow k}\rho')$ is a fixed point of transformer $F_{\lambda\rho'.0}^{\langle \mathcal{M}_m[k],c\rangle}$, where c stands for the body of c_{BB84} loop and $\mathcal{T}$ is as specified in Figure 1b; full derivation can be found in the Appendix. The overall analysis yields that program c_{BB84} has (at most) runtime

$$\mathcal{T}[|0\rangle] + \mathcal{T}[\mathcal{M}_m] + 2m\mathcal{T}$$

for any initial partial density matrix (from which it is executed), which matches the anticipated result: in average, the program terminates after $2m$ iterations; each iteration takes $\mathcal{T}$ units of time, there is a final measurement upon loop exit requiring $\mathcal{T}[\mathcal{M}_m]$ units of time, and the upfront initialization requires $\mathcal{T}[|0\rangle]$ units of time.

5 FUTURE WORK

The current work opens several research directions we plan to address shortly. Among them are i) defining an operational notion of expected runtime (e.g. based on quantum Markov chains) and proving a correspondence with our approach; ii) relating the ert transformer with the termination behavior of programs, showing that a finite expected runtime implies termination with probability 1; and iii) studying the effect of entangled states in the runtime of programs. Another long-term, more challenging goals comprise i) studying the problem of automation, more specifically loop invariant synthesis; ii) reasoning about the asymptotic runtime of programs; and iii) extending the language with further constructs such as general recursion or non-determinism.

REFERENCES

[1] C. H. Bennett and G. Brassard. Quantum cryptography: public key distribution and coin tossing. *Theoretical Compututer Science*, 560(12):7–11, 2014.

[2] U. Dal Lago, A. Masini, and M. Zorzi. Quantum implicit computational complexity. *Theoretical Computer Science*, 411(2):377–409, 2010.

[3] A. JavadiAbhari, S. Patil, D. Kudrow, J. Heckey, A. Lvov, F. T. Chong, and M. Martonosi. Scaffcc: A framework for compilation and analysis of quantum computing programs. In *Proceedings of the 11th ACM Conference on Computing Frontiers*, CF '14, pages 1:1–1:10. ACM, 2014.

[4] B. L. Kaminski, J.-P. Katoen, C. Matheja, and F. Olmedo. Weakest precondition reasoning for expected run–times of probabilistic programs. In P. Thiemann, editor, *Proceedings of the 25th European Symposium on Programming (ESOP 2016)*, volume 9632 of *Lecture Notes in Computer Science*, pages 364–389, Cham, 2016. Springer.

[5] B. L. Kaminski, J.-P. Katoen, C. Matheja, and F. Olmedo. Weakest precondition reasoning for expected runtimes of randomized algorithms. *Journal of the ACM*, 65(5):30:1–30:68, 2018.

[6] J. Liu, L. Zhou, and M. Ying. Expected runtime of quantum programs. arXiv:1911.12557, 2019.

[7] F. Olmedo, B. L. Kaminski, J.-P. Katoen, and C. Matheja. Reasoning about recursive probabilistic programs. In *Proccedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2016)*, pages 672–681, NY, USA, 2016. IEEE Computer Society.

[8] J. W. Sanders and P. Zuliani. Quantum programming. In R. B. N. Oliveira, editor, *Proceedings of the 5th International Conference on Mathematics of Program Construction (MPC 2000)*, volume 1837, pages 80–99, Berlin, Heidelberg, 2000. Springer.

[9] A. Scherer, B. Valiron, S.-C. Mau, S. Alexander, E. Van den Berg, and T. E. Chapuran. Concrete resource analysis of the quantum linear-system algorithm used to compute the electromagnetic scattering cross section of a 2d target. *Quantum Information Processing*, 16(3):16–60, 2017.

[10] P. Selinger. Towards a quantum programming language. *Mathematical Structures in Computer Science*, 14(4):527–586, 2004.

[11] D. R. Simon. On the power of quantum computation. *SIAM journal on computing*, 26(5):1474–1483, 1997.

[12] W. Wechler. *Universal Algebra for Computer Scientists*, volume 25 of *EATCS Monographs on Theoretical Computer Science*. Springer, Berlin, Heidelberg, 1992.

[13] M. Ying. *Foundations of Quantum Programming*. Morgan Kaufmann, Cambridge, USA, 2016.

APPENDIX: OMITTED CALCULATIONS FROM CASE STUDY IN § 4

In this appendix we include the omitted calculations in the runtime derivation sketched in Fig. 1b, that is, the application of the proof rule in Equation (1) to upper bound the runtime of the program loop. We apply the proof rule with runtime invariant

$$I \;\triangleq\; \lambda\rho.\, \mathcal{T}[\mathcal{M}_m] + 2(\mathcal{T}[\mathcal{M}_m] + \mathcal{T}^\star) \sum_{h=-\infty}^{m} (m-h)\cdot \mathrm{tr}(|h\rangle\langle h|^{\uparrow k}\rho)\,,$$

where $\mathcal{T}^\star \triangleq \mathcal{T}[|{+}{+}\rangle] + \mathcal{T}[|{+}\rangle] + \mathcal{T}[\mathcal{M}_A] + \mathcal{T}[\mathcal{M}_B] + \frac{1}{2}\big(\frac{1}{2}\mathcal{T}[U_{P_0}] + \frac{1}{2}\mathcal{T}[U_{P_1}] + \mathcal{T}[U_>]\big) + \frac{1}{2}$. The proof rule application requires showing that I is a pre-fixpoint of runtime transformer

$$F^{\langle \mathcal{M}_m[k],c\rangle}_{\lambda\rho'.\,0}(t') \;=\; \lambda\rho.\, \mathcal{T}[M_m] + \mathrm{Pr}_\rho[\mathcal{M}_m{=}1]\cdot \mathrm{ert}[c](t')\Big(\rho|_{\mathcal{M}_m=1}\Big)\,.$$

(Recall that $\mathcal{M}_m[k]$ stands for the loop guard measurement and c for the loop body.) We thus continue unfolding the above definition calculating $\mathrm{ert}[c](t')$. To this end we let $c_1 = A := |{+}{+}\rangle; B := |{+}\rangle$ be the first two initializations in c, $c_2 = \square\cdot \mathcal{M}_A[A] \;=\; \ldots$ be the remaining case statement in c and finally for each $e, b \in \{0,1\}$, we let $c_{2eb} = |eb\rangle \to \square\cdot\mathcal{M}_B[B]\ldots$ be the corresponding branch of c_2 case statement. Then we have,

$$\mathrm{ert}[c](t') \;=\; \mathrm{ert}[c_1](\mathrm{ert}[c_2](t')) \;=\; \mathrm{ert}[c_1]\left(\lambda\rho.\, \mathcal{T}[M_A] + \frac{1}{4}\sum_{e,b\in\{0,1\}} \underline{\mathrm{ert}[c_{2eb}](t')\Big(\rho|_{\mathcal{M}_A=eb}\Big)}\right) \tag{2}$$

where

$$\begin{aligned}
&\underline{\mathrm{ert}[c_{2eb}](t')}\\
&= \lambda\rho.\, \mathcal{T}[M_B] + \frac{1}{2}.\mathrm{ert}\big[[k,Q] := U_{P_b}[k,Q]; k := U_> \; k\big](t')\Big(\rho|_{\mathcal{M}_B=e}\Big) + \frac{1}{2}.\underline{\mathrm{ert}[\mathsf{skip}](t')\Big(\rho|_{\mathcal{M}_B=\neg e}\Big)}\\
&= \lambda\rho.\, \mathcal{T}[M_B] + \frac{1}{2}.\underline{\mathrm{ert}\big[[k,Q] := U_{P_b}[k,Q]; k := U_> \; k\big](t')}\Big(\rho|_{\mathcal{M}_B=e}\Big) + \frac{1}{2}.\Big(1 + t'\Big(\rho|_{\mathcal{M}_B=\neg e}\Big)\Big)\\
&= \lambda\rho.\, \mathcal{T}[M_B] + \frac{1}{2}.\mathrm{ert}\big[[k,Q] := U_{P_b}[k,Q]\big]\Big(\underline{\mathrm{ert}[k := U_> \; k](t')}\Big)\Big(\rho|_{\mathcal{M}_B=e}\Big) + \frac{1}{2}.\Big(1 + t'\Big(\rho|_{\mathcal{M}_B=\neg e}\Big)\Big)\\
&= \lambda\rho.\, \mathcal{T}[M_B] + \frac{1}{2}.\underline{\mathrm{ert}\big[[k,Q] := U_{P_b}[k,Q]\big]\Big(\lambda\rho.\, \mathcal{T}[U_>] + t'(U_>^{\uparrow k}\rho U_>^{\uparrow k\dagger})\Big)}\Big(\rho|_{\mathcal{M}_B=e}\Big)\\
&\qquad\qquad + \frac{1}{2}.\Big(1 + t'\Big(\rho|_{\mathcal{M}_B=\neg e}\Big)\Big)\\
&= \lambda\rho.\, \mathcal{T}[M_B] + \frac{1}{2}.\Big(\lambda\rho'.\, \mathcal{T}[U_{P_b}] + \underline{\Big(\lambda\rho.\mathcal{T}[U_>] + t'(U_>^{\uparrow k}\rho U_>^{\uparrow k\dagger})\Big)\Big(U_{P_b}^{\uparrow k,Q}\rho' U_{P_b}^{\uparrow k,Q\dagger}\Big)}\Big)\Big(\rho|_{\mathcal{M}_B=e}\Big)\\
&\qquad\qquad + \frac{1}{2}.\Big(1 + t'\Big(\rho|_{\mathcal{M}_B=\neg e}\Big)\Big)\\
&= \lambda\rho.\, \mathcal{T}[M_B] + \frac{1}{2}.\underline{\Big(\lambda\rho'.\, \mathcal{T}[U_{P_b}] + \mathcal{T}[U_>] + t'(U_>^{\uparrow k}U_{P_b}^{\uparrow k,Q}\rho' U_{P_b}^{\uparrow k,Q\dagger}U_>^{\uparrow k\dagger})\Big)\Big(\rho|_{\mathcal{M}_B=e}\Big)}\\
&\qquad\qquad + \frac{1}{2}.\Big(1 + t'\Big(\rho|_{\mathcal{M}_B=\neg e}\Big)\Big)\\
&= \lambda\rho.\, \mathcal{T}[M_B] + \frac{1}{2}.\Big(\mathcal{T}[U_{P_b}] + \mathcal{T}[U_>] + t'(U_>^{\uparrow k}U_{P_b}^{\uparrow k,Q}\Big(\rho|_{\mathcal{M}_B=e}\Big)U_{P_b}^{\uparrow k,Q\dagger}U_>^{\uparrow k\dagger})\Big) + \frac{1}{2}.\Big(1 + t'\Big(\rho|_{\mathcal{M}_B=\neg e}\Big)\Big)
\end{aligned}$$

Plugging the above expansion of $\mathsf{ert}[c_{2eb}]$ into (2) yields:

$$\mathsf{ert}[c_1]\left(\lambda\rho.\,\mathcal{T}[M_A] + \frac{1}{4}\sum_{e,b\in\{0,1\}} \underline{\mathsf{ert}[c_{2eb}](t')\Big(\rho|_{\mathcal{M}_A=eb}\Big)}\right)$$

$$= \mathsf{ert}[c_1]\left(\begin{array}{l}\lambda\rho.\,\mathcal{T}[M_A] + \frac{1}{4}\sum_{e,b\in\{0,1\}} \underline{(\mathcal{T}[M_B]} \\ \qquad \underline{+\frac{1}{2}.\Big(\mathcal{T}[U_{P_b}] + \mathcal{T}[U_{\succ}] + t'(U_{\succ}^{\uparrow k} U_{P_b}^{\uparrow k,Q}\Big(\Big(\rho|_{\mathcal{M}_A=eb}\Big)\Big|_{\mathcal{M}_B=e}\Big) U_{P_b}^{\uparrow k,Q^\dagger} U_{\succ}^{\uparrow k^\dagger})\Big)} \\ \qquad \underline{+\frac{1}{2}.\Big(1 + t'\Big(\Big(\rho|_{\mathcal{M}_A=eb}\Big)\Big|_{\mathcal{M}_B=\neg e}\Big)\Big)\Big)}\end{array}\right)$$

$$= \mathsf{ert}\Big[\underline{c_1}\Big]\left(\begin{array}{l}\lambda\rho.\,\mathcal{T}[M_A] + \mathcal{T}[M_B] + \frac{1}{2}.(\frac{1}{2}.\mathcal{T}[U_{P_0}] + \frac{1}{2}.\mathcal{T}[U_{P_1}] + \mathcal{T}[U_{\succ}]) + \frac{1}{2} \\ \quad + \frac{1}{8}\sum_{e,b\in\{0,1\}} \Big(t'\Big(U_{\succ}^{\uparrow k} U_{U_{P_b}}^{\uparrow k,Q}\Big(\Big(\rho|_{\mathcal{M}_A=eb}\Big)\Big|_{\mathcal{M}_B=e}\Big) U_{U_{P_b}}^{\uparrow k,Q^\dagger} U_{\succ}^{\uparrow k^\dagger}\Big) + t'\Big(\Big(\rho|_{\mathcal{M}_A=eb}\Big)\Big|_{\mathcal{M}_B=\neg e}\Big)\Big)\end{array}\right)$$

$$= \mathsf{ert}\Big[\underline{A := |{+}{+}\rangle}\Big]\left(\begin{array}{l}\lambda\rho.\,\mathcal{T}[|{+}\rangle] + \mathcal{T}[M_A] + \mathcal{T}[M_B] + \frac{1}{2}.(\frac{1}{2}.\mathcal{T}[U_{P_0}] + \frac{1}{2}.\mathcal{T}[U_{P_1}] + \mathcal{T}[U_{\succ}]) + \frac{1}{2} \\ \quad + \frac{1}{8}\sum_{e,b\in\{0,1\}} \Big(t'\Big(U_{\succ}^{\uparrow k} U_{U_{P_b}}^{\uparrow k,Q}\Big(\Big((\rho[B\mapsto |{+}\rangle\langle{+}|])|_{\mathcal{M}_A=eb}\Big)\Big|_{\mathcal{M}_B=e}\Big) U_{U_{P_b}}^{\uparrow k,Q^\dagger} U_{\succ}^{\uparrow k^\dagger}\Big) \\ \qquad\qquad + t'\Big(\Big((\rho[B\mapsto |{+}\rangle\langle{+}|])|_{\mathcal{M}_A=eb}\Big)\Big|_{\mathcal{M}_B=\neg e}\Big)\Big)\end{array}\right)$$

$$= \lambda\rho.\,\mathcal{T}^\star + \frac{1}{8}\sum_{e,b\in\{0,1\}} (t'\psi_{eb}(\rho) + t'\phi_{eb}(\rho))$$

where

- $\mathcal{T}^\star$ is as already defined on the previous page,
- $\psi_{eb}(\rho) = U_{\succ}^{\uparrow k} U_{U_{P_b}}^{\uparrow k,Q}\Big(\Big((\rho[AB\mapsto |{+}{+}{+}\rangle\langle{+}{+}{+}|])|_{\mathcal{M}_A=eb}\Big)\Big|_{\mathcal{M}_B=e}\Big) U_{U_{P_b}}^{\uparrow k,Q^\dagger} U_{\succ}^{\uparrow k^\dagger}$, and
- $\phi_{eb}(\rho) = \Big((\rho[AB\mapsto |{+}{+}{+}\rangle\langle{+}{+}{+}|])|_{\mathcal{M}_A=eb}\Big)\Big|_{\mathcal{M}_B=\neg e}$.

All in all, we have:

$$F^{\langle \mathcal{M}_m[k],c\rangle}_{\lambda\rho'.0}(t') \;=\; \lambda\rho.\,\mathcal{T}[M_m] + \mathsf{Pr}_\rho[\mathcal{M}_m{=}1].\left\{\mathcal{T}^\star + \frac{1}{8}\sum_{e,b\in\{0,1\}}\Big(t'\psi_{eb}\Big(\rho|_{\mathcal{M}_m=1}\Big) + t'\phi_{eb}\Big(\rho|_{\mathcal{M}_m=1}\Big)\Big)\right\}$$

Now we proceed to verify that I is a (pre-)fixpoint of $F^{\langle \mathcal{M}_m[k],c\rangle}_{\lambda\rho'.0}$:

$$\underline{F^{\langle \mathcal{M}_m[k],c\rangle}_{\lambda\rho'.0}(I)}$$

$$\boxed{\textit{definition of } F^{\langle \mathcal{M}_m[k],c\rangle}_{\lambda\rho'.0}}$$

$$= \lambda\rho.\, \mathcal{T}[M_m] + \mathsf{Pr}_\rho[\mathcal{M}_m{=}1]\Bigg\{\mathcal{T}^\star + \frac{1}{8}\sum_{e,b\in\{0,1\}}\Big(\underline{I\psi_{eb}\Big(\rho|_{\mathcal{M}_m=1}\Big)} + \underline{I\phi_{eb}\Big(\rho|_{\mathcal{M}_m=1}\Big)}\Big)\Bigg\}$$

definition of I and application

$$= \lambda\rho.\, \mathcal{T}[M_m] + \mathsf{Pr}_\rho[\mathcal{M}_m{=}1]\Bigg\{\mathcal{T}^\star + \frac{1}{8}\sum_{e,b\in\{0,1\}} \Big(\mathcal{T}[M_m] + 2(\mathcal{T}[M_m] + \mathcal{T}^\star)\sum_{h=-\infty}^{m}(m-h)\mathsf{tr}\Big(|h\rangle\langle h|^{\uparrow k}\psi_{eb}\Big(\rho|_{\mathcal{M}_m=1}\Big)\Big) + \mathcal{T}[M_m] + 2(\mathcal{T}[M_m] + \mathcal{T}^\star)\underline{\sum_{h=-\infty}^{m}(m-h)\mathsf{tr}(|h\rangle\langle h|^{\uparrow k}\phi_{eb}\Big(\rho|_{\mathcal{M}_m=1}\Big))}\Big)\Bigg\}$$

$$\boxed{\textstyle\sum_{h=-\infty}^{m}(m-h)\mathsf{tr}(|h\rangle\langle h|^{\uparrow k}\phi_{eb}\Big(\rho|_{\mathcal{M}_m=1}\Big)) = \sum_{h=-\infty}^{m-1}(m-h)\frac{\mathsf{tr}(|h\rangle\langle h|^{\uparrow k}\rho)}{\mathsf{Pr}_\rho[\mathcal{M}_m=1]}}$$

$$= \lambda\rho.\, \mathcal{T}[M_m] + \mathsf{Pr}_\rho[\mathcal{M}_m{=}1]\Bigg\{\mathcal{T}^\star + \frac{1}{8}\sum_{e,b\in\{0,1\}} \Big(\mathcal{T}[M_m] + 2(\mathcal{T}[M_m] + \mathcal{T}^\star)\underline{\sum_{h=-\infty}^{m}(m-h)\mathsf{tr}\Big(|h\rangle\langle h|^{\uparrow k}\psi_{eb}\Big(\rho|_{\mathcal{M}_m=1}\Big)\Big)} + \mathcal{T}[M_m] + 2(\mathcal{T}[M_m] + \mathcal{T}^\star)\sum_{h=-\infty}^{m-1}(m-h)\frac{\mathsf{tr}(|h\rangle\langle h|^{\uparrow k}\rho)}{\mathsf{Pr}_\rho[\mathcal{M}_m=1]}\Big)\Bigg\}$$

$$\boxed{\textstyle\sum_{h=-\infty}^{m}(m-h)\mathsf{tr}(|h\rangle\langle h|^{\uparrow k}\psi_{eb}\Big(\rho|_{\mathcal{M}_m=1}\Big)) = \sum_{h=-\infty}^{m}(m-h)\frac{\mathsf{tr}(|h-1\rangle\langle h-1|^{\uparrow k}\rho)}{\mathsf{Pr}_\rho[\mathcal{M}_m=1]}}$$

$$= \lambda\rho.\, \mathcal{T}[M_m] + \mathsf{Pr}_\rho[\mathcal{M}_m{=}1]\Bigg\{\mathcal{T}^\star + \underline{\frac{1}{8}\sum_{e,b\in\{0,1\}}} \Big(\mathcal{T}[M_m] + 2(\mathcal{T}[M_m] + \mathcal{T}^\star)\sum_{h=-\infty}^{m}(m-h)\frac{\mathsf{tr}(|h-1\rangle\langle h-1|^{\uparrow k}\rho)}{\mathsf{Pr}_\rho[\mathcal{M}_m=1]} + \mathcal{T}[M_m] + 2(\mathcal{T}[M_m] + \mathcal{T}^\star)\sum_{h=-\infty}^{m-1}(m-h)\frac{\mathsf{tr}(|h\rangle\langle h|^{\uparrow k}\rho)}{\mathsf{Pr}_\rho[\mathcal{M}_m=1]}\Big)\Bigg\}$$

Removing the summation

$$= \lambda\rho.\, \mathcal{T}[M_m] + \mathsf{Pr}_\rho[\mathcal{M}_m{=}1]\Bigg\{\mathcal{T}^\star + \frac{1}{2}$$
$$\Bigg(\underline{\mathcal{T}[M_m] + 2(\mathcal{T}[M_m] + \mathcal{T}^\star)\sum_{h=-\infty}^{m}(m-h)\frac{\mathrm{tr}(|h-1\rangle\langle h-1|^{\uparrow k}\rho)}{\mathsf{Pr}_\rho[\mathcal{M}_m{=}1]}}$$
$$\underline{+\mathcal{T}[M_m] + 2(\mathcal{T}[M_m] + \mathcal{T}^\star)\sum_{h=-\infty}^{m-1}(m-h)\frac{\mathrm{tr}(|h\rangle\langle h|^{\uparrow k}\rho)}{\mathsf{Pr}_\rho[\mathcal{M}_m{=}1]}}\Bigg)\Bigg\}$$

Adding up

$$= \lambda\rho.\, \mathcal{T}[M_m] + \underline{\mathsf{Pr}_\rho[\mathcal{M}_m{=}1]}\Big\{\underline{\mathcal{T}^\star + \mathcal{T}[M_m] + (\mathcal{T}[M_m] + \mathcal{T}^\star)}$$
$$\Bigg(\sum_{h=-\infty}^{m}(m-h)\frac{\mathrm{tr}(|h-1\rangle\langle h-1|^{\uparrow k}\rho)}{\mathsf{Pr}_\rho[\mathcal{M}_m{=}1]} + \sum_{h=-\infty}^{m-1}(m-h)\frac{\mathrm{tr}(|h\rangle\langle h|^{\uparrow k}\rho)}{\mathsf{Pr}_\rho[\mathcal{M}_m{=}1]}\Bigg)\Big\}$$

Factoring $(\mathcal{T}[M_m] + \mathcal{T}^\star)$ *out and distributing* $\mathsf{Pr}_\rho[\mathcal{M}_m{=}1]$

$$= \lambda\rho.\, \mathcal{T}[M_m] + (\mathcal{T}[M_m] + \mathcal{T}^\star)$$
$$\Big\{\underline{\mathsf{Pr}_\rho[\mathcal{M}_m{=}1]} + \sum_{h=-\infty}^{m}(m-h)\mathrm{tr}(|h-1\rangle\langle h-1|^{\uparrow k}\rho) + \sum_{h=-\infty}^{m-1}(m-h)\mathrm{tr}(|h\rangle\langle h|^{\uparrow k}\rho)\Big\}$$

$\mathsf{Pr}_\rho[\mathcal{M}_m{=}1] = \sum_{h=-\infty}^{m}\mathrm{tr}(|h\rangle\langle h|^{\uparrow k}\rho)$

$$= \lambda\rho.\, \mathcal{T}[M_m] + (\mathcal{T}[M_m] + \mathcal{T}^\star)$$
$$\Bigg\{\sum_{h=-\infty}^{m}\mathrm{tr}(|h\rangle\langle h|^{\uparrow k}\rho) + \sum_{\underline{h=-\infty}}^{m}(m-\underline{h})\mathrm{tr}(\underline{|h-1\rangle\langle h-1|}^{\uparrow k}\rho) + \sum_{h=-\infty}^{m-1}(m-h)\mathrm{tr}(|h\rangle\langle h|^{\uparrow k}\rho)\Bigg\}$$

Summing up

$$= \lambda\rho.\, \mathcal{T}[M_m] + 2(\mathcal{T}[M_m] + \mathcal{T}^\star)\sum_{h=-\infty}^{m}(m-h)\mathrm{tr}(|h\rangle\langle h|^{\uparrow k}\rho)$$

Definition of I

$$= I$$