Quantum Probabilistic Model Checking for Time-Bounded Properties

SEUNGMIN JEON, KAIST, South Korea
KYEONGMIN CHO*, Rebellions, South Korea
CHAN GU KANG, Korea University, South Korea
JANGGUN LEE, KAIST, South Korea
HAKJOO OH, Korea University, South Korea
JEEHOON KANG, KAIST, South Korea

Probabilistic model checking (PMC) is a verification technique for analyzing the properties of probabilistic systems. However, existing techniques face challenges in verifying large systems with high accuracy. PMC struggles with *state explosion*, where the number of states grows exponentially with the size of the system, making large system verification infeasible. While statistical model checking (SMC) avoids PMC's state explosion problem by using a simulation approach, it suffers from *runtime explosion*, requiring numerous samples for high accuracy.

To address these limitations in verifying large systems with high accuracy, we present *quantum probabilistic model checking* (QPMC), the first method leveraging quantum computing for PMC with respect to time-bounded properties. QPMC addresses state explosion by encoding PMC problems into quantum circuits that *superpose* states within qubits. Additionally, QPMC resolves runtime explosion through Quantum Amplitude Estimation, efficiently estimating the probabilities of specified properties. We prove that QPMC correctly solves PMC problems and achieves a quadratic speedup in time complexity compared to SMC.

CCS Concepts: • **Hardware → Quantum computation**; • **Software and its engineering → Model checking**; • **Mathematics of computing → Markov-chain Monte Carlo methods**.

Additional Key Words and Phrases: quantum computing, probabilistic model checking, bounded reachability

ACM Reference Format:
Seungmin Jeon, Kyeongmin Cho, Chan Gu Kang, Janggun Lee, Hakjoo Oh, and Jeehoon Kang. 2024. Quantum Probabilistic Model Checking for Time-Bounded Properties. *Proc. ACM Program. Lang.* 8, OOPSLA2, Article 291 (October 2024), 31 pages. https://doi.org/10.1145/3689731

1 Introduction

Systems with multiple interacting components such as distributed systems often exhibit probabilistic behaviors to maintain stability, even in the presence of the uncertainties inherent in their complex interactions [Herman 1990; Kwiatkowska et al. 2022, 2003]. For example, Herman's randomized self-stabilization protocol [Herman 1990] is designed to allow a distributed system to recover from

*Work done while at KAIST.

Authors' Contact Information: Seungmin Jeon, KAIST, Daejeon, South Korea, seungmin.jeon@kaist.ac.kr; Kyeongmin Cho, Rebellions, Seongnam, South Korea, kyeongmin.cho@rebellions.ai; Chan Gu Kang, Korea University, Seoul, South Korea, changukang@korea.ac.kr; Janggun Lee, KAIST, Daejeon, South Korea, janggun.lee@kaist.ac.kr; Hakjoo Oh, Korea University, Seoul, South Korea, hakjoo_oh@korea.ac.kr; Jeehoon Kang, KAIST, Daejeon, South Korea, jeehoon.kang@kaist.ac.kr.

ACM 2475-1421/2024/10-ART291
https://doi.org/10.1145/3689731

arbitrary initial states to a stable one even if some components temporarily behave incorrectly due to various factors such as network glitches, hardware malfunctions, or temporary software issues.

To automatically analyze the safety and performance of such probabilistic systems, *probabilistic model checking* (PMC) has been employed [Baier et al. 2018]. PMC answers various property queries, including time-bounded ones like, "the probability of reaching a stable state from a certain initial state *within 10 steps*," by modeling the system as a discrete-time Markov chain (DTMC) and analyzing the model. PMC has been actively utilized to analyze numerous real-world probabilistic systems [Filieri et al. 2011; Giannarakis et al. 2021; Holtzen et al. 2021; Jansen et al. 2020; Smolka et al. 2019; Su et al. 2016].

Challenges. However, classical approaches to PMC face scalability challenges due to the *state explosion problem* [Valmari 1996], which renders it computationally infeasible to obtain the precise probability for large probabilistic systems. For instance, in Herman's self-stabilizing algorithm [Herman 1990], the number of states grows exponentially with an increase in the number of processes.

While several approaches have been proposed to address the state explosion problem [Baier et al. 1997; Hensel et al. 2021; Holtzen et al. 2021; Katoen 2016; Younes and Simmons 2002], their scalability comes at the cost of sacrificing accuracy. A typical way to avoid state explosion is *statistical model checking* (SMC), which utilizes Monte Carlo simulations to estimate probabilities without exhaustive exploration of state space [Hérault et al. 2004; Younes and Simmons 2002]. But SMC suffers from the *runtime explosion problem* [Agha and Palmskog 2018; Budde et al. 2020; Hahn et al. 2019; Rubino et al. 2009], requiring a large number of samples to achieve high accuracy, thus making it computationally expensive, especially when analyzing rare events. For instance, achieving an error margin of 10^{-5} would require billions of samples, potentially requiring several weeks or more to explore [Budde et al. 2020].

Contributions. We address PMC's aforementioned scalability-accuracy tradeoff by utilizing quantum computers. Quantum computing has the potential to solve computational problems that are intractable on classical computers, especially with small input data sets [Hoefler et al. 2023]. This aspect is particularly relevant in PMC, where extensive computation is required to verify large probabilistic systems, which are often characterized by small model descriptions and property specifications.

Recognizing the potential of quantum computing to be well-suited for PMC, we introduce *quantum probabilistic model checking* (QPMC) for time-bounded properties, a method that translates PMC problems for these properties into quantum circuits. QPMC *superposes* states and transitions within qubits, and estimates the superposed probabilities in the resultant qubits for the given property queries by employing the Quantum Amplitude Estimation (QAE) algorithm [Brassard et al. 2002]. The QAE algorithm offers a quadratic speedup compared to SMC: it reaches an error bound of ϵ (< 1) with $1/\epsilon$ samples, whereas classical Monte Carlo methods (on which SMC is based) requires $1/\epsilon^2$ samples [Montanaro 2015]. Consequently, QAE has been successfully applied to a variety of problems, including financial risk measures [Egger et al. 2020; Stamatopoulos et al. 2020].

While doing so, we tackle the technical challenge of efficiently encoding large-scale PMC problems into quantum circuits. A potential strategy is to convert matrices into quantum-compatible unitary forms [Gilyén et al. 2019] and then create their corresponding quantum circuits. However, this strategy results in the expansion of DTMC models into exponentially large matrices, incurring the state explosion problem. To address this challenge, we make the following contributions:

- In §3, we encode PMC problems for time-bounded properties into quantum circuits. QPMC first preprocesses a DTMC model to desugar *inter-module synchronizations* and then converts it

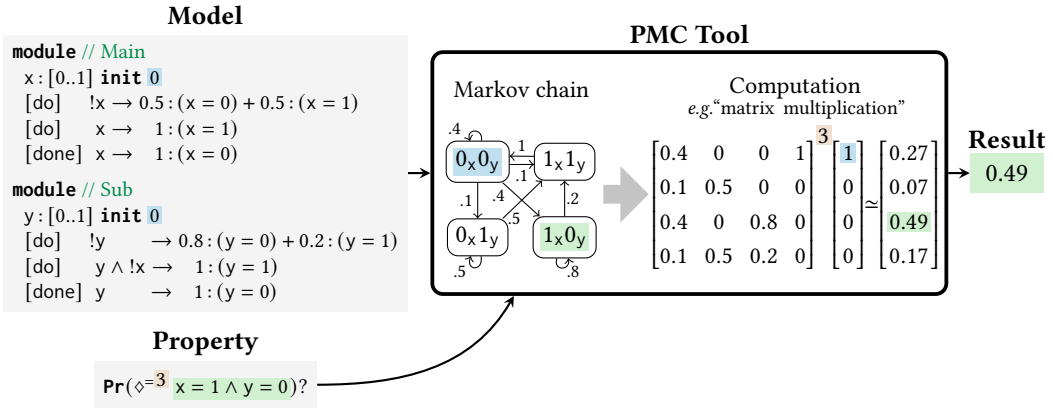


Fig. 1. Probabilistic model checking example.

into a circuit that corresponds to a single transition. It then constructs the final circuit tailored to the given property and estimates the result probabilities utilizing the QAE algorithm.

- In §4, we design QPMC's optimization technique, *batch recycling*, to reduce the number of qubits required. This technique involves repeatedly applying delayed *uncomputation* [Bennett 1973] to qubits that are no longer in use, making them available for later use.
- In §5, we prove that the encoded quantum circuit correctly solves the original PMC problem.
- In §6, we implement QPMC's translation algorithm using the Qiskit framework [Abraham et al. 2019] for quantum computing and observe that QPMC correctly verifies PMC problems via end-to-end classical simulation of small examples (up to 29 qubits).
- In §7, we analyze the performance of QPMC. QPMC achieves a quadratically lower time complexity compared to SMC for the same error bound and confidence level. Furthermore, QPMC's qubit optimization technique results in a 37% reduction in qubit requirements on average across benchmarks.

In §2, we review the technical background. In §8, we conclude with related and future work. All our implementation, experimental results, and proofs are available in the supplementary material [Jeon et al. 2024].

2 Background

2.1 Probabilistic Model Checking

Probabilistic model checking (PMC) is a formal technique designed to automate the analysis of probabilistic systems. It extends traditional model checking by quantifying probabilistic properties such as the "probability of unsafe events." PMC provides a robust framework for assessing safety and performance, and fault tolerance across diverse domains, ranging from computer networks to biological processes [Baier et al. 2018; Hensel et al. 2021; Katoen 2016].

Various PMC tools [Hartmanns and Hermanns 2014; Hensel et al. 2021; Kwiatkowska et al. 2011] follow a common workflow shown in Fig. 1, consisting of the following three phases:

(1) Model description: Users first describe probabilistic systems using model description languages such as Prism [Kwiatkowska et al. 2011] and Jani [Budde et al. 2017]. Fig. 1 shows

(Model)

$$
\begin{aligned}
\text{Model} \ni m &::= md_1 \,||\, \ldots \,||\, md_n \\
\text{Module} \ni md &::= \texttt{module}\ \vec{d}\ \vec{c} \\
\text{Decl} \ni d &::= v : [0..n_{\max}]\ \texttt{init}\ n_{\text{init}} \\
\text{Comm} \ni c &::= [a]\ g \to \lambda_1 : u_1 + \ldots + \lambda_n : u_n \\
\text{Guard} \ni g &::= e \\
\text{Update} \ni u &::= \overrightarrow{(v = e)} \\
\text{Expr} \ni e &::= n \mid v \mid !e \mid e_1\ op\ e_2 \\
Op \ni op &::= + \mid \leq \mid \wedge \mid \ldots \\
n \in \mathbb{N} \quad v \in \text{Var} \quad \lambda \in [0,1] \quad a &\in \text{Action} \cup \{\bot\}
\end{aligned}
$$

(Model)

$$
\begin{aligned}
[\![m]\!](j,i) &= \prod_{md \in m} \sum_{c \in md \wedge \text{Enabled}(c,i)} [\![c]\!](j,i) \\
[\![c]\!](j,i) &= \sum_{(\lambda:u) \in c \wedge \forall (v=e) \in u. [\![e]\!](i) = j[v]} \lambda \\
[\![u]\!](s) &= ([\![v_n = e_n]\!] \circ \ldots \circ [\![v_1 = e_1]\!])(s) \\
[\![v = e]\!](s) &= s[v \mapsto [\![e]\!](s)] \\
[\![e]\!](s) &\triangleq \text{(Evaluated value of } e \text{ in } s) \\
\text{Enabled}(c,s) &\triangleq [\![c.g]\!](s) \wedge \text{Sync}(c.a, s) \\
\text{Sync}(a,s) &\triangleq \forall md \in m.\ \exists c \in md.\ [\![c.g]\!](s) \wedge c.a = a
\end{aligned}
$$

(Configuration)

$$
\begin{aligned}
\text{Config} \ni \tau &::= (m, p) \\
\text{Prop} \ni p &::= \texttt{Pr}(\Diamond^{=t}\, \omega) \\
t \in \mathbb{N} \quad \omega &\in \text{Expr}
\end{aligned}
$$

(a) Syntax.

(Configuration)

$$
\begin{aligned}
[\![(m, \texttt{Pr}(\Diamond^{=t}\, \omega))]\!] &= \text{Res}(\omega, [\![m]\!]^t \pi_0(m)) \\
\text{Res}(\omega, \pi) &= \sum_{s \in S_\omega} \pi(s) \text{ where } S_\omega = \{s \in S \mid [\![\omega]\!](s)\} \\
\pi_0(m) &\triangleq \text{(Initial state of } m)
\end{aligned}
$$

(b) Semantics.

Fig. 2. PMC language.

a model comprising two *modules*: `Main` that waits for `Sub` to complete its tasks. Each module has a *state variable*, `x` and `y`, respectively, indicating whether it is ready (0) or done (1). The model describes state transition rules via *commands*. For instance, `Main`'s first command `[do] !x` $\to 0.5 : (\mathsf{x} = 0) + 0.5 : (\mathsf{x} = 1)$ implies that `x` will be updated to 0 or 1, each with a 50% probability, when this command is enabled. A command is enabled if its *guard condition* (`!x`) is met and its *synchronization action* (`do`) is enabled. An action is enabled if each module contains a command of the action whose guard is satisfied.[1] For example, in the state $\{\mathsf{x} \mapsto 1, \mathsf{y} \mapsto 0\}$, `Main`'s second command and `Sub`'s first command with the action `do` are enabled as their guard conditions are met. That is, `x` remains 1 and `y` transitions to 1 with 20% probability.

(2) Property specification: Users specify the properties for analysis using temporal logic formulas, such as PCTL [Hansson and Jonsson 1994]. Fig. 1 includes a time-bounded query about the probability of `Main` completing its job and `Sub` being ready after three transitions.

(3) Computation: PMC tools compute the probability of the specified property for the described model. Fig. 1 shows the model's Markov chain with four states and its corresponding *transition probability matrix*, describing the transition probabilities between states. The result is obtained by multiplying this matrix three times to the one-hot vector for the initial state $\{\mathsf{x} \mapsto 0, \mathsf{y} \mapsto 0\}$, resulting in a 49% probability of reaching the target state $\{\mathsf{x} \mapsto 1, \mathsf{y} \mapsto 0\}$ after three transitions.

[1] We do not consider cases where multiple commands are enabled in a module, which necessitates normalization for DTMC [Kwiatkowska et al. 2011]. In cases where no guard condition is met in a module, a self-loop command is implicitly included as in prior work [Holtzen et al. 2021].

Syntax. We use a core language of PMC simplified from Prism [Kwiatkowska et al. 2011]. Fig. 2a presents the syntax. A DTMC model comprises distinct modules, each with variable declarations and commands. Variables are specified with an initial value n_{init} and a maximum value n_{max}, restricting them to integer values ranging from 0 to n_{max}, inclusively. Each command comprises **(1)** action name (a) for synchronization; **(2)** guard condition (g) that dictates when the command can be invoked; and **(3)** probabilistic updates. Each probabilistic update, denoted by $\lambda_i : u_i$ for a given index i, includes **(1)** the probability λ_i to take this update, and **(2)** the update operation u_i of the assignment form $\overrightarrow{(v = e)}$ where v is a variable and e is an expression. An expression e is defined in a standard fashion. As a syntactic sugar, we use *formula* as temporary variables used in expressions.

A PMC problem configuration, denoted by τ, comprises a model m and a property p. A property p consists of the number t of transitions and a boolean expression ω that defines the set of target states. For example, the property $\mathbf{Pr}(\Diamond^{=3}\, \mathsf{x} = 1 \wedge \mathsf{y} = 0)$ in Fig. 1 represents the time-bounded query for the probability of reaching the state $\{\mathsf{x} \mapsto 1, \mathsf{y} \mapsto 0\}$ after three transitions.[2] Note that while the original PMC can handle a variety of properties from PCTL [Hansson and Jonsson 1994], we focus specifically on time-bounded reachability properties in this work [Katoen 2016].

Semantics. Fig. 2b presents the semantics as transition probabilities among states. A state $s \in S \triangleq \text{Var} \to \mathbb{N}$ is a variable map, and the semantics of a model m is a function $[\![m]\!] : S \times S \to [0, 1]$. Here, $[\![m]\!](j, i)$ indicates the probability of transitioning from state i to j. For simplicity, we also refer to $[\![m]\!]$ as a matrix and state s as a number that represents the global state of the model (*e.g.*, $s = 10_2$ for the state $\{\mathsf{x} \mapsto 1, \mathsf{y} \mapsto 0\}$). The transition probability $[\![m]\!](j, i)$ is the product of local transition probabilities across all enabled commands in all modules. A command c is enabled ($\mathsf{Enabled}(c, i)$) if its guard condition $c.g$ is met and its corresponding action $c.a$ is enabled. An action a is enabled ($\mathsf{Sync}(a, i)$) if each module has at least one command of the action whose guard is satisfied. The local transition probability $[\![c]\!](j, i)$ of a command from state i to j is the sum of the probabilities of all updates in c that change the variables from i to j. Also, a model should satisfy the following condition to ensure DTMC's basic properties:

$$\forall md \in m,\ s \in S.\ \exists! c.\ c \in md \wedge \mathsf{Enabled}(c, s)\ ,$$

meaning that for every step in any module, exactly one command is enabled.

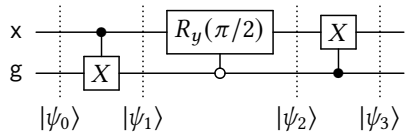
The semantics $[\![\tau]\!] \in [0, 1]$ of a PMC configuration $\tau = (m, \mathbf{Pr}(\Diamond^{=t}\, \omega))$ is the probability of ω being satisfied in m after t steps. The initial state of the model $\pi_0(m) : S \to [0, 1]$ is a one-hot vector that represents the initial state of m determined by the initial values of the variables; and $S_\omega \subseteq S$ denotes the set of target states satisfying the boolean expression ω.

For example, the semantics of the PMC configuration $\tau = (m, \mathbf{Pr}(\Diamond^{=3}\, \mathsf{x} = 1 \wedge \mathsf{y} = 0))$ presented in Fig. 1, where $S_\omega = \{10_2\}$, is defined as follows:

$$[\![\tau]\!] = \sum_{s \in S_\omega} [\![m]\!]^3 \pi_0(m) = \sum_{s \in S_\omega} \left(\begin{bmatrix} 0.4 & 0 & 0 & 1 \\ 0.1 & 0.5 & 0 & 0 \\ 0.4 & 0 & 0.8 & 0 \\ 0.1 & 0.5 & 0.2 & 0 \end{bmatrix}^3 \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right) \simeq \sum_{s \in \{10_2\}} \begin{bmatrix} 0.27 \\ 0.07 \\ 0.49 \\ 0.17 \end{bmatrix} = 0.49\ .$$

State explosion problem. Fig. 1 illustrates the direct matrix multiplication method to PMC. However, this approach is feasible only for tiny models due to the state explosion problem: the number of states and transitions grows exponentially relative to the model size. In a system with N processes,

[2] Bounded property $\mathbf{Pr}(\Diamond^{\leq t}\, \omega)$, which represents the probability of reaching the target state *within* t transitions, can be encoded with transient properties in the form of $\mathbf{Pr}(\Diamond^{=t}\, \omega)$ by tweaking the model [Katoen 2016].



$$C_{\text{main}} = Ct^1(X)(\mathsf{x},\mathsf{g});\ Ct^0(R_y(\pi/2))(\mathsf{g},\mathsf{x});\ Ct^1(X)(\mathsf{g},\mathsf{x});$$

(a) Diagram and syntax.

$$\begin{aligned}
|\psi_0\rangle &= \sqrt{0.3}\,|0\rangle_{\mathsf{x}}\,|0\rangle_{\mathsf{g}} &&+ \sqrt{0.7}\,|1\rangle_{\mathsf{x}}\,|0\rangle_{\mathsf{g}}\\
|\psi_1\rangle = [\![Ct^1(X)(\mathsf{x},\mathsf{g})]\!](|\psi_0\rangle) &= \sqrt{0.3}\,|0\rangle_{\mathsf{x}}\,|0\rangle_{\mathsf{g}} &&+ \sqrt{0.7}\,|1\rangle_{\mathsf{x}}\,|1\rangle_{\mathsf{g}}\\
|\psi_2\rangle = [\![Ct^0(R_y(\pi/2))(\mathsf{g},\mathsf{x})]\!](|\psi_1\rangle) &= \sqrt{0.15}\,|0\rangle_{\mathsf{x}}\,|0\rangle_{\mathsf{g}} + \sqrt{0.15}\,|1\rangle_{\mathsf{x}}\,|0\rangle_{\mathsf{g}} &&+ \sqrt{0.7}\,|1\rangle_{\mathsf{x}}\,|1\rangle_{\mathsf{g}}\\
|\psi_3\rangle = [\![Ct^1(X)(\mathsf{g},\mathsf{x})]\!](|\psi_2\rangle) &= \sqrt{0.15}\,|0\rangle_{\mathsf{x}}\,|0\rangle_{\mathsf{g}} + \sqrt{0.15}\,|1\rangle_{\mathsf{x}}\,|0\rangle_{\mathsf{g}} &&+ \sqrt{0.7}\,|0\rangle_{\mathsf{x}}\,|1\rangle_{\mathsf{g}}
\end{aligned}$$

(b) Semantics.

Fig. 3. 2-qubit quantum circuit for the `Main` module of the DTMC model in Fig. 1.

each in one of M states, the total possible states amount to M^N, leading to a transition matrix size of $M^N \times M^N$ [Herman 1990; Valmari 1996]. This exponential increase in complexity poses substantial computational challenges. While several techniques such as symbolic representation and abstraction have been proposed to mitigate the problem [Baier et al. 1997; Dehnert et al. 2012; Hahn et al. 2010; Katoen 2016; Kattenbelt et al. 2010], they still encounter exponential state growth (see §8 for details).

To avoid exhaustive state exploration, *statistical model checking* (SMC) employs Monte Carlo simulation to estimate model behavior [Bogdoll et al. 2011; Hérault et al. 2004; Lassaigne and Peyronnet 2002; Younes et al. 2006; Younes and Simmons 2002]. SMC determines whether a finite number of simulations of the system support or refute a given property. Consequently, this approach enables the handling of larger models while reducing computational demand and memory usage.

However, SMC is impractical for high-accuracy analysis. To achieve low error bound, SMC requires an inverse quadratically growing number of samples. For instance, achieving the error margin of 10^{-5} requires billions of samples and thus several weeks or more of analysis time [Budde et al. 2020]. This limitation is particularly crucial in safety-critical systems like transportation and nuclear plants, where even rare events are of significant consequence [Lagnoux 2006; Rubino et al. 2009].

2.2 Quantum Computing

Quantum computing leverages quantum mechanical phenomena to accelerate tasks that are computationally intensive. Fig. 3a depicts a *circuit* that represents a quantum algorithm. In circuits, each wire represents a *qubit* (quantum bit), the basic unit of information. Unlike classical bits, a qubit can exist in a *superposition* of two basis states, loosely meaning it inhabits both states probabilistically, thus enabling inherent parallel computation of probabilities. The blocks are quantum *gates* that manipulate the states of the qubits.

Qubits. A state of n qubits is represented as a unit vector in $\mathbb{C}^{2^n}$. This state can be represented as a superposition (linear combination) of *computational* (standard) basis vectors in $\mathbb{C}^{2^n}$. For example,

a 1-qubit state can be represented as $a_0 |0\rangle + a_1 |1\rangle$, where $a_0, a_1 \in \mathbb{C}$ satisfy $|a_0|^2 + |a_1|^2 = 1$; and $|0\rangle$ and $|1\rangle$ are the computational basis vectors $[1\ 0]^\dagger$ and $[0\ 1]^\dagger$, respectively, with $\dagger$ indicating the conjugate transpose. Generally, an n-qubit state $|\psi\rangle$ can be represented as $\sum_{i=0}^{2^n-1} a_i |i\rangle$, where each $|i\rangle$ is one-hot vector in 2^n-dimensions with the i-th element being 1; and their coefficients satisfy $\sum_{i=0}^{2^n-1} |a_i|^2 = 1$. Here, a basis vector is a tensor product of those of smaller qubit systems, *e.g.*, $|10_2\rangle = |1_2\rangle \otimes |0_2\rangle$ where $|10_2\rangle$ is a basis vector in 2-qubit system and $|1_2\rangle$, $|0_2\rangle$ in 1-qubit system. For simplicity, we omit the subscript "$_2$" denoting a binary number and the tensor product symbol $\otimes$ when it is clear from the context.

Each quantum state poses a probability distribution over possible measurement outcomes. Measuring the state $\sum_{i=0}^{2^n-1} a_i |i\rangle$ collapses into a basis state, say $|i\rangle$, with a probability of $|a_i|^2$. For example, in Fig. 3b, the last state $|\psi_3\rangle$ of qubits x and g is given as $\sqrt{.15}\,|0\rangle_{\mathsf{x}}\,|0\rangle_{\mathsf{g}} + \sqrt{.15}\,|1\rangle_{\mathsf{x}}\,|0\rangle_{\mathsf{g}} + \sqrt{.7}\,|0\rangle_{\mathsf{x}}\,|1\rangle_{\mathsf{g}}$, indicating a 15% probability of measuring $|00\rangle$, 15% for $|10\rangle$, and 70% for $|01\rangle$. Here, the subscript "$_{\mathsf{x}}$" or "$_{\mathsf{g}}$" indicates the qubit's name. The probability distribution for a *partial measurement* is the marginal probability distribution over possible outcomes. For instance, measuring only the qubit x from the state $|\psi_3\rangle$ implies a 85% probability of $|0\rangle_{\mathsf{x}}$ and 15% for $|1\rangle_{\mathsf{x}}$. We use $\mathsf{Pm}(|\psi\rangle, \vec{q})$ to denote the probability distribution of partial measurement of the qubits $\vec{q}$ from the state $|\psi\rangle$, *e.g.*, $\mathsf{Pm}(|\psi_3\rangle, \mathsf{x}) = [0.85, 0.15]$ (see Def. B.1 in the supplementary material [Jeon et al. 2024] for the full definition).

Gates. A gate represents a unitary transformation of qubit states. Formally, an n-qubit gate, say G, represents a unitary matrix $[\![G]\!]$ of size $2^n \times 2^n$ that maps a state $|\psi\rangle \in \mathbb{C}^{2^n}$ to a new state $[\![G]\!]\,|\psi\rangle$. For simplicity, we abuse the notation to denote by $[\![G]\!]$ the mapping as well. In circuit diagrams, a gate is represented as a block on the wires. For example, –$\boxed{X}$– denotes the X gate.

In this paper, we will use several well-known gates from the literature, with their meanings provided as follows:

$$[\![I]\!] := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad [\![X]\!] := \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad [\![R_y(\theta)]\!] := \begin{bmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{bmatrix}$$

The I gate is the 1-qubit identity gate that leaves the state unchanged; and the X gate maps $|i\rangle$ to $|\bar{i}\rangle$, where $\bar{i}$ is i's bit-complement. The $R_y(\theta)$ "y-rotation" gate maps $|i\rangle$ to $\cos(\theta/2)\,|i\rangle + (-1)^i \sin(\theta/2)\,|\bar{i}\rangle$, where θ is the rotation angle. We will use this gate to represent arbitrary probabilistic distributions (see §3.3 for details). It is worth noting that these mappings from the bases vectors fully characterizes the semantics of gates represented as unitary matrices.

The $Ct^{\vec{b}}(G)$ "*controlled-G*" gate applies G to the *target* qubits if the *control* qubits are in state $\vec{b}$. Formally, $[\![Ct^{\vec{b}}(G)]\!] := |i_1\rangle\,|i_2\rangle \mapsto |i_1\rangle\,[\![G]\!]^{i_1=\vec{b}}\,|i_2\rangle$, where $|i_1\rangle \in \{|0\rangle, |1\rangle\}^{n_1}$ and $|i_2\rangle \in \{|0\rangle, |1\rangle\}^{n_2}$ are basis vectors for control and target qubits, respectively; $\vec{b} \in \{0,1\}^{n_1}$ is the control condition for the n_1-qubits; and G is an n_2-qubit gate. One-bit controlled gate is visualized as a block with control qubits indicated by an empty circle (–○–) for $b = 0$, or a filled circle (–●–) for $b = 1$.

Circuits. Fig. 4 presents the syntax and semantics of quantum circuits. A circuit, say C, is a series of gate applications $G(\vec{q})$, where G is a gate; $\vec{q}$ is a sequence of qubits; and Q is the finite set of qubits considered. For example, C_{main} in Fig. 3a represents the 4-gate circuit. Also, a circuit should satisfy the following well-formedness condition:

$$\mathsf{wf}(C) \triangleq \forall G(\vec{q}) \in C.\ (|G| = |\vec{q}|) \wedge (\forall i, j.\ i \neq j \Rightarrow q_i \neq q_j)\ .$$

Here, the size $|G|$ of an n-qubit gate G is defined as n. In essence, each gate's arity should be observed and each gate should operate on distinct qubits.

$$C \ni C ::= \overrightarrow{G(\vec{q})}$$
$$\mathcal{G} \ni G ::= I \mid X \mid R_y(\theta) \mid Ct^{\vec{b}}(G) \mid \cdots$$
$$q \in Q \quad \theta \in [0, 2\pi) \quad b \in \{0, 1\}$$

(a) Syntax.

$$[\![[]]\!] \triangleq \text{(identitiy)} \qquad [\![G(\vec{q}); C]\!] \triangleq [\![C]\!] \circ [\![G(\vec{q})]\!]$$
$$[\![G(\vec{q})]\!] \triangleq \left(\bigotimes_{k \in \vec{q}} |i_k\rangle_k\right) \otimes \left(\bigotimes_{l \in (Q \setminus \vec{q})} |i_l\rangle_l\right)$$
$$\mapsto [\![G]\!]\left(\bigotimes_{k \in \vec{q}} |i_k\rangle_k\right) \otimes \left(\bigotimes_{l \in (Q \setminus \vec{q})} |i_l\rangle_l\right)$$

(b) Semantics.

Fig. 4. Quantum circuit language.

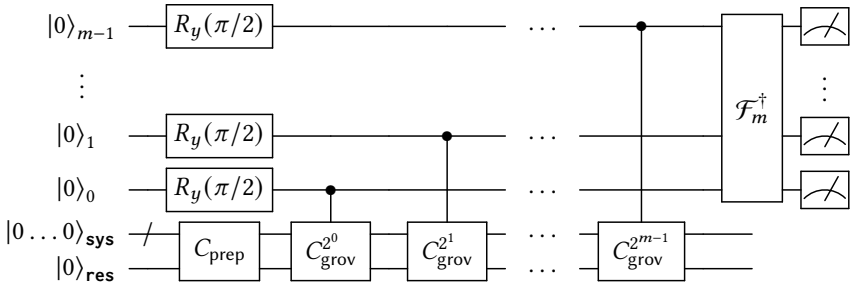


Fig. 5. The quantum circuit of QAE. The m qubits above represent the evaluation qubits for the QPE, while the qubits **sys** and **res** below represent the qubits for the state preparation circuit C_{prep}. C^i_{grov} denotes the application of the Grover operator i times, $\mathcal{F}_m^\dagger$ is the inverse quantum Fourier transform for m qubits, and the measurement is shown on the right side of $\mathcal{F}_m^\dagger$.

The semantics of C is a mapping $[\![C]\!] : \mathbb{C}^{2^n} \to \mathbb{C}^{2^n}$ for $n = |Q|$, defined as the composition of gate applications in C. In turn, $[\![G(\vec{q})]\!]$ applies gate G to the qubits in $\vec{q}$ while leaving the other qubits unaffected. For example, Fig. 3 presents a circuit C_{main} that *mirrors* the state transitions of the `Main` module from Fig. 1, without action synchronization. Qubits `x` and `g` represent the state of `Main` and a temporary value. Starting with the initial state $|\psi_0\rangle$ (30% probability of `x` = 0 and 70% of `x` = 1), $|\psi_1\rangle$ copies `x`'s basis to `g`. Then, $|\psi_2\rangle$ and $|\psi_3\rangle$ mirrors the first and third commands of `Main` for `x` = $|0\rangle$ and $|1\rangle$, respectively. Overall, the C_{main} transforms the state of two qubits `x` and `g` from $|\psi_0\rangle$ to $|\psi_3\rangle$ (85% probability of `x` = 0 and 15% of `x` = 1), thus parallelizing the state transition in the `Main` module except for the second command.

2.3 Quantum Amplitude Estimation

Quantum amplitude estimation (QAE) [Brassard et al. 2002] is a quantum algorithm that estimates the probability of measuring a specific state in a quantum state. Consider a circuit C_{prep} acting on a quantum state $|0\ldots0\rangle_{\mathbf{sys}} |0\rangle_{\mathbf{res}}$ for some multiple qubits **sys**, single qubit **res**. QAE efficiently estimates $a \in [0, 1]$, the probability of measuring $|1\rangle$ on the qubit **res** after applying C_{prep} to initial state $|0\ldots0\rangle_{\mathbf{sys}} |0\rangle_{\mathbf{res}}$, or more formally:[3]

$$a = \mathsf{Pm}([\![C_{\text{prep}}]\!](|0\ldots0\rangle_{\mathbf{sys}} |0\rangle_{\mathbf{res}}), \mathbf{res})(1) \ .$$

[3]This equation matches the form commonly found in QAE papers [Brassard et al. 2002; Stamatopoulos et al. 2020]: $[\![C_{\text{prep}}]\!](|0\ldots0\rangle_{\mathbf{sys}} |0\rangle_{\mathbf{res}}) = \sqrt{1-a}\,|\psi_0\rangle_{\mathbf{sys}} |0\rangle_{\mathbf{res}} + \sqrt{a}\,|\psi_1\rangle_{\mathbf{sys}} |1\rangle_{\mathbf{res}}$, where $|\psi_0\rangle$ and $|\psi_1\rangle$ are some normalized states.

QAE estimates a with an error of ϵ using $O(1/\epsilon)$ applications of C_{grov} (see below), where each application of C_{grov} corresponds to one quantum sample. QAE has a success probability of 81%, so by repeating the process a few times and taking the median result, the algorithm succeeds almost with certainty. This results in a quadratic speedup compared to classical Monte Carlo methods, which require $O(1/\epsilon^2)$ samples to achieve the same error ϵ [Montanaro 2015].

QAE circuit. Fig. 5 shows the complete QAE circuit. Instead of directly estimating a from C_{prep}, QAE estimates θ_a such that $\sin^2(\theta_a) = a$. The estimation of θ_a involves two primary components:

- Grover Operator (C_{grov}): The Grover operator C_{grov} is a unitary matrix constructed from C_{prep}, which has eigenvalues $e^{\pm 2i\theta_a}$.
- Quantum Phase Estimation (QPE) [Kitaev 1995]: The QPE process estimates the eigenvalue of a given unitary matrix.

Within the QAE circuit, the QPE algorithm is applied to C_{grov} to estimate θ_a. Subsequently, this estimated θ_a can be post-processed to determine the value of a.

The Grover operator C_{grov} is a key component used in Grover's search algorithm [Grover 1996] to amplify the probability amplitude of the desired output states. It consists of four components, $C_{\text{grov}} = C_{\text{orac}}; C_{\text{prep}}^{-1}; C_{\text{zero}}; C_{\text{prep}}$, and has eigenvalues $e^{\pm 2i\theta_a}$, where C_{prep}^{-1} is the inverse of C_{prep}, and C_{orac} and C_{zero} are boolean oracle and zero reflection circuits that perform a phase flip (*i.e.*, transform $|\psi\rangle$ to $-|\psi\rangle$) when the result qubit **res** is $|1\rangle$ and when **sys** and **res** are both $|0\rangle$, respectively. This C_{grov} can be easily constructed once C_{prep} is provided: **(1)** C_{prep}^{-1} is constructed by reversing the order and rotation angle of the gates in C_{prep}. **(2)** C_{orac} and C_{zero} are constructed using a single-qubit Z gate and a multi-controlled Z gate, respectively [Nielsen and Chuang 2010].

As shown in Fig. 5, QPE requires m additional qubits for evaluation, positioned at the top left of the figure, and $2^m - 1$ applications of C_{grov}, shown in the middle of the figure, to estimate θ_a with m bits of accuracy. QPE involves three procedures: First, the state of m qubits is transformed to represent an eigenvalue-related term. This involves initializing m qubits to an equal superposition state using $R_y(\pi/2)$ gates, initializing **sys** and **res** using C_{prep}, and then controlling the m qubits by different powers of C_{grov}. Second, m qubits are transformed to represent the eigenvalue using the inverse Quantum Fourier Transform (QFT) [Coppersmith 2002]. Finally, after applying the inverse QFT, the qubits are measured to produce an integer $y \in \{0, ..., 2^m - 1\}$, which is m-bit representation of the estimate $\tilde{\theta_a}$ for θ_a. From $\tilde{\theta_a}$, we can obtain the estimate $\tilde{a} = \sin^2(\tilde{\theta_a}) \in [0, 1]$.

Example. To estimate the probability of measuring $|1\rangle_{\mathsf{x}}$ in the state $|\psi_3\rangle$ of the C_{main} in Fig. 3 using QAE, we set C_{main} as C_{prep} and designate the qubit x in C_{main} as **res** in C_{prep}. By constructing C_{grov} and applying it $O(1/\epsilon)$ times along with QPE, we can estimate the probability of measuring $|1\rangle_{\mathsf{x}}$ in the state $|\psi_3\rangle$, which is 0.15 with an error of ϵ.

3 Quantum Probabilistic Model Checking

Generalizing the encoding depicted in Fig. 3, QPMC uses quantum computers to answer the PMC configuration quadratically faster than SMC. QPMC constructs C_{prep}, the core component of QAE (§2.3), to encode the PMC configuration into a quantum circuit and employs QAE to estimate the probability of the property being satisfied. It superposes multiple states and transitions them in parallel using qubits and quantum gates. QPMC then estimates the probability for the PMC configuration by applying C_{prep} to the QAE algorithm [Brassard et al. 2002]. For the translation algorithm T from PMC configurations τ to the quantum circuit $C_{\text{prep}} = \mathsf{T}(\tau)$, we prove the following property (see §5 for the proof):

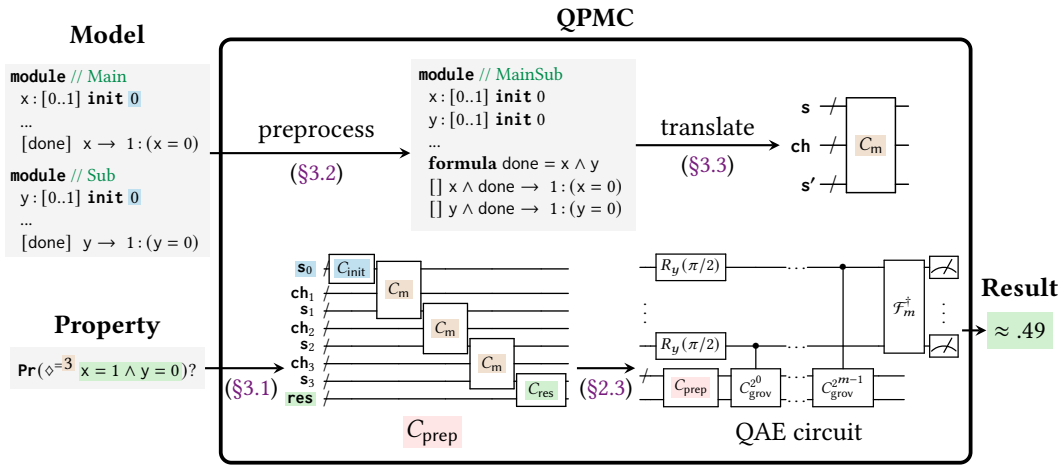


Fig. 6. Overview of QPMC.

Theorem 3.1 (Correctness of QPMC). *Let τ be a PMC configuration and* **res** *be the result qubit. We have* $[\![\tau]\!] = \mathsf{Pm}([\![\mathsf{T}(\tau)]\!](|\mathbf{0}\rangle), \mathbf{res})(1)$.

The theorem states that $[\![\tau]\!]$ equals to the probability of the result qubit being $|1\rangle$ after applying the translated circuit to the initial state $|\mathbf{0}\rangle$. Here, $\mathsf{Pm}(|\psi\rangle, q)(v)$ indicates the probability of the qubit q being $|v\rangle$ from the given state $|\psi\rangle$ (§2.2), and $|\mathbf{0}\rangle$ is the zero vector. With this property, QPMC can estimate $[\![\tau]\!]$ via QAE, achieving a quadratic speedup compared to SMC (§2.3).

3.1 Overview

Fig. 6 illustrates the QPMC process for the example presented in Fig. 1. QPMC converts a given PMC configuration $\tau = (m, \mathbf{Pr}(\diamond^{=t}\, \omega))$ into a quantum circuit C_{prep} and estimates $[\![\tau]\!]$ through the following four steps.

Preprocessing (§3.2). First, the DTMC model m is preprocessed into a unified *system module* md_{sys}, resolving the inter-module synchronization that makes it challenging to directly encode the state transitions of multiple modules into a quantum circuit. For example, Fig. 6 shows the preprocessed system module for the model in Fig. 1, which unifies the two modules `Main` and `Sub` into a single module by resolving the synchronized action `done` using formulas and additional guard conditions. This preprocessing preserves semantics, *i.e.*, $[\![m]\!] = [\![md_{\text{sys}}]\!]$, without increasing the size of the model, unlike the standard approach that incurs an exponential increase in the number of commands [PRISM 2010].

Translating model (§3.3). Second, the system module md_{sys} is translated into a quantum circuit C_{m} that mirrors its state transitions. This circuit takes a probabilistic distribution of the current state s in qubits $\mathbf{s}$ and produces that on the next state s' in qubits $\mathbf{s'}$, utilizing "choice qubits" $\mathbf{ch}$. For the set S of all possible states, their probabilistic distribution is represented with $\log_2 |S|$ qubits in $\mathbf{s}$ and $\mathbf{s'}$, enabling simultaneous representation and parallel processing of multiple states. For example, Fig. 6 shows the circuit C_{m} for the preprocessed model of Fig. 1, which encodes the variable `x` and `y` into the state qubits $\mathbf{s}$ and $\mathbf{s'}$, respectively.

Translating property. Third, $\mathtt{Pr}(\Diamond^{=t}\,\omega)$ is translated into a quantum circuit $C_{\mathsf{prep}} = \mathsf{T}(\tau)$ that mirrors the computation of $[\![(m, \mathtt{Pr}(\Diamond^{=t}\,\omega))]\!]$ as follows:

(1) Initialization Circuit $C_{\mathsf{init}} = \mathsf{T}_{\mathsf{init}}(md_{\mathsf{sys}})$: loading the preprocessed system module's initial state distribution, which is the one-hot vector of the initial state. For example, Fig. 6 shows the circuit C_{init} for the model in Fig. 1, which encodes the initial state $\{\mathsf{x} \mapsto 0, \mathsf{y} \mapsto 0\}$ into the qubits $\mathbf{s}_0$.

(2) Transition Circuit $\mathsf{T}_{\mathsf{rep}}(C_{\mathsf{m}}, t)$: applying the circuit C_{m} for t times. For exmaple, $\mathsf{T}_{\mathsf{rep}}(C_{\mathsf{m}}, 3)$ shown in Fig. 6 applies C_{m} 3 times in series.

(3) Result Circuit $C_{\mathsf{res}} = \mathsf{T}_{\mathsf{res}}(\omega, t)$: encoding the probability of reaching target states ω after t transitions into the qubit **res**. For example, C_{res} shown in Fig. 6 encodes the probability of $\mathsf{x} = 1 \wedge \mathsf{y} = 0$ being satisfied.

The C_{init} and C_{res} circuits are completely characterized by the following mappings because states form a basis for $\mathbf{s}$ (§2.2). Furthermore, both circuits are defined by classical operations such as arithmetics and comparisons so that they can be easily constructed from their classical logic gate representations [Bolhassani and Haghparast 2016; Haghparast and Bolhassani 2016; Jayashree et al. 2016; Xia et al. 2018]:

$$|0\rangle_{\mathbf{s}_0} \xmapsto{[\![C_{\mathsf{init}}]\!]} \sum_{(s \mapsto \lambda) \in \pi_0(md_{\mathsf{sys}})} \sqrt{\lambda}\,|s\rangle_{\mathbf{s}_0} \qquad \text{and} \qquad |s\rangle_{\mathbf{s}_t}\,|0\rangle_{\mathbf{res}} \xmapsto{[\![C_{\mathsf{res}}]\!]} |s\rangle_{\mathbf{s}_t}\,[\![X]\!]^{\mathbf{1}_{S_\omega}(s)}\,|0\rangle_{\mathbf{res}}\ .$$

In contrast, $\mathsf{T}_{\mathsf{rep}}(C_{\mathsf{m}}, t)$ is a proper quantum circuit with superposition, which cannot be represented in classical circuits. After applying these sub-circuits in C_{prep}, the result qubit **res** represents $[\![\tau]\!]$, meaning that the probability of measuring $|1\rangle$ in **res** is equivalent to the probability of reaching the target states ω after t transitions in τ.

Estimating the property. Finally, QPMC employs the QAE algorithm [Brassard et al. 2002] to estimate the probability of measuring $|1\rangle$ in **res**, which corresponds to $[\![\tau]\!]$. To achieve this, QPMC applies the translated circuit C_{prep} to the QAE algorithm [Brassard et al. 2002] as described in §2.3.

3.2 Preprocessing DTMC Model

QPMC's preprocessing desugars inter-module synchronization by merging all modules into a unified system module md_{sys}. For the preprocessing algorithm $\mathsf{T}_{\mathsf{pre}}$ from DTMC models to system modules, we prove that $\mathsf{T}_{\mathsf{pre}}$ preserves transition matrix (see §5 for the proof):

Lemma 3.2. *For every model m, we have $[\![m]\!] = [\![\mathsf{T}_{\mathsf{pre}}(m)]\!]$.*

Motivation. To perform PMC on a model, one first should address its inter-module synchronization because it significantly complicates the semantics. Prior work such as Prism [Kwiatkowska et al. 2011] has employed preprocessing approaches to encode synchronization with simpler constructs. However, these approaches are impractical for accelerating the model checking process in that, as more modules are synchronized, the number of commands in the preprocessed model increases exponentially, introducing a significant computational challenges at the later stages of model checking [PRISM 2010].

To address this issue, we design a preprocessing algorithm that does not increase the number of commands. To this end, we introduce a new semantics for preprocessed modules that preserves the semantics of the original model and also facilitates semantics-preserving translation to quantum circuits at the same time. Particularly, new semantics permits the execution of multiple commands at each step, which aligns with the parallel nature of quantum circuits (see §3.3 for details).

Algorithm 1 Preprocessing the given DTMC model m into md_{sys}.

```
1:  function T_pre(m : Model)
2:      info ← SyncInfo(m)                          ▷ Dictionary {Module : {Action : Comm⃗}}
3:      md_sys ← empty module md
4:      md_sys.d⃗ ← (m.md_1.d⃗ ++ ··· ++ m.md_n.d⃗)                          ▷ Add original decls
5:      for i ∈ [1, |m.md⃗|] do                                            ▷ Add original guards
6:          for (a ↦ c⃗) ∈ info[md_i] do
7:              for j ∈ [1, |c⃗|] do
8:                  AddFormula(md_sys, GetId(md_i, a, j), c⃗[j].g)
9:      for a ∈ m.Action do                                               ▷ Add action guards
10:         al ← []
11:         for i ∈ [1, |m.md⃗|] do
12:             al.add(⋁_{j∈|info[md_i][a]|} GetId(md_i, a, j))
13:         AddFormula(md_sys, a, (⋀_{e∈al} e))
14:     k ← 1
15:     for i ∈ [1, |m.md⃗|] do                                            ▷ Add sync-resolved commands
16:         for (a ↦ c⃗) ∈ info[md_i] do
17:             for j ∈ [1, |c⃗|] do
18:                 AddFormula(md_sys, g_k, GetId(md_i, a, j) ∧ a)        ▷ New guard
19:                 md_sys.c⃗[k] ← ([⊥] g_k → probabilistic updates of c⃗[j])   ▷ New command
20:                 k ← k + 1
21:     return md_sys
```

Algorithm. Algorithm 1 presents T_{pre}, which consolidates synchronized commands from different modules into a system module and encodes inter-module synchronization in additional guard conditions as follows. **(1)** T_{pre} creates a dictionary using SyncInfo, grouping commands by action names within each module (line 2). **(2)** T_{pre} copies all variables and guards from m to md_{sys}, renaming and declaring them as formulas to address synchronization (line 4-line 8). Here, GetId returns a unique identifier based on the module name, action name, and command index; and AddFormula(md_{sys}, n, g) appends a new formula to md_{sys} with the name n and the guard condition g. **(3)** T_{pre} then constructs formulas that determines action enablement (line 9-line 13). An action is enabled if all modules have an enabled command for the action. Here, whether a module has an enabled command for a given action is expressed as a disjunction (line 12); and whether all modules are enabled as a conjunction (line 13). **(4)** T_{pre} finally strengthens the guard conditions of the original commands to encode synchronization (line 14-line 20).

For example, Fig. 7a presents the system module md_{sys} preprocessed from the model shown in Fig. 1. The module md_{sys} incorporates variables `x` from `Main` and `y` from `Sub` modules, and represents their command guard conditions as formulas $\mathsf{x}_{\mathsf{do1}}$, $\mathsf{x}_{\mathsf{do2}}$, $\mathsf{x}_{\mathsf{done1}}$, and so on. For example, $\mathsf{x}_{\mathsf{do1}}$ denotes the guard condition for the first command with action `do` in `Main`. The formulas `do` and `done` represent the enablement of actions. The formulas g_1, g_2, and so on, denote the strengthened guard conditions that also encode synchronization. The commands mirror those of the original model but with these strengthened guard conditions.

Semantics. The semantics definition for preprocessed system modules differs from that for original models. Recall from Fig. 2b that the semantics $[\![m]\!]$ of the original model m involves the *sequential execution* of synchronized commands in each module. Each module identifies its enabled commands

```
module // md_sys (MainSub)
  x : [0..1] init 0, y : [0..1] init 0
  // formulas for original command guard
  formula x_do1, x_do2, x_done1 =!x,      x, x
  formula y_do1, y_do2, y_done1 =!y, y ∧ !x, y
  // formulas for action guard
  formula do = (x_do1 ∨ x_do2) ∧ (y_do1 ∨ y_do2)
  formula done = (x_done1) ∧ (y_done1)
  // formulas for new command guard
  formula g1 = x_do1   ∧    do, g2 = x_do2 ∧ do
  formula g3 = x_done1 ∧ done, g4 = y_do1 ∧ do
  formula g5 = y_do2   ∧    do, g6 = y_done1 ∧ done
  // commands of module 1
  [] g1 → 0.5 : (x = 0) + 0.5 : (x = 1)
  [] g2 →   1 : (x = 1)
  [] g3 →   1 : (x = 0)
  // commands of module 2
  [] g4 → 0.8 : (y = 0) + 0.2 : (y = 1)
  [] g5 →   1 : (y = 1)
  [] g6 →   1 : (y = 0)
```

(a) Example (from Fig. 1).

$$[\![md_{\text{sys}}]\!](j,i) = \sum_{\substack{(ch \mapsto \lambda) \in \text{CH}_i \\ \wedge \text{F}_i(ch) = j}} \lambda$$

$$\text{CH}_s = \prod_{i \in [1, |\vec{c}|] \wedge \text{G}(s)_i} c_i.\vec{\lambda}$$

$$\text{F}_s(ch) = \text{U}(s, \text{G}(s), ch)$$

$$\text{G}(s) = \vec{c}.\text{map}(|c|\ [\![c.g]\!](s))$$

$$\text{U}(s, gb, ch) = \left(\biguplus_{i \in [1, |\vec{c}|] \wedge gb_i} [\![c_i.\vec{u}[ch[\text{Mi}(c_i)]]]\!] \right)(s)$$

where $\vec{c}$ is the commands of md_{sys}

$\text{Mi}(c) \triangleq$ (the original module index of command c)

(b) Semantics.

Fig. 7. Example and semantics of the system module.

and applies the corresponding updates in sequence. Therefore, $[\![m]\!](j,i)$ is the product of the local transition probabilities of each module.

In contrast, the semantics $[\![md_{\text{sys}}]\!]$ of the preprocessed system module md_{sys} is based on the *parallel execution* of synchronized commands to replicate the individual behaviors of the original modules. Specifically, it identifies all enabled commands and simultaneously applies their updates, guided by three key functions depicted in Fig. 7b: **(1)** The guard function G, which identifies enabled commands. **(2)** The choice function CH_i, which constructs the joint probability distribution of choices for all possible updates from state i. **(3)** The update function U, which calculates the next state for a given set of updates. Thus, $[\![md_{\text{sys}}]\!](j,i)$ is defined by summing the probabilities of choices (from *all possible combinations of updates* in CH_i) that transition state i to j (with the next state j determined by U). The three functions are defined as follows.

(1) Guard: The function $\text{G} : S \to \mathbb{B}^{|\vec{c}|}$ takes the current state and returns whether each command's guard is satisfied, where $\vec{c}$ is the commands in md_{sys}.

(2) Choice: The function $\text{CH}_i : \mathbb{N}^{|\overrightarrow{md}|} \to [0,1]$ denotes the joint probability distribution of update choices at state i, where $\text{CH}_i(n_1, \ldots, n_{|\overrightarrow{md}|})$ represents the probability of choosing the n_1-th update in the first enabled command, the n_2-th in the second, and so on, with $\overrightarrow{md}$ being the modules in the original model m. Therefore, CH_i represents the joint probability distribution of all possible update combinations for each command that satisfies its guard condition.

(3) Update: The function $\text{F}_i : \mathbb{N}^{|\overrightarrow{md}|} \to S$ calculates the next state from state i and the given choice $ch \in \mathbb{N}^{|\overrightarrow{md}|}$ of updates. This function is defined by the update function $\text{U} : S \times \mathbb{B}^{|\vec{c}|} \times \mathbb{N}^{|\overrightarrow{md}|} \to S$, which takes the current state s, a boolean vector gb representing the guard conditions, and a choice ch. It calculates the next state s by applying a *chosen combination of updates* for the commands whose guards are met in the current state. Each n-th element of ch represents the index of the update to be applied in the n-th enabled command in md_{sys}. To find such an index n, we use the function $\text{Mi}(c)$, which returns the index of the element in ch for the enabled

command c. This index is determined by the original module index of the command c, as each enabled command corresponds directly with one of the modules in the original model m (§2.1).

Example. Fig. 8b presents the execution of md_{sys} shown in Fig. 7a. Consider the transition probability from $s = \{\mathsf{x} \mapsto 0, \mathsf{y} \mapsto 0\}$ to $s' = \{\mathsf{x} \mapsto 0, \mathsf{y} \mapsto 1\}$.

(1) Guard: Since !x and !y are true, the enabled commands are identified by $\mathsf{G}(s) = [1, 0, 0, 1, 0, 0]$, indicating that c_1 and c_4 are enabled.

(2) Choice: At state s, the possible choices are determined by $\mathsf{CH}_s = c_1.\vec{\lambda} \times c_4.\vec{\lambda}$, resulting in four potential choices, each associated with a specific probability at state s. For instance, $\mathsf{CH}_s((1,2)) = 0.1$ means the choice of the first update of c_1 and the second update of c_4 happens with a probability of 0.1.

(3) Update: For each choice $ch : \mathbb{N}^2$, where $ch(1)$ and $ch(2)$ are the indices of the chosen updates for the first and second enabled commands, respectively, the next state from s is computed by the function $\mathsf{F}_s(ch) = \mathsf{U}(s, \mathsf{G}(s), ch)$. For example, the enabled commands c_1 and c_4 apply the updates of the indices $ch(\mathsf{Mi}(c_1)) = ch(1)$ and $ch(\mathsf{Mi}(c_4)) = ch(2)$, respectively. Following this, $\mathsf{F}_s((1,2)) = \mathsf{U}(s, \mathsf{G}(s), (1,2))$ applies the first update of c_1 and the second update of c_4 to current state s, resulting in the next state $\{\mathsf{x} \mapsto 0, \mathsf{y} \mapsto 1\}$.

Finally, the transition probability from state s to state s' is determined by summing the probabilities of all choices from CH_s that transition s to s', which is $[\![md_{\mathsf{sys}}]\!](s', s) = \mathsf{CH}_s((1,2)) = 0.1$. This probability equals to $[\![m]\!](s', s)$ in the original model m depicted in Fig. 1.

3.3 Translating Preprocessed DTMC Model to Quantum Circuit

QPMC subsequently translates md_{sys} into a quantum circuit C_{m} that encodes $[\![md_{\mathsf{sys}}]\!]$. At a high level, C_{m} takes the current state and generates the next state's probability distribution by employing qubits for current states, guards, choices, and next states to superpose parallel execution of enabled transitions. For the translation T_{m} from system modules to quantum circuits, we prove the following property (see §5 for the proof):

Lemma 3.3. *Let md_{sys} be a system module, and $\mathbf{s}$, $\mathbf{g}$, $\mathbf{ch}$, $\mathbf{s'}$ the qubits representing its current state, guards, choices, and next state, respectively. We have:*

$$\forall s, s' \in S, [\![md_{\mathsf{sys}}]\!](s', s) = \mathsf{Pm}([\![\mathsf{T}_{\mathsf{m}}(md_{\mathsf{sys}})]\!](|s\rangle_{\mathbf{s}} |\mathbf{0}\rangle_{\mathbf{g}} |\mathbf{0}\rangle_{\mathbf{ch}} |\mathbf{0}\rangle_{\mathbf{s'}}), \mathbf{s'})(s') \ .$$

This lemma states that $[\![md_{\mathsf{sys}}]\!](s', s)$ equals to the probability of measuring s' from the next state qubits $\mathbf{s'}$ after applying the translated circuit $\mathsf{T}_{\mathsf{m}}(md_{\mathsf{sys}})$ to the input state $|s\rangle_{\mathbf{s}} |\mathbf{0}\rangle_{\mathbf{g}} |\mathbf{0}\rangle_{\mathbf{ch}} |\mathbf{0}\rangle_{\mathbf{s'}}$.

Fig. 9 overviews the circuit $C_{\mathsf{m}} = \mathsf{T}_{\mathsf{m}}(md_{\mathsf{sys}})$ that replicates the transitions of md_{sys}. It consists of four circuits that encode G, CH_s, U, and G^{-1} (§3.2):

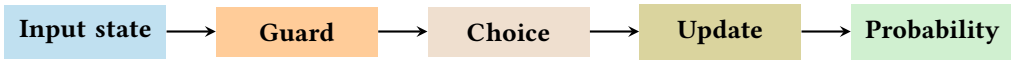
(1) Guard Circuit $C_{\mathsf{g}} = \mathsf{T}_{\mathsf{g}}(md_{\mathsf{sys}})$: encoding G to evaluates the guard conditions for state s and encodes the results into guard qubits $\mathbf{g}$.

(2) Choice Circuit $C_{\mathsf{ch}} = \mathsf{T}_{\mathsf{ch}}(md_{\mathsf{sys}})$: encoding CH_s to generates a superposition within the choice qubits $\mathbf{ch}$, representing all possible update combination choices from state s, along with their probabilities.

(3) Update Circuit $C_{\mathsf{u}} = \mathsf{T}_{\mathsf{u}}(md_{\mathsf{sys}})$: encoding U to calculates the resulting state s' in $\mathbf{s'}$ from the update combinations indicated in each basis of $\mathbf{ch}$.

(4) Guard Uncomputation Circuit $C_{\mathsf{g}}^{-1} = \mathsf{T}_{\mathsf{ginv}}(md_{\mathsf{sys}})$: encoding G^{-1} to *uncompute* [Bennett 1973] the guard qubits $\mathbf{g}$ back to their initial state $|\mathbf{0}\rangle$ for later use.

These four circuits in series map each basis state $|s\rangle$ in $\mathbf{s}$ into the probability distribution for the subsequent state $|s'\rangle$ in $\mathbf{s'}$. Recall from §3.1 that mappings from these bases fully characterizes the



(a) Execution flow of md_{sys} and $\mathsf{T_m}(md_{\text{sys}})$.

(Guard)

$$\begin{aligned} G(s) &= [\mathsf{x}_{\text{do1}} \wedge \text{do}, \mathsf{x}_{\text{do2}} \wedge \text{do}, \\ &\quad \ldots, \mathsf{y}_{\text{done1}} \wedge \text{done}] \\ &= [1, 0, 0, 1, 0, 0] \end{aligned}$$

(Choice)

$$\begin{aligned} \mathsf{CH}_s &= c_1.\vec{\lambda} \times c_4.\vec{\lambda} \\ &= \{1 \mapsto 0.5,\ 2 \mapsto 0.5\} \times \{1 \mapsto 0.8,\ 2 \mapsto 0.2\} \\ &= \{(1,1) \mapsto 0.4,\ (1,2) \mapsto 0.1,\ (2,1) \mapsto 0.4,\ (2,2) \mapsto 0.1\} \end{aligned}$$

(Update)

$$\begin{aligned} \mathsf{F}_s(ch = (1,1)) = \mathsf{U}(s, G(s), (1,1)) &= ([\![c_4.\vec{u}[ch(\mathsf{Mi}(c_4))]]\!] \circ [\![c_1.\vec{u}[ch(\mathsf{Mi}(c_1))]]\!])(s) \\ &= ([\![c_4.\vec{u}[ch(2)]]\!] \circ [\![c_1.\vec{u}[ch(1)]]\!])(s) \\ &= ([\![c_4.u_1]\!] \circ [\![c_1.u_1]\!])(s) = \{\mathsf{x} \mapsto 0, \mathsf{y} \mapsto 0\} \\ \mathsf{F}_s((1,2)) = \mathsf{U}(s, G(s), (1,2)) &= ([\![c_4.u_2]\!] \circ [\![c_1.u_1]\!])(s) = \{\mathsf{x} \mapsto 0, \mathsf{y} \mapsto 1\} \\ \mathsf{F}_s((2,1)) = \mathsf{U}(s, G(s), (2,1)) &= ([\![c_4.u_1]\!] \circ [\![c_1.u_2]\!])(s) = \{\mathsf{x} \mapsto 1, \mathsf{y} \mapsto 0\} \\ \mathsf{F}_s((2,2)) = \mathsf{U}(s, G(s), (2,2)) &= ([\![c_4.u_2]\!] \circ [\![c_1.u_2]\!])(s) = \{\mathsf{x} \mapsto 1, \mathsf{y} \mapsto 1\} \end{aligned}$$

(Probability of transitioning from $s = \{\mathsf{x} \mapsto 0, \mathsf{y} \mapsto 0\}$ to $s' = \{\mathsf{x} \mapsto 0, \mathsf{y} \mapsto 1\}$)

$$[\![md_{\text{sys}}]\!](s', s) = \sum_{(ch \mapsto \lambda) \in \mathsf{CH}_s \wedge \mathsf{F}_s(ch) = s'} \lambda \quad = \mathsf{CH}_s((1,2)) = 0.1$$

(b) Execution of system module md_{sys}.

(Guard Circuit)

$$\begin{aligned} &|00\rangle_{\mathbf{s}} |000000\rangle_{\mathbf{g}} |00\rangle_{\mathbf{ch}} |00\rangle_{\mathbf{s'}} \\ \xrightarrow{[\![C_{\mathsf{g}}]\!]} &|00\rangle_{\mathbf{s}} |\ G(s)\ \rangle_{\mathbf{g}} |00\rangle_{\mathbf{ch}} |00\rangle_{\mathbf{s'}} \\ = &|00\rangle_{\mathbf{s}} |100100\rangle_{\mathbf{g}} |00\rangle_{\mathbf{ch}} |00\rangle_{\mathbf{s'}} \\ = &|\psi_1\rangle \end{aligned}$$

(Choice Circuit)

$$\begin{aligned} |\psi_1\rangle \xmapsto{[\![C_{\text{ch}}]\!]} &\sum_{(ch \mapsto \lambda) \in \mathsf{CH}_s} \sqrt{\lambda}\, |00\rangle_{\mathbf{s}} |100100\rangle_{\mathbf{g}} |ch-1\rangle_{\mathbf{ch}} |00\rangle_{\mathbf{s'}} \\ = &\sqrt{0.4}\, |00\rangle_{\mathbf{s}} |100100\rangle_{\mathbf{g}} |0\rangle_{\mathbf{chi}_1} |0\rangle_{\mathbf{chi}_2} |00\rangle_{\mathbf{s'}} \\ &+ \sqrt{0.1}\, |00\rangle_{\mathbf{s}} |100100\rangle_{\mathbf{g}} |0\rangle_{\mathbf{chi}_1} |1\rangle_{\mathbf{chi}_2} |00\rangle_{\mathbf{s'}} \\ &+ \sqrt{0.4}\, |00\rangle_{\mathbf{s}} |100100\rangle_{\mathbf{g}} |1\rangle_{\mathbf{chi}_1} |0\rangle_{\mathbf{chi}_2} |00\rangle_{\mathbf{s'}} \\ &+ \sqrt{0.1}\, |00\rangle_{\mathbf{s}} |100100\rangle_{\mathbf{g}} |1\rangle_{\mathbf{chi}_1} |1\rangle_{\mathbf{chi}_2} |00\rangle_{\mathbf{s'}} \\ = &|\psi_2\rangle \end{aligned}$$

(Update Circuit)

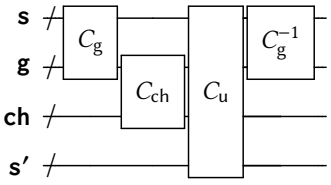
$$\begin{aligned} |\psi_2\rangle \xmapsto{[\![C_{\text{u}}]\!]} &\sqrt{0.4}\, |00\rangle_{\mathbf{s}} |100100\rangle_{\mathbf{g}} |0\rangle_{\mathbf{chi}_1} |0\rangle_{\mathbf{chi}_2} |\mathsf{U}(s, G(s), (01_2, 01_2))\rangle_{\mathbf{s'}} \\ &+ \sqrt{0.1}\, |00\rangle_{\mathbf{s}} |100100\rangle_{\mathbf{g}} |0\rangle_{\mathbf{chi}_1} |1\rangle_{\mathbf{chi}_2} |\mathsf{U}(s, G(s), (01_2, 10_2))\rangle_{\mathbf{s'}} \\ &+ \sqrt{0.4}\, |00\rangle_{\mathbf{s}} |100100\rangle_{\mathbf{g}} |1\rangle_{\mathbf{chi}_1} |0\rangle_{\mathbf{chi}_2} |\mathsf{U}(s, G(s), (10_2, 01_2))\rangle_{\mathbf{s'}} \\ &+ \sqrt{0.1}\, |00\rangle_{\mathbf{s}} |100100\rangle_{\mathbf{g}} |1\rangle_{\mathbf{chi}_1} |1\rangle_{\mathbf{chi}_2} |\mathsf{U}(s, G(s), (10_2, 10_2))\rangle_{\mathbf{s'}} \\ = &\sqrt{0.4}\, |00\rangle_{\mathbf{s}} |100100\rangle_{\mathbf{g}} |0\rangle_{\mathbf{chi}_1} |0\rangle_{\mathbf{chi}_2} |00\rangle_{\mathbf{s'}} + \sqrt{0.1}\, |00\rangle_{\mathbf{s}} |100100\rangle_{\mathbf{g}} |0\rangle_{\mathbf{chi}_1} |1\rangle_{\mathbf{chi}_2} |01\rangle_{\mathbf{s'}} \\ &+ \sqrt{0.4}\, |00\rangle_{\mathbf{s}} |100100\rangle_{\mathbf{g}} |1\rangle_{\mathbf{chi}_1} |0\rangle_{\mathbf{chi}_2} |10\rangle_{\mathbf{s'}} + \sqrt{0.1}\, |00\rangle_{\mathbf{s}} |100100\rangle_{\mathbf{g}} |1\rangle_{\mathbf{chi}_1} |1\rangle_{\mathbf{chi}_2} |11\rangle_{\mathbf{s'}} \\ = &|\psi_3\rangle \end{aligned}$$

(Probability of measuring $s' = \{\mathsf{x} \mapsto 0, \mathsf{y} \mapsto 1\} = 01_2$ on qubits $\mathbf{s'}$ after applying $\mathsf{T_m}(md_{\text{sys}})$)

$$\mathsf{Pm}([\![\mathsf{T_m}(md_{\text{sys}})]\!](|s\rangle_{\mathbf{s}} |\mathbf{0}\rangle_{\mathbf{g}} |\mathbf{0}\rangle_{\mathbf{ch}} |\mathbf{0}\rangle_{\mathbf{s'}}), \mathbf{s'})(s') = \mathsf{Pm}(|\psi_3\rangle, \mathbf{s'})(s') = 0.1$$

(c) Execution of circuit $\mathsf{T_m}(md_{\text{sys}})$.

Fig. 8. Execution example of md_{sys} and $\mathsf{T_m}(md_{\text{sys}})$ with initial state $s = \{\mathsf{x} \mapsto 0, \mathsf{y} \mapsto 0\} = 00_2$.



(a) Diagram.

$$
\begin{aligned}
& & |s\rangle_{\mathbf{s}}\,|\mathbf{0}\rangle_{\mathbf{g}} & & |\mathbf{0}\rangle_{\mathbf{ch}}\,|\mathbf{0}\rangle_{\mathbf{s'}} \\
&\xrightarrow{[\![C_g]\!]} & |s\rangle_{\mathbf{s}}\,|\mathrm{G}(s)\rangle_{\mathbf{g}} & & |\mathbf{0}\rangle_{\mathbf{ch}}\,|\mathbf{0}\rangle_{\mathbf{s'}} \\
&\xrightarrow{[\![C_{ch}]\!]} \sum_{(ch\mapsto\lambda)\in \mathrm{CH}_s} \sqrt{\lambda}\ & |s\rangle_{\mathbf{s}}\,|\mathrm{G}(s)\rangle_{\mathbf{g}} & |ch-1\rangle_{\mathbf{ch}} & |\mathbf{0}\rangle_{\mathbf{s'}} \\
&\xrightarrow{[\![C_u]\!]} \sum_{(ch\mapsto\lambda)\in \mathrm{CH}_s} \sqrt{\lambda}\ & |s\rangle_{\mathbf{s}}\,|\mathrm{G}(s)\rangle_{\mathbf{g}} & |ch-1\rangle_{\mathbf{ch}} & |\mathrm{U}(s,\mathrm{G}(s),ch)\rangle_{\mathbf{s'}} \\
&\xrightarrow{[\![C_g^{-1}]\!]} \sum_{(ch\mapsto\lambda)\in \mathrm{CH}_s} \sqrt{\lambda}\ & |s\rangle_{\mathbf{s}}\,|\mathbf{0}\rangle_{\mathbf{g}} & |ch-1\rangle_{\mathbf{ch}} & |\mathrm{U}(s,\mathrm{G}(s),ch)\rangle_{\mathbf{s'}}
\end{aligned}
$$

(b) Semantics (G, CH_s, U are functions of md_{sys}).

Fig. 9. Quantum circuit C_{m} translated from system module md_{sys}.

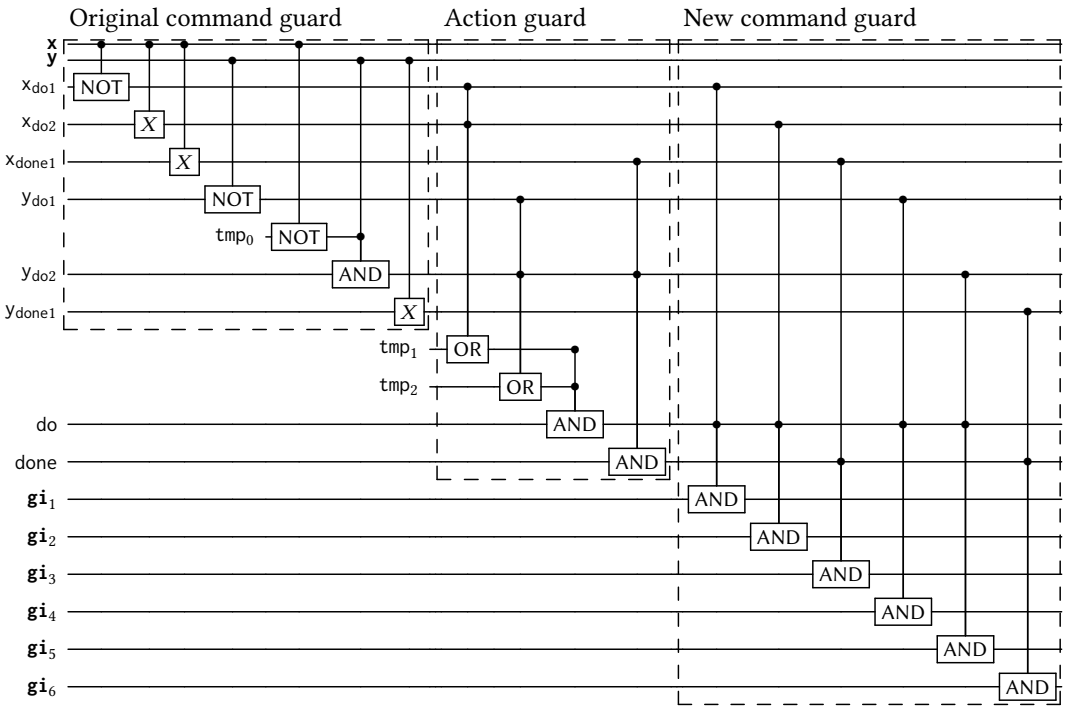
circuit for parallel processing of states. Now we explain these four circuits in details. From now on, we will omit writing md_{sys} in texts and expressions when their association with md_{sys} is clear from the context.

Guards. The circuit C_{g} encodes G that evaluates the guards for all commands: $[\![C_{\mathrm{g}}]\!]$ transforms the input state $|s\rangle_{\mathbf{s}}\,|\mathbf{0}\rangle_{\mathbf{g}}$ into the output state $|s\rangle_{\mathbf{s}}\,|\mathrm{G}(s)\rangle_{\mathbf{g}}$. Since G comprises classical operations, we can easily construct C_{g} as we did for C_{init} and C_{res} in §3.1. For example, Fig. 10a illustrates the circuit C_{g} for the system module md_{sys} in Fig. 7a. It replicates the guard function G of md_{sys} in guard qubits by mirroring each formula's expressions in quantum gates. Here, NOT, AND, OR represent the corresponding classical logic gates (Fig. 11).

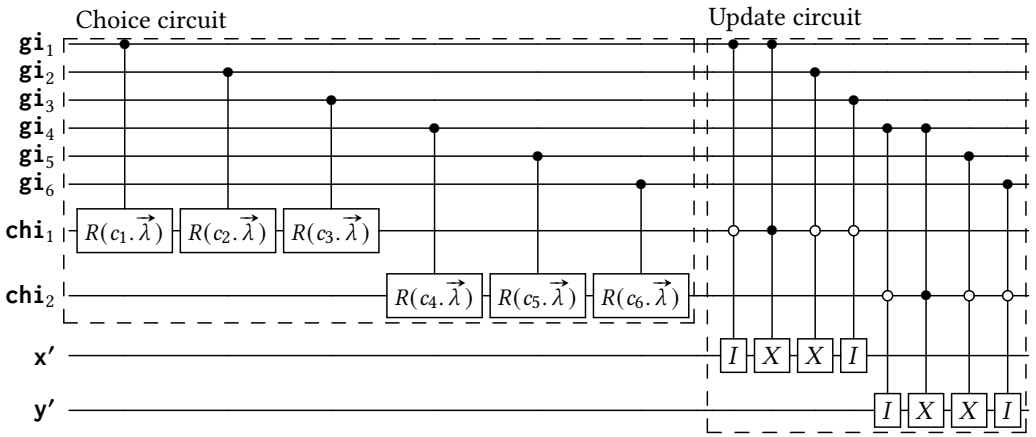
Additionally, after replicating state transitions using C_{ch} and C_{u}, QPMC uncomputes [Bennett 1973] the state of $\mathbf{g}$ back to its initial state $|\mathbf{0}\rangle$ using C_{g}^{-1}, enabling the reuse of $\mathbf{g}$ in subsequent state transitions. Uncomputation is necessary for reuse because quantum physics does not permit to overwrite qubits unlike classical bits. The uncomputation process, which involves applying the gates in C_{g} in reverse order, can be safely done without impacting other qubits if **(1)** they are constructed by a circuit with classical semantics; and **(2)** the qubits used to construct the qubits are still available [Bennett 1973; Bichsel et al. 2020]. Formally, for a circuit C whose semantics are defined by a classical function f, the transformation $|x\rangle\,|\mathbf{0}\rangle \xrightarrow{[\![C]\!]} |x\rangle\,|f(x)\rangle$ can be reversed to $|x\rangle\,|f(x)\rangle \xrightarrow{[\![C^{-1}]\!]} |x\rangle\,|\mathbf{0}\rangle$, where the inverse circuit C^{-1} applies each gate in C in the reverse order.

Choices. The circuit C_{ch} encodes CH_s that enumerates all possible choices along with their probabilities for the current state s: $[\![C_{\mathrm{ch}}]\!]$ transforms the input state $|gb\rangle_{\mathbf{g}}\,|\mathbf{0}\rangle_{\mathbf{ch}}$ into a superposed state $\sum_{(ch\mapsto\lambda)\in \mathrm{CH}_s} \sqrt{\lambda}\,|gb\rangle_{\mathbf{g}}\,|ch-1\rangle_{\mathbf{ch}}$, where gb is the boolean vector of guard results generated by C_{g}. The circuit is a concatenation of individual circuits that set up choices for each enabled command.

For example, the left side of Fig. 10b shows the circuit C_{ch} translated from md_{sys} shown in Fig. 7a. The left three gates set the probabilities for the first, second, and third commands c_1, c_2, and c_3 of md_{sys} in the choice qubit $\mathbf{chi}_{\mathrm{Mi}(c_1)} = \mathbf{chi}_{\mathrm{Mi}(c_2)} = \mathbf{chi}_{\mathrm{Mi}(c_3)} = \mathbf{chi}_1$, when the corresponding guard qubit $\mathbf{gi}_1$, $\mathbf{gi}_2$, and $\mathbf{gi}_3$ are in the state $|1\rangle$, respectively. $\mathbf{chi}_{\mathrm{Mi}(c)}$ refers to the qubits allocated for choices of the command c, which are shared among all commands originally in the same module in m. Similarly, the right three gate sets the probabilities for the fourth, fifth, and sixth commands c_4, c_5, and c_6 of md_{sys} in the choice qubit $\mathbf{chi}_{\mathrm{Mi}(c_4)} = \mathbf{chi}_{\mathrm{Mi}(c_5)} = \mathbf{chi}_{\mathrm{Mi}(c_6)} = \mathbf{chi}_2$, when the

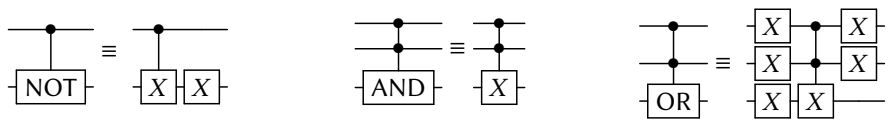


(a) Guard circuit.



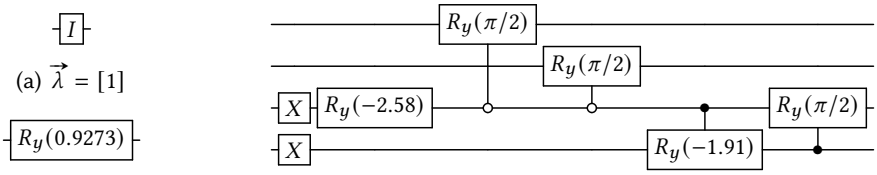
(b) Choice and update circuit.

Fig. 10. C_{m} example (from Fig. 7a). $\mathbf{x}, \mathbf{y}$ and $\mathbf{x'}, \mathbf{y'}$ are the current and next state qubits, respectively. $\mathbf{gi}_n$ is the guard qubit for the n-th command, and $\mathbf{chi}_n$ are the choice qubit for the n-th enabled command, which are shared among all guard qubits originating from the same original module. Other qubits, such as $\mathsf{x}_{\mathsf{do1}}$ and $\mathtt{tmp}_0$ are auxiliary qubits to store intermediate results of guard conditions. C_{g}^{-1} is omitted for brevity.



(a) NOT (b) AND (c) OR

Fig. 11. Quantum circuit for classical logic gates.



(a) $\vec{\lambda} = [1]$ (b) $\vec{\lambda} = [0.8, 0.2]$. (c) $\vec{\lambda} = [1/13, \ldots, 1/13]$ (uniform) [Shukla and Vedula 2024].

Fig. 12. $R(\vec{\lambda})$ gate for preparing probability distribution $\vec{\lambda}$ with semantics $[\![R(\vec{\lambda})]\!](|\mathbf{0}\rangle) = \sum_{i=0}^{|\vec{\lambda}|-1} \sqrt{\lambda_i}\,|i\rangle$.

corresponding guard qubit $\mathbf{gi}_4$, $\mathbf{gi}_5$, and $\mathbf{gi}_6$ are in the state $|1\rangle$, respectively. Here, $R(\vec{\lambda})$ is a gate that establishes the probability distribution $\vec{\lambda}$ on qubits (see §A in the supplementary material [Jeon et al. 2024] for the full definition).

Formally, $C_{\mathsf{ch}} = \mathsf{T}_{\mathsf{ch}}(md_{\mathsf{sys}}) = \mathbin{+\!\!+}_{n=1}^{|\vec{c}|} Ct^1(R(c_n.\vec{\lambda}))(\mathbf{gi}_n, \mathbf{chi}_{\mathsf{Mi}(c_n)})$, with $+\!\!+$ for list concatenation and c_n as the n-th command in md_{sys}. $\mathbf{ch}$ is represented as $\bigotimes_{n=1}^{|\overrightarrow{md}|} \mathbf{chi}_n$, where $\overrightarrow{md}$ is the modules in m. The safety of sharing choice qubits is ensured because the guards of the same md are mutually exclusive (§2.1). For example, as depicted in Fig. 10b, the three gates on the left target the same qubits, $\mathbf{chi}_1$. However, at any given basis state, exactly one of their control qubits, $\mathbf{gi}_1$, $\mathbf{gi}_2$, and $\mathbf{gi}_3$, is in the state $|1\rangle$, ensuring that only one of the three gates is activated.

While constructing the $R(\vec{\lambda})$ gate can be challenging for arbitrary probability distributions [Shende et al. 2005], it is feasible for the PMC models of interest. PMC models typically involve regular probability distributions, such as uniform distributions, and the size of these distributions is generally small. For example, all DTMC models in the Prism benchmark suite [Kwiatkowska et al. 2012] including the case studies in §7.2 have either uniform distributions or non-uniform distributions with a size of 2. Uniform distribution with size n can be prepared using $O(\log_2 n)$ elementary gates [Shukla and Vedula 2024] and non-uniform distribution with size 2 can be prepared using single $R_y(\theta)$ gate as shown in Fig. 12.

Updates. The circuit C_{u} encodes U that generates the next state from the current state, guard conditions, and probabilistic choices: $[\![C_{\mathsf{u}}]\!]$ transforms the input state $|s\rangle_{\mathbf{s}} |gb\rangle_{\mathbf{g}} |ch\rangle_{\mathbf{ch}} |\mathbf{0}\rangle_{\mathbf{s'}}$ into the output state $|s\rangle_{\mathbf{s}} |gb\rangle_{\mathbf{g}} |ch\rangle_{\mathbf{ch}} |\mathsf{U}(s, gb, ch)\rangle_{\mathbf{s'}}$. Since U comprises classical operations, we can easily construct C_{u}.

For example, the right side of Fig. 10b illustrates the circuit C_{u} for the system module md_{sys} in Fig. 7a. The left two gates generate the next state for two choices of the first command c_1, and the third gate does so for single choice of the second command c_2. Specifically, the second gate $Ct^{11_2}(X)([\mathbf{gi}_1, \mathbf{chi}_1], \mathbf{x'})$ updates the basis of $\mathbf{x'}$ from $|0\rangle$ to $|1\rangle$ when the guard qubit $\mathbf{gi}_1$ and choice qubit $\mathbf{chi}_1$ are both in state $|1\rangle$, replicating the second update of the first command c_1 ($\mathsf{x} = 1$). Similarly, each subsequent gate replicates each update of all commands.

Example. Fig. 8c shows the execution of $\mathsf{T}_{\mathsf{m}}(md_{\mathrm{sys}})$ for md_{sys} shown in Fig. 7a. The focus is on the probability of measuring the next state $s' = \{\mathsf{x} \mapsto 0, \mathsf{y} \mapsto 1\} = 01_2$ on the qubits $\mathbf{s'}$ after C_{m} is applied to the initial state $s = \{\mathsf{x} \mapsto 0, \mathsf{y} \mapsto 0\} = 00_2$. Auxiliary qubits are omitted for simplicity. Initially, the quantum state is configured as $|00\rangle_{\mathbf{s}} |000000\rangle_{\mathbf{g}} |00\rangle_{\mathbf{ch}} |00\rangle_{\mathbf{s'}}$, with $|00\rangle_{\mathbf{s}}$ representing the initial state s, $|000000\rangle_{\mathbf{g}} = |0\rangle_{\mathbf{gi}_1} \ldots |0\rangle_{\mathbf{gi}_6}$ for the guards of six commands, $|00\rangle_{\mathbf{ch}} = |0\rangle_{\mathbf{chi}_1} |0\rangle_{\mathbf{chi}_2}$ for the choices for the two enabled commands, and $|00\rangle_{\mathbf{s'}}$ for the next state s', all presented in binary representation.

(1) **Guard**: The C_{g} operation identifies enabled commands, transforming $|00\rangle_{\mathbf{s}} |000000\rangle_{\mathbf{g}}$ into $|00\rangle_{\mathbf{s}} |100100\rangle_{\mathbf{g}}$. This indicates that c_1 and c_4 are enabled.

(2) **Choice**: The C_{ch} operation superposes the possible choices of the enabled commands at $|00\rangle_{\mathbf{s}}$. It maps $|100100\rangle_{\mathbf{g}} |0\rangle_{\mathbf{chi}_1} |0\rangle_{\mathbf{chi}_2}$ into $|100100\rangle_{\mathbf{g}} (\sqrt{0.4} |0\rangle_{\mathbf{chi}_1} |0\rangle_{\mathbf{chi}_2} + \ldots + \sqrt{0.1} |1\rangle_{\mathbf{chi}_1} |1\rangle_{\mathbf{chi}_2})$. This results in four possible choices with their respective probabilities. For example, the last term $\sqrt{0.1} |1\rangle_{\mathbf{chi}_1} |1\rangle_{\mathbf{chi}_2}$ means that the second update ($|1_2\rangle$) of the first enabled command ($\mathbf{chi}_1$) and the second enabled command ($\mathbf{chi}_2$) are selected with a probability of 0.1.

(3) **Update**: The C_{u} operation determines the next state for each choice $|ch_1\rangle_{\mathbf{chi}_1} |ch_2\rangle_{\mathbf{chi}_2}$ by transforming the basis of $\mathbf{s'}$ given the bases of $\mathbf{s}$, $\mathbf{g}$, and $\mathbf{ch}$. For instance, C_{u} updates the basis of $\mathbf{s'}$ from $|00\rangle$ to $|01\rangle$ in $|00\rangle_{\mathbf{s}} |100100\rangle_{\mathbf{g}} |0\rangle_{\mathbf{chi}_1} |1\rangle_{\mathbf{chi}_2} |00\rangle_{\mathbf{s'}}$, and from $|00\rangle$ to $|10\rangle$ in $|00\rangle_{\mathbf{s}} |100100\rangle_{\mathbf{g}} |1\rangle_{\mathbf{chi}_1} |0\rangle_{\mathbf{chi}_2} |00\rangle_{\mathbf{s'}}$.

The probability of measuring $|01\rangle_{\mathbf{s'}}$ in the output state is determined by summing the probabilities of all basis states containing $|01\rangle$ in $\mathbf{s'}$. In this example, only one basis $|00\rangle_{\mathbf{s}} \ldots |0\rangle_{\mathbf{chi}_1} |1\rangle_{\mathbf{chi}_2} |01\rangle_{\mathbf{s'}}$ exist with a probability of 0.1, which equals to $[\![md_{\mathrm{sys}}]\!](s', s)$ as shown in Fig. 8b.

Note that the guard uncomputation circuit C_{g}^{-1} is not shown in the figure, but it resets the guard qubits $\mathbf{g}$ to $|000000\rangle_{\mathbf{g}}$ after the update operation without affecting the probability of measuring $|s'\rangle_{\mathbf{s'}}$ for any $s' \in S$.

4 Optimizing Number of Qubits using Uncomputation

While $C_\tau = \mathsf{T}(\tau)$ correctly solves the PMC configuration τ, managing a large number of transition steps becomes increasingly demanding due to the linear increase in the required state qubits $\mathbf{s}$ and choice qubits $\mathbf{ch}$. This is significant as qubits are a key resource in quantum computing.

To address this issue, we focus on minimizing the number of state qubits, which can be uncomputed [Bennett 1973] and reused, unlike choice qubits. QPMC employs a *batch recycling strategy* to reduce the number of state qubits from $O(t)$ to $O(\sqrt{t})$. This significantly reduces the overall number of qubits (§7.2) because the number of state qubits is typically larger than that of choice qubits, due to the latter's sharing among commands. Specifically, we design a qubit-optimized translation algorithm $\mathsf{T}_{\mathrm{opt}}$ that satisfies the following property (see §5 for the proof):

Theorem 4.1 (Correctness of QPMC with Qubit Optimization). *Let τ be a PMC configuration and $\mathbf{res}$ be the result qubit. We have* $[\![\tau]\!] = \mathsf{Pm}([\![\mathsf{T}_{\mathrm{opt}}(\tau)]\!](|\mathbf{0}\rangle), \mathbf{res})(1)$.

4.1 Recycling State Qubits

Similar to the uncomputation of guard qubits $\mathbf{g}$ (§3.3), state qubits $\mathbf{s}$ can also be uncomputed and recycled to reduce the total number of state qubits needed for multiple transitions. Recall from §3.3 that qubits can be uncomputed if **(1)** their states are constructed by a circuit with classical semantics, and **(2)** the corresponding input qubits, used to construct these states, remain available. Since the update circuit C_{u} operates classically, its outcome can be reversed to recycle the state qubits that are no longer needed for subsequent transitions, as long as the input qubits remain available.

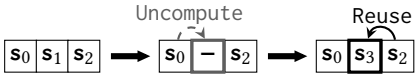


Fig. 13. Recycling state qubits.

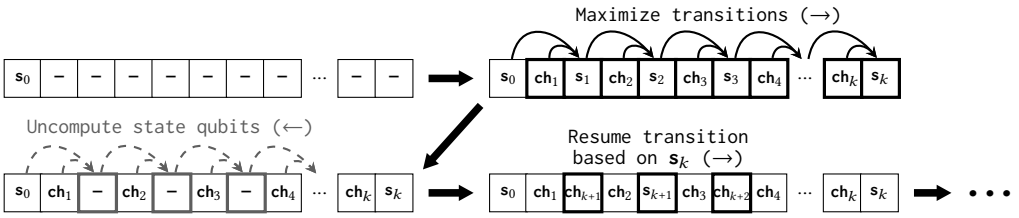


Fig. 14. Batch recycling with forward transitions and backward uncomputation.

For example, Fig. 13 shows the recycling of state qubits $\mathbf{s}_1$ when transitioning from $\mathbf{s}_0$ to $\mathbf{s}_3$. Here, other qubits are omitted for simplicity. **(1)** Suppose the transition from $\mathbf{s}_0$ to $\mathbf{s}_1$ and then $\mathbf{s}_2$ has been computed. **(2)** $\mathbf{s}_1$ can be safely uncomputed because it is no longer needed; its construction from $\mathbf{s}_0$, $\mathbf{g}$[4] and $\mathbf{ch}_1$ (not shown in the figure) is classical; and the input qubits $\mathbf{s}_0$, $\mathbf{g}$, and $\mathbf{ch}_1$ are still available. Here, the uncomputation of $\mathbf{s}_1$ is achieved by applying the circuit C_{u}^{-1} to the same input qubits $\mathbf{s}_0$, $\mathbf{g}$, $\mathbf{ch}_1$. **(3)** Subsequently, the uncomputed state qubits $\mathbf{s}_1$ can be reused for later quantum gates.

4.2 Batch Recycling

However, immediately recycling unnecessary state qubits is not always efficient. In Fig. 13, $\mathbf{s}_2$ becomes impossible to recycle because its previous state $\mathbf{s}_1$ is no longer available. Such immediate recycling leads to a significant number of non-reusable state qubits.

To improve qubit efficiency, QPMC utilizes a batch recycling strategy, illustrated in Fig. 14, which reduces the number of non-reusable state qubits and maximizes overall qubit utilization. Specifically, the strategy proceeds as follows: **(1) Transitions (→)**: State qubits from $\mathbf{s}_1$ to $\mathbf{s}_k$ and their corresponding input choices are sequentially computed to utilize all available qubits, where k is the limit of achievable transitions. **(2) Uncomputation (←)**: State qubits from $\mathbf{s}_1$ to $\mathbf{s}_{k-1}$ are uncomputed in reverse order, preserving $\mathbf{s}_k$ for the next set of transitions. Note that choices cannot be uncomputed because they are not evaluated by classical operations. **(3)** Subsequently, the state qubits $\mathbf{s}_1$ to $\mathbf{s}_{k-1}$ can be reused for later transitions ($\mathbf{ch}_{k+1}$, $\mathbf{s}_{k+1}$, …), from the last state qubits $\mathbf{s}_k$. We repeat this process until the final state qubits $\mathbf{s}_t$ is reached.

4.3 Space Complexity of State Qubits

By repeating batch recycling, the optimized circuit reaches s_t using only $O(\sqrt{t})$ state qubits. Now we analyze this space complexity of state qubits.

Let $n(t)$ be the minimum number of qubits required to complete t transitions with batch recycling; and suppose the transitions are divided into b batches, the i-th batch being with t_i transitions, where $\sum_{i=1}^{b} t_i = t$. After all transitions are completed, we will observe that the $n(t)$ qubits are divided into t choice qubits and b state qubits. This is because choice qubits are necessary for each step and cannot be uncomputed, thus we require t choice qubits. In contrast, only one non-reusable

[4]Guard qubits $\mathbf{g}$ are uncomputed for each step so that they can be reused across multiple transitions. (§3.3). To uncompute state qubits, the corresponding guard qubits must be temporarily recomputed.

state qubit remains after each batch, so the total number of state qubits is b. As such:

$$n(t) = CH \times t + ST \times b\,,$$

where CH and ST represent the size of single choice qubits and state qubits, respectively.

From this formula, we can calculate the upper bound of the number b of batches, relative to the minimum number of transitions t_i for each i-th batch. In the first batch, t_1 is at least b since we have b state qubits and enough choice qubits. After b transitions, b choice qubits are utilized, but $t - b$ choice qubits are not utilized. Hence, we can do more transitions using these idle choice qubits in the first batch, which means $t_1 \geq b$. In the second batch, t_2 is at least $b - 1$ since $b - 1$ state qubits are uncomputed in the first batch. Generalizing this, we have:

$$b - i + 1 \leq t_i\,.$$

By summing up the inequalities, we have:

$$\frac{b(b+1)}{2} = \sum_{i=1}^{b}(b - i + 1) \leq \sum_{i=1}^{b} t_i = t\,.$$

Then $b \in O(\sqrt{t})$ and thus we require $O(\sqrt{t})$ state qubits.

5 Correctness

We prove the correctness of QPMC and its optimization techniques. Specifically, we sketch the proofs of Theorem 3.1 and Theorem 4.1. For the full proof, we refer the reader to §B in the supplementary material [Jeon et al. 2024].

5.1 Correctness Proof of QPMC

Theorem 3.1 (Correctness of QPMC). *Let τ be a PMC configuration and* **res** *be the result qubit. We have* $[\![\tau]\!] = \mathsf{Pm}([\![\mathsf{T}(\tau)]\!](|\mathbf{0}\rangle), \mathbf{res})(1)$.

The key idea of the proof is using the fact that the transition circuit represents the transition probability matrix that correctly accumulates the transition probabilities. To see concretely, let $\tau = (m, \mathbf{Pr}(\Diamond^{=t}\,\omega))$, π_0 be the initial state distribution of model m, $md_{\mathsf{sys}} = \mathsf{T}_{\mathsf{pre}}(m)$, and $\mathbf{s}_0$, $\mathbf{s}_t$ be the qubits for the initial and final states after t transitions, respectively. Then we have:

$$\begin{aligned}
&\mathsf{Pm}([\![\mathsf{T}(\tau)]\!](|\mathbf{0}\rangle), \mathbf{res})(1)\\
&= \mathsf{Pm}(([\![\mathsf{T}_{\mathsf{res}}(\omega, t)]\!] \circ [\![\mathsf{T}_{\mathsf{rep}}(\mathsf{T}_{\mathsf{m}}(md_{\mathsf{sys}}), t)]\!] \circ [\![\mathsf{T}_{\mathsf{init}}(md_{\mathsf{sys}})]\!])(|\mathbf{0}\rangle), \mathbf{res})(1) && (①\text{ by the definition of T})\\
&= \mathsf{Pm}(([\![\mathsf{T}_{\mathsf{res}}(\omega, t)]\!] \circ [\![\mathsf{T}_{\mathsf{rep}}(\mathsf{T}_{\mathsf{m}}(md_{\mathsf{sys}}), t)]\!])(|\psi_0\rangle), \mathbf{res})(1) && (②\text{ by property of } \mathsf{T}_{\mathsf{init}})\\
&= \mathsf{Pm}([\![\mathsf{T}_{\mathsf{res}}(\omega, t)]\!](|\psi_1\rangle), \mathbf{res})(1) && (③\text{ by Lemma 5.1})\\
&= \mathsf{Res}(\omega, [\![md_{\mathsf{sys}}]\!]^t \pi_0) && (④\text{ by Lemma 5.2})\\
&= \mathsf{Res}(\omega, [\![m]\!]^t \pi_0) && (⑤\text{ by Lemma 3.2})\\
&= [\![\tau]\!]\,, && (⑥\text{ by the definition of } [\![\tau]\!])
\end{aligned}$$

where $|\psi_0\rangle = [\![\mathsf{T}_{\mathsf{init}}(md_{\mathsf{sys}})]\!](|\mathbf{0}\rangle)$ and $|\psi_1\rangle = [\![\mathsf{T}_{\mathsf{rep}}(\mathsf{T}_{\mathsf{m}}(md_{\mathsf{sys}}), t)]\!](|\psi_0\rangle)$ satisfy $\mathsf{Pm}(|\psi_0\rangle, \mathbf{s}_0) = \pi_0$ and $\mathsf{Pm}(|\psi_1\rangle, \mathbf{s}_t) = [\![md_{\mathsf{sys}}]\!]^t \pi_0$, respectively.

Now we explain each step in more detail. First, eq. ② is straightforward by the definition of $\mathsf{T}_{\mathsf{init}}(md_{\mathsf{sys}})$: we ensure that $|\psi_0\rangle$ satisfies $\mathsf{Pm}(|\psi_0\rangle, \mathbf{s}_0) = \pi_0$. For eq. ③, we prove Lemma 5.1 that demonstrates $\mathsf{Pm}(|\psi_1\rangle, \mathbf{s}_t) = [\![md_{\mathsf{sys}}]\!]^t \pi_0$:

Lemma 5.1. *Let md_{sys} be a system module, π a state distribution, $t \in \mathbb{N}$. If the quantum state $|\psi\rangle$ satisfies $\mathsf{Pm}(|\psi\rangle, \mathbf{s}_0) = \pi$, then:*

$$[\![md_{\mathsf{sys}}]\!]^t \pi = \mathsf{Pm}([\![\mathsf{T}_{\mathsf{rep}}(\mathsf{T}_{\mathsf{m}}(md_{\mathsf{sys}}), t)]\!](|\psi\rangle\, |\mathbf{0}\rangle_{\mathbf{ch}_1}\, |\mathbf{0}\rangle_{\mathbf{s}_1} \dots |\mathbf{0}\rangle_{\mathbf{ch}_t}\, |\mathbf{0}\rangle_{\mathbf{s}_t}), \mathbf{s}_t)\ ,$$

where $\mathbf{ch_i}$ and $\mathbf{s_i}$, $i > 0$, are qubits that do not belong to $|\psi\rangle$.

This lemma states that after applying $\mathsf{T}_{\mathsf{rep}}(\mathsf{T}_{\mathsf{m}}(md_{\mathsf{sys}}), t)$ to $|\psi\rangle\, |\mathbf{0}\rangle_{\mathbf{ch}_1}\, |\mathbf{0}\rangle_{\mathbf{s}_1} \dots |\mathbf{0}\rangle_{\mathbf{ch}_t}\, |\mathbf{0}\rangle_{\mathbf{s}_t}$, the final state $\mathbf{s}_t$ correctly encodes the probability distribution of the system module md_{sys} after t transitions. It is proven by induction on t and using Lemma 3.3, which is the key for proving the base case of single state transition.

For eq. ④, we prove Lemma 5.2 that shows the final result circuit encodes the desired probability in the result qubit **res**:

Lemma 5.2. *Let $\mathbf{Pr}(\Diamond^{=t}\, \omega)$ be a property and π be a state distribution. If the input quantum state $|\psi\rangle$ satisfies $\mathsf{Pm}(|\psi\rangle, \mathbf{s_t}) = \pi$ and **res** is qubit that belong to $|\psi\rangle$ with initial state $|0\rangle_{\mathbf{res}}$, then:*

$$\mathsf{Res}(\omega, \pi) = \mathsf{Pm}([\![\mathsf{T}_{\mathsf{res}}(\omega, t)]\!](|\psi\rangle), \mathbf{res})(1)\ .$$

This lemma states that after applying $\mathsf{T}_{\mathsf{res}}(\omega, t)$ to $|\psi\rangle$, the probability of the target state ω after t transitions is correctly encoded in the probability of measuring $|1\rangle$ on the result qubit **res**. It is proven by unfolding the definition of $\mathsf{T}_{\mathsf{res}}(\omega, t)$.

For eq. ⑤, we prove Lemma 3.2 that shows the semantics of the system module md_{sys} is equivalent to that of the original model m:

Lemma 3.2. *For every model m, we have $[\![m]\!] = [\![\mathsf{T}_{\mathsf{pre}}(m)]\!]$.*

This lemma is proven by unfolding the semantics of the resulting system module md_{sys} and exploiting the fact that, in the original model m, only one command is enabled per module at each step. As such, the product-sum resulting from unfolding the semantics $[\![md_{\mathsf{sys}}]\!]$ of the system module equals to that of $[\![m]\!]$ for the original model.

5.2 Correctness Proof of Optimization

Theorem 4.1 (Correctness of QPMC with Qubit Optimization). *Let τ be a PMC configuration and **res** be the result qubit. We have $[\![\tau]\!] = \mathsf{Pm}([\![\mathsf{T}_{\mathsf{opt}}(\tau)]\!](|\mathbf{0}\rangle), \mathbf{res})(1)$.*

The proof is similar to that of Theorem 3.1, only differing in the use of Lemma 5.3 instead of Lemma 5.1 for the state transitions of system module md_{sys} with qubit optimizations. We define $\mathsf{T}_{\mathsf{optrep}}$ as the optimized version of $\mathsf{T}_{\mathsf{rep}}$ and FreeQubits as a function that yields a list of free qubits which are used for the batch recycling strategy (see §A.2 in the supplementary material [Jeon et al. 2024] for the full definition).

Lemma 5.3. *Let md_{sys} be a system module and π a state distribution. If the input quantum state $|\psi\rangle$ satisfies $\mathsf{Pm}(|\psi\rangle, \mathbf{s}_0) = \pi$, and $\mathbf{s}_t$ and $\boldsymbol{\beta} = \mathsf{FreeQubits}(C_{\mathsf{m}}, t)$ are qubits and free qubits respectively, which do not belong to $|\psi\rangle$, then*

$$[\![md_{\mathsf{sys}}]\!]^t \pi = \mathsf{Pm}([\![\mathsf{T}_{\mathsf{optrep}}(\mathsf{T}_{\mathsf{m}}(md_{\mathsf{sys}}), t)]\!](|\psi\rangle\, |\mathbf{0}\rangle_{\boldsymbol{\beta}}\, |\mathbf{0}\rangle_{\mathbf{s}_t}), \mathbf{s}_t)\ .$$

This lemma is proven in two steps: **(1)** dividing the t transitions into n batches where the total sum of the transitions done in each batch is equal to t (*i.e.*, $t = \sum_{i=1}^{n} t_i$ where t_i denote the number of transitions in the i-th batch); and **(2)** proving that the state transition of i-th batch corresponds to the state transition of system module t_i times with qubit recycling.

To prove the second step, we explicitly categorize the free qubits in each batch into state qubits and choice qubits. This allows us to safely apply forward transitions ($\rightarrow$) and backward uncomputations ($\leftarrow$) to the corresponding type of qubits in each batch. Subsequently, all state qubits uncomputed by the backward uncomputations become the free qubits for the next batch.

6 Implementation

We implement QPMC's translation algorithm T (§3) and $\mathsf{T}_{\mathsf{opt}}$ (§4) using the Qiskit [Abraham et al. 2019] framework for quantum computing.[5] Our implementation leverages Qiskit's classical circuit components, such as adders [Vedral et al. 1996], and arbitrary state preparation component [Shende et al. 2005] to build the $R(\vec{\lambda})$ gate (§3.3). Additionally, we use Unqomp [Paradis et al. 2021] to automatically insert uncomputation circuits.

For the optimized translation algorithm $\mathsf{T}_{\mathsf{opt}}$, we start with a binary search to determine the minimum number of free qubits required for completing t transitions using batch recycling (§4). With these qubits, we recursively apply forward transitions ($\rightarrow$) and backward uncomputations ($\leftarrow$) until all t transitions are completed.

Validation. To validate the correctness of QPMC, we compare the results of the end-to-end simulation of QPMC with those obtained from the Prism model checker [Kwiatkowska et al. 2011]. We connect the circuit C_{prep} generated by T to the QAE algorithm in Qiskit, and simulate the entire model checking process via classical simulation.

Given the limitations of classical simulation to small numbers of qubits, we consider small DTMC models up to 29 qubits, such as the following model with property $\mathbf{Pr}(\Diamond^{=t}\, \mathsf{x} = 1 \wedge \mathsf{y} = 1)$ for $t = 1, 2$:

```
module                                    module
 x: [0..1] init 0                          y: [0..1] init 0
 [do] !x → 0.4: (x = 0) + 0.6: (x = 1)     [do] 1 → 0.3: (y = 0) + 0.7: (y = 1)
 [do]  x → 0.25: (x = 0) + 0.75: (x = 1)
```

We observe that the probabilities estimated by QPMC closely match the exact probabilities calculated by Prism, with the result matching up to 10^{-10} for all tested cases.[5]

7 Analysis

We assess the performance of QPMC with the following research questions:

RQ1 Does QPMC have better time complexity than classical PMC algorithms?
RQ2 Does the optimization in §4 improve qubit space usage?

To answer **RQ1**, we compare the time complexities of QPMC and classical methods (§7.1). To answer **RQ2**, we analyze the optimization's effect on the number of qubits and gates, both in complexities and in case studies (§7.2). For the detailed complexity analysis of QPMC, we refer the readers to §C in the supplementary material [Jeon et al. 2024].

7.1 Time Complexity

To address **RQ1**, we compare the performance of QPMC with that of SMC. Both methods handle large models, which pose challenges for non-statistical approaches, and yield probabilistic answers, with accuracy improving as computation time increases. For the same error bound ϵ (< 1) and confidence level, QPMC requires quadratically fewer samples than SMC. For analysis, we assume m is a DTMC model; S is the set of states in m; and t is the number of transition steps.

- QPMC's time complexity is $O(t|m|/\epsilon)$ from: **(1)** $O(1/\epsilon)$: the required number of samples (circuit executions) for the desired error bound ϵ in the QAE algorithm [Brassard et al. 2002]; and **(2)** $O(t|m|)$: the single circuit execution time derived from the required number t of state transitions and the number of quantum gates $O(|m|)$ in transition circuit C_{m}.

[5] Available in the supplementary material [Jeon et al. 2024].

- SMC's time complexity is $O(t|m|/\epsilon^2)$ from: **(1)** $O(1/\epsilon^2)$: the required number of samples for the desired error bound ϵ in the Chernoff–Hoeffding bound [Hoeffding 1994][6]; and **(2)** $O(t|m|)$: sampling's execution time derived from the required number t of state transitions and sampling's execution time $O(|m|)$ [PRISM 2017; Younes and Simmons 2006].

We emphasize that QPMC can fully leverage the performance advantages of QAE without additional costs for two main reasons: (1) it constructs circuits directly from high-level model descriptions, and (2) the number of gates in the resulting circuit scales proportionally to the number of operations Op in the model, similar to the sampling time of SMC.

Threats to validity. Although QPMC demonstrates a quadratic speedup over SMC in terms of the required number of samples, it is practical mainly for large models ($1 \ll |m|$) and when high accuracy ($\epsilon \ll 1$) is needed. **(1)** For small models, non-statistical approaches like sparse representations are sufficient for verification, making statistical methods like SMC and QPMC unnecessary. **(2)** For large models, statistical methods become essential due to the scalability limitations of non-statistical approaches (§8). For example, handling the LeaderSync case study with a state space of 2^{101} (Table 1) or larger models is challenging with non-statistical methods. In contrast, both QPMC and SMC scale well; managing a model with a state space of 2^{202} only doubles the required resources (quantum bits and gates), avoiding exponential resource increases. **(3)** While QPMC may be less efficient than SMC at low accuracy due to higher constant costs, its superior asymptotic complexity offers an advantage at high accuracy.

7.2 Benefits of Optimization

To answer **RQ2**, we compare the number of gates and qubits in the translated circuits of QPMC, both in complexities and in numbers for case studies.

Gate complexity. The optimization retains the number of gates at $O(t|m|)$. This arises from the fact that a batch recycling strategy involves adding $t_i - 1$ uncomputation circuits (C_{u}^{-1}) for each i-th batch, where t_i represents the number of transitions in the i-th batch. Since $t = \sum_i t_i$, the total number of uncomputation circuits is always less than t (*i.e.*, $\sum_i (t_i - 1) < t$), ensuring that there is no change in the gate complexity.

Qubit space complexity. The optimization reduces the number of qubits required from $O(t \log_2 |S| + t|\overrightarrow{md}| \log_2 h)$ to $O(\sqrt{t} \log_2 |S| + t|\overrightarrow{md}| \log_2 h)$, where $\overrightarrow{md}$ is the modules in m; and h is the maximum number of probabilistic updates among all commands in the model m. Here, $O(\log_2 |S|)$ and $O(|\overrightarrow{md}| \log_2 h)$ denote the number of qubits required to represent states and choices, respectively. Notably, the total number of state qubits is reduced through a recycling strategy.

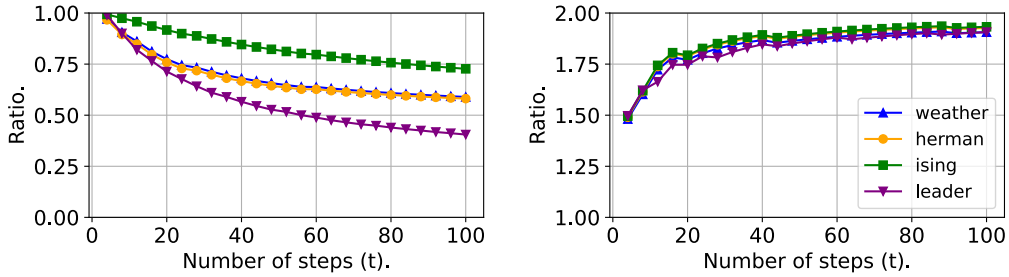
Case studies. We compare qubit and gate counts before and after optimization on PMC case studies. For this purpose, we measure the number of qubits and gates in the circuits generated by T and $\mathsf{T}_{\mathrm{opt}}$ (§6), considering the gates $X, R_y(\theta), Ct^{\vec{b}}(X), Ct^{\vec{b}}(R_y(\theta))$ (§2.2), where $|\vec{b}| \leq 5$, following Qiskit's level 2 gate optimization.

Table 1 shows the statistics of the benchmark models we used. We focus on four models with a large number of modules that are widely used in the evaluation of PMC tools [Holtzen et al. 2021; Kwiatkowska et al. 2011]: **(1) WeatherFactory**: An illustrative example that calculates factory strike probabilities from weather conditions; **(2) Herman**: A self-stabilizing algorithm of distributed systems; **(3) Ising**: A one-dimensional Ising model for spin systems; and **(4) LeaderSync**: A leader

[6]Heuristic methods like sequential testing [Wald 1992] and importance sampling [Srinivasan 2002] are excluded for their unpredictable sample sizes and no impact on asymptotic complexity.

Table 1. Model statistics. $\max|\vec{\lambda}_{\text{nu}}|$ and $\max|\vec{\lambda}_{\text{u}}|$ represent the maximum sizes of non-uniform and uniform distributions in the model's commands, respectively.

Model	$\lvert\overrightarrow{md}\rvert$	$\lvert S\rvert$	$\max\lvert\vec{\lambda}_{\text{nu}}\rvert$	$\max\lvert\vec{\lambda}_{\text{u}}\rvert$
WeatherFactory [Holtzen et al. 2021]	25	2^{26}	2	N/A
Herman [Herman 1990]	25	2^{25}	2	N/A
Ising [Sekizawa et al. 2009]	20	$\approx 2^{24}$	2	20
LeaderSync [Itai and Rodeh 1990]	14	$\approx 2^{101}$	N/A	4



(a) Decrease rate of qubits.

(b) Increase rate of gates.

Fig. 15. Rates of decrease for qubits and increase for gates after optimization.

election protocol for distributed systems. We measure qubit and gate counts for these models, ranging from $t = 4$ to $t = 100$ in increments of 4.

Fig. 15 shows the rate of change in qubit and gate counts after optimization. For all models, the decrease rate of qubits and the increase rate of gate become more significant with larger t, showing that optimization has a larger impact on both qubits and gates at higher t. For example, at $t = 100$, the WeatherFactory, Herman and Ising models exhibit a 35% decrease in qubits (from 5,887 to 3,469, 5,551 to 3,226 and 8,515 to 6,190, respectively) and an 93% increase in gates (from 285K to 543K, 185K to 357K and 1,153K to 2,229K). The Leader model shows a more substantial 59% decrease in qubits (15,412 to 6,247) and a 91% increase in gates (1,104K to 2,105K). This notable change is mainly due to the larger state space in the Leader model compared to other models, which means that decreasing the number of state qubits results in a more pronounced reduction in the total number of qubits required. While the reduction rate of qubits varies depending on the proportion of state qubits in the model, the increase rate of gates is always < 2.

We emphasize that the benefits of the decrease in qubits outweigh the costs of the increase in gates. Compared to gates, scaling up qubits is a much more challenging task in quantum computing [Gyongyosi and Imre 2019; Kielpinski et al. 2002; Ladd et al. 2010; Monroe et al. 2014]. As such, even in the most extreme case where the gate count almost doubles, the reduction of qubits by more than half shows that the optimization can increase the scalability of QPMC.

Threats to validity. Although the optimization significantly reduces the number of qubits, the translated circuit still may not be operated with current early-stage quantum computers due to inherent noise and limited qubit scales, typically consisting of only a few hundred physical qubits.

However, IBM, a company engaged in quantum computing, aims to develop systems with thousands of logical qubits by 2030+ [IBM 2024], potentially enabling QPMC to address realistic PMC problems in the near future. With such expectations, various quantum algorithms are being potentially embraced [Dalzell et al. 2023b]. For example, applications like lithium-ion battery technology, which requires 10^4 to 10^5 logical qubits [Kim et al. 2022], and finance portfolio optimization, which may necessitate 10^7 or more logical qubits [Dalzell et al. 2023a], are proposed.

8 Related and Future Work

Classical approaches to PMC. Various approaches for PMC can be categorized into two groups: non-statistical and statistical methods (§2.1).

On the one hand, several non-statistical methods have attempted to tackle the state explosion problem with simplification techniques, but they face fundamental scalability limitations. Symbolic representation methods compress state spaces using decision diagrams (DDs) such as multi-terminal binary DDs or multi-valued DDs [Baier et al. 1997], leveraging structural symmetries for compact representation [Hensel et al. 2021; Holtzen et al. 2021]. Abstraction strategies like bisimulation and game-based methods reduce model size by removing details from concrete models that are not relevant to the property of interest [Dehnert et al. 2012; Hahn et al. 2010; Katoen 2016; Kattenbelt et al. 2010]. For example, bisimulation identifies and merges states that are indistinguishable from each other [Katoen et al. 2007]. Model reduction techniques [Kamaleson 2018; Kwiatkowska et al. 2006] simplify models by removing redundant states or less significant states, preserving critical probabilities and behaviors while reducing complexity. In the domain of Markov Decision Processes (MDPs), CompMDP [Watanabe et al. 2023] integrates string diagram techniques for multi-module analysis, and other study [Junges and Spaan 2022] focus on utilizing the critical components to accelerate abstraction-refinement processes. However, these approaches still struggle to handle complex systems due to their reliance on exploring the entire state space.

On the other hand, several statistical methods have been developed to mitigate the runtime explosion in simulating rare events for SMC, but they do not reduce the asymptotic complexity of sampling. They typically use heuristics to identify and focus on crucial parts of the model's search space to decrease computational costs. For instance, Modes [Budde et al. 2018, 2020] employs importance splitting and sampling techniques [Srinivasan 2002], using guidance to steer the simulation toward the goal state. However, these approaches still require the user to define the importance function or select the sampling strategy, which can be challenging for non-experts lacking prior knowledge of the model.

In contrast to these classical approaches, QPMC offers a quantum solution to PMC that addresses the limitations of both non-statistical and statistical methods. **(1)** Compared to the non-statistical methods, QPMC efficiently handles large-scale systems by superposing states within a logarithmic-scale number of qubits (§7.2). **(2)** Compared to the statistical methods, QPMC improves asymptotic complexity over SMC (§7.1) without the need for prior model knowledge by systematically translating the model into quantum circuits (§3).

Quantum algorithms. Including QPMC, various quantum algorithms offer theoretical speedup over classical counterparts. Quantum simulation [Childs and Wiebe 2012; Low and Chuang 2017] and optimization [Peruzzo et al. 2014] have the potential to handle large-scale tasks that classical computers simply cannot, such as molecular modeling and logistics optimization. The QAE algorithm [Brassard et al. 2002], which QPMC utilizes, has been applied in financial risk analysis [Egger et al. 2020; Rebentrost et al. 2018], a field challenging due to its high complexity and vast datasets.

Similar to QPMC, IBM's recent study [Layden et al. 2023] utilizes quantum computers to replicate the probabilistic transitions of the Markov chain and shows its practical advantages. The

authors accelerated Monte Carlo simulations for the Boltzmann distribution of the Ising model by implementing quantum circuits on current noisy quantum computers. However, this specialized algorithm targets only the Boltzmann distribution of the Ising model, limiting their general applicability. In contrast, QPMC is general, suitable for DTMCs in PMC languages (§3). We translated the existing Ising model described in PMC language [Sekizawa et al. 2009] into quantum circuits capable of systematically analyzing its properties (§7) modulo a primitive gate and quantum noise (see below).

Future work. While QPMC is general, it does not support all primitives required to subsume existing quantum algorithms, like the R_{ZZ} gate used in the aforementioned work [Layden et al. 2023]. Such R_{ZZ} gates are essential for introducing phase shifts on qubits and achieving more efficient quantum circuits. Also, the current implementation of QPMC does not consider the effects of noise on quantum circuits, which can pose challenges in achieving the expected speedup in the near future. As future work, we will support the gate to fully represent the existing quantum algorithms in PMC language, and investigate the impact of noise on quantum circuits to highlight the practical advantages of QPMC.

In addition, we plan to explore the following areas: **(1)** *MDP support*: We aim to extend beyond bounded reachability problems in DTMCs to a wider class PMC problems, including MDPs. **(2)** *Runtime analysis*: We aim to estimate the execution times of QPMC on both noisy and noiseless quantum hardware. For example, we plan to estimate the runtime by assuming the reasonable fault-tolerant quantum hardware likes IBM [Egger et al. 2020]. **(3)** *New quantum programming abstraction*: We aim to design a high-level quantum programming language by extending QPMC's approach to probabilistic programming languages. As QPMC serves as an initial bridge between probabilistic models and quantum circuits, our goal is to naturally extend this connection to encompass a more general probabilistic programming language. This allows for the development of quantum algorithms as programs based on probabilistic models, providing a more user-friendly programming environment compared to existing low-level, gate-centric quantum programming languages.

Acknowledgments

We thank the CAV'24 and OOPSLA'24 reviewers for valuable feedback and suggestions for improvements. This work was supported by: **(1)** National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) partly under No. RS-2024-00347786 (Practical Deductive Verification of Logic Chips, 60%) and No. 2021R1A5A1021944 (10%); and **(2)** Institute for Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) partly under the Information Technology Research Center (ITRC) support program (No. IITP-2024-2020-0-01795, Development of Dependable and Highly Usable Big Data Platform, and Analysis and Prediction Services Technology in Edge Clouds, 10%), the Graduate School of Artificial Intelligence Semiconductor (No. IITP-2024-RS-2023-00256472, 10%), and the SW STAR LAB (No. 2020-0-01337, Research on Highly-Practical Automated Software Repair, 10%).

Data-Availability Statement

The development and appendix for this paper can be found in [Jeon et al. 2024].

References

Christopher Abraham et al. 2019. Qiskit: An open-source framework for quantum computing. *arXiv preprint arXiv:1909.05820* (2019).

Gul Agha and Karl Palmskog. 2018. A survey of statistical model checking. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 28, 1 (2018), 1–39.

Christel Baier, Edmund M Clarke, Vasiliki Hartonas-Garmhausen, Marta Kwiatkowska, and Mark Ryan. 1997. Symbolic model checking for probabilistic processes. In *Automata, Languages and Programming: 24th International Colloquium, ICALP'97 Bologna, Italy, July 7–11, 1997 Proceedings 24*. Springer, 430–440.

Christel Baier, Luca de Alfaro, Vojtěch Forejt, and Marta Kwiatkowska. 2018. Model checking probabilistic systems. *Handbook of Model Checking* (2018), 963–999.

Charles H Bennett. 1973. Logical reversibility of computation. *IBM journal of Research and Development* 17, 6 (1973), 525–532.

Benjamin Bichsel, Maximilian Baader, Timon Gehr, and Martin Vechev. 2020. 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*. 286–300.

Jonathan Bogdoll, Luis María Ferrer Fioriti, Arnd Hartmanns, and Holger Hermanns. 2011. Partial order methods for statistical model checking and simulation. In *International Conference on Formal Methods for Open Object-Based Distributed Systems*. Springer, 59–74.

Ali Bolhassani and Majid Haghparast. 2016. Optimised reversible divider circuit. *International Journal of Innovative Computing and Applications* 7, 1 (2016), 13–33.

Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. 2002. Quantum amplitude amplification and estimation. *Contemp. Math.* 305 (2002), 53–74.

Carlos E Budde, Christian Dehnert, Ernst Moritz Hahn, Arnd Hartmanns, Sebastian Junges, and Andrea Turrini. 2017. JANI: quantitative model and tool interaction. In *Tools and Algorithms for the Construction and Analysis of Systems: 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part II 23*. Springer, 151–168.

Carlos E Budde, Pedro R D'Argenio, Arnd Hartmanns, and Sean Sedwards. 2018. A statistical model checker for nondeterminism and rare events. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 340–358.

Carlos E Budde, Pedro R D'Argenio, Arnd Hartmanns, and Sean Sedwards. 2020. An efficient statistical model checker for nondeterminism and rare events. *International journal on software tools for technology transfer* 22, 6 (2020), 759–780.

Andrew M Childs and Nathan Wiebe. 2012. Hamiltonian simulation using linear combinations of unitary operations. *arXiv preprint arXiv:1202.5822* (2012).

Don Coppersmith. 2002. An approximate Fourier transform useful in quantum factoring. *arXiv preprint quant-ph/0201067* (2002).

Alexander M Dalzell, B David Clader, Grant Salton, Mario Berta, Cedric Yen-Yu Lin, David A Bader, Nikitas Stamatopoulos, Martin JA Schuetz, Fernando GSL Brandão, Helmut G Katzgraber, et al. 2023a. End-to-end resource analysis for quantum interior-point methods and portfolio optimization. *PRX Quantum* 4, 4 (2023), 040325.

Alexander M Dalzell, Sam McArdle, Mario Berta, Przemyslaw Bienias, Chi-Fang Chen, András Gilyén, Connor T Hann, Michael J Kastoryano, Emil T Khabiboulline, Aleksander Kubica, et al. 2023b. Quantum algorithms: A survey of applications and end-to-end complexities. *arXiv preprint arXiv:2310.03011* (2023).

Christian Dehnert, Daniel Gebler, Michele Volpato, and David N Jansen. 2012. On abstraction of probabilistic systems. *International Autumn School on Rigorous Dependability Analysis Using Model Checking Techniques for Stochastic Systems* (2012), 87–116.

Daniel J Egger, Ricardo García Gutiérrez, Jordi Cahué Mestre, and Stefan Woerner. 2020. Credit risk analysis using quantum computers. *IEEE transactions on computers* 70, 12 (2020), 2136–2145.

Antonio Filieri, Carlo Ghezzi, and Giordano Tamburrelli. 2011. Run-time efficient probabilistic model checking. In *Proceedings of the 33rd international conference on software engineering*. 341–350.

Nick Giannarakis, Alexandra Silva, and David Walker. 2021. ProbNV: probabilistic verification of network control planes. *Proceedings of the ACM on Programming Languages* 5, ICFP (2021), 1–30.

András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. 2019. Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. 193–204.

Lov K Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 212–219.

Laszlo Gyongyosi and Sandor Imre. 2019. A survey on quantum computing technology. *Computer Science Review* 31 (2019), 51–71.

Majid Haghparast and Ali Bolhassani. 2016. Optimization approaches for designing quantum reversible arithmetic logic unit. *International Journal of Theoretical Physics* 55 (2016), 1423–1437.

Ernst Moritz Hahn, Arnd Hartmanns, Christian Hensel, Michaela Klauck, Joachim Klein, Jan Křetínskỳ, David Parker, Tim Quatmann, Enno Ruijters, and Marcel Steinmetz. 2019. The 2019 Comparison of Tools for the Analysis of Quantitative Formal Models: (QComp 2019 Competition Report). In *International Conference on Tools and Algorithms for the Construction*

and Analysis of Systems. Springer, 69–92.

Ernst Moritz Hahn, Holger Hermanns, Björn Wachter, and Lijun Zhang. 2010. PASS: Abstraction refinement for infinite probabilistic models. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 353–357.

Hans Hansson and Bengt Jonsson. 1994. A logic for reasoning about time and reliability. *Formal aspects of computing* 6 (1994), 512–535.

Arnd Hartmanns and Holger Hermanns. 2014. The Modest Toolset: An integrated environment for quantitative modelling and verification. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 593–598.

Christian Hensel, Sebastian Junges, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. 2021. The probabilistic model checker Storm. *International Journal on Software Tools for Technology Transfer* (2021), 1–22.

Thomas Hérault, Richard Lassaigne, Frédéric Magniette, and Sylvain Peyronnet. 2004. Approximate probabilistic model checking. In *Verification, Model Checking, and Abstract Interpretation: 5th International Conference, VMCAI 2004 Venice, Italy, January 11-13, 2004 Proceedings 5*. Springer, 73–84.

Ted Herman. 1990. Probabilistic self-stabilization. *Inform. Process. Lett.* 35, 2 (1990), 63–67.

Wassily Hoeffding. 1994. Probability inequalities for sums of bounded random variables. *The collected works of Wassily Hoeffding* (1994), 409–426.

Torsten Hoefler, Thomas Häner, and Matthias Troyer. 2023. Disentangling hype from practicality: on realistically achieving quantum advantage. *Commun. ACM* 66, 5 (2023), 82–87.

Steven Holtzen, Sebastian Junges, Marcell Vazquez-Chanlatte, Todd Millstein, Sanjit A Seshia, and Guy Van den Broeck. 2021. Model checking finite-horizon Markov chains with probabilistic inference. In *International Conference on Computer Aided Verification*. Springer, 577–601.

IBM. 2024. IBM Quantum Roadmap. https://www.ibm.com/roadmaps/quantum/

Alon Itai and Michael Rodeh. 1990. Symmetry breaking in distributed networks. *Information and Computation* 88, 1 (1990), 60–87.

Nils Jansen, Bettina Könighofer, JSL Junges, AC Serban, and Roderick Bloem. 2020. Safe reinforcement learning using probabilistic shields. (2020).

HV Jayashree, Himanshu Thapliyal, Hamid R Arabnia, and Vinod Kumar Agrawal. 2016. Ancilla-input and garbage-output optimized design of a reversible quantum integer multiplier. *The Journal of Supercomputing* 72 (2016), 1477–1493.

Seungmin Jeon, Kyeongmin Cho, Changu Kang, Janggun Lee, Hakjoo Oh, and Jeehoon Kang. 2024. Artifact for "Quantum Probabilistic Model Checking for Time-Bounded Properties". https://doi.org/10.5281/zenodo.13377564

Sebastian Junges and Matthijs TJ Spaan. 2022. Abstraction-refinement for hierarchical probabilistic models. In *International Conference on Computer Aided Verification*. Springer, 102–123.

Nishanthan Kamaleson. 2018. *Model reduction techniques for probabilistic verification of Markov chains*. Ph. D. Dissertation. University of Birmingham.

Joost-Pieter Katoen. 2016. The probabilistic model checking landscape. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*. 31–45.

Joost-Pieter Katoen, Tim Kemna, Ivan Zapreev, and David N Jansen. 2007. Bisimulation minimisation mostly speeds up probabilistic model checking. In *Tools and Algorithms for the Construction and Analysis of Systems: 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24-April 1, 2007. Proceedings 13*. Springer, 87–101.

Mark Kattenbelt, Marta Kwiatkowska, Gethin Norman, and David Parker. 2010. A game-based abstraction-refinement framework for Markov decision processes. *Formal Methods in System Design* 36 (2010), 246–280.

David Kielpinski, Chris Monroe, and David J Wineland. 2002. Architecture for a large-scale ion-trap quantum computer. *Nature* 417, 6890 (2002), 709–711.

Isaac H Kim, Ye-Hua Liu, Sam Pallister, William Pol, Sam Roberts, and Eunseok Lee. 2022. Fault-tolerant resource estimate for quantum chemical simulations: Case study on Li-ion battery electrolyte molecules. *Physical Review Research* 4, 2 (2022), 023019.

A Yu Kitaev. 1995. Quantum measurements and the Abelian stabilizer problem. *arXiv preprint quant-ph/9511026* (1995).

Marta Kwiatkowska, Gethin Norman, and David Parker. 2006. Symmetry reduction for probabilistic model checking. In *International Conference on Computer Aided Verification*. Springer, 234–248.

Marta Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings 23*. Springer, 585–591.

Marta Kwiatkowska, Gethin Norman, and David Parker. 2012. The PRISM benchmark suite. In *9th International Conference on Quantitative Evaluation of SysTems*. IEEE CS press, 203–204.

Marta Kwiatkowska, Gethin Norman, and David Parker. 2022. Probabilistic model checking and autonomy. *Annual Review of Control, Robotics, and Autonomous Systems* 5 (2022), 385–410.

Marta Kwiatkowska, Gethin Norman, and Jeremy Sproston. 2003. Probabilistic model checking of deadline properties in the IEEE 1394 FireWire root contention protocol. *Formal Aspects of Computing* 14 (2003), 295–318.

Thaddeus D Ladd, Fedor Jelezko, Raymond Laflamme, Yasunobu Nakamura, Christopher Monroe, and Jeremy Lloyd O'Brien. 2010. Quantum computers. *nature* 464, 7285 (2010), 45–53.

Agnes Lagnoux. 2006. Rare event simulation. *Probability in the Engineering and Informational Sciences* 20, 1 (2006), 45–66.

Richard Lassaigne and Sylvain Peyronnet. 2002. Approximate verification of probabilistic systems. In *Joint International Workshop von Process Algebra and Probabilistic Methods, Performance Modeling and Verification*. Springer, 213–214.

David Layden, Guglielmo Mazzola, Ryan V Mishmash, Mario Motta, Pawel Wocjan, Jin-Sung Kim, and Sarah Sheldon. 2023. Quantum-enhanced markov chain Monte Carlo. *Nature* 619, 7969 (2023), 282–287.

Guang Hao Low and Isaac L. Chuang. 2017. Optimal Hamiltonian Simulation by Quantum Signal Processing. *Phys. Rev. Lett.* 118 (Jan 2017), 010501. Issue 1. https://doi.org/10.1103/PhysRevLett.118.010501

Christopher Monroe, Robert Raussendorf, Alex Ruthven, Kenneth R Brown, Peter Maunz, L-M Duan, and Jungsang Kim. 2014. Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects. *Physical Review A* 89, 2 (2014), 022317.

Ashley Montanaro. 2015. Quantum speedup of Monte Carlo methods. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 471, 2181 (2015), 20150301.

Michael A Nielsen and Isaac L Chuang. 2010. *Quantum computation and quantum information*. Cambridge university press.

Anouk Paradis, Benjamin Bichsel, Samuel Steffen, and Martin Vechev. 2021. Unqomp: synthesizing uncomputation in Quantum circuits. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 222–236.

Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'brien. 2014. A variational eigenvalue solver on a photonic quantum processor. *Nature communications* 5, 1 (2014), 4213.

PRISM. 2010. The PRISM Language - Semantics. http://www.prismmodelchecker.org/doc/semantics.pdf

PRISM. 2017. Statistical Model Checking. https://www.prismmodelchecker.org/manual/RunningPRISM/StatisticalModelChecking

Patrick Rebentrost, Brajesh Gupt, and Thomas R. Bromley. 2018. Quantum computational finance: Monte Carlo pricing of financial derivatives. *Phys. Rev. A* 98 (Aug 2018), 022321. Issue 2. https://doi.org/10.1103/PhysRevA.98.022321

Gerardo Rubino, Bruno Tuffin, et al. 2009. *Rare event simulation using Monte Carlo methods*. Vol. 73. Wiley Online Library.

Toshifusa Sekizawa, Tatsuhiro Tsuchiya, Koichi Takahashi, and Tohru Kikuno. 2009. Probabilistic model checking of the one-dimensional Ising model. *IEICE transactions on information and systems* 92, 5 (2009), 1003–1011.

Vivek V Shende, Stephen S Bullock, and Igor L Markov. 2005. Synthesis of quantum logic circuits. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*. 272–275.

Alok Shukla and Prakash Vedula. 2024. An efficient quantum algorithm for preparation of uniform quantum superposition states. *Quantum Information Processing* 23, 2 (2024), 38.

Steffen Smolka, Praveen Kumar, David M Kahn, Nate Foster, Justin Hsu, Dexter Kozen, and Alexandra Silva. 2019. Scalable verification of probabilistic networks. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 190–203.

Rajan Srinivasan. 2002. *Importance sampling: Applications in communications and detection*. Springer Science & Business Media.

Nikitas Stamatopoulos, Daniel J Egger, Yue Sun, Christa Zoufal, Raban Iten, Ning Shen, and Stefan Woerner. 2020. Option pricing using quantum computers. *Quantum* 4 (2020), 291.

Guoxin Su, David S Rosenblum, and Giordano Tamburrelli. 2016. Reliability of run-time quality-of-service evaluation using parametric model checking. In *Proceedings of the 38th International Conference on Software Engineering*. 73–84.

Antti Valmari. 1996. The state explosion problem. In *Advanced Course on Petri Nets*. Springer, 429–528.

Vlatko Vedral, Adriano Barenco, and Artur Ekert. 1996. Quantum networks for elementary arithmetic operations. *Physical Review A* 54, 1 (1996), 147.

Abraham Wald. 1992. Sequential tests of statistical hypotheses. In *Breakthroughs in statistics: Foundations and basic theory*. Springer, 256–298.

Kazuki Watanabe, Clovis Eberhart, Kazuyuki Asada, and Ichiro Hasuo. 2023. Compositional Probabilistic Model Checking with String Diagrams of MDPs. In *35th International Conference on Computer Aided Verification (CAV 2023)*.

Haiying Xia, Haisheng Li, Han Zhang, Yan Liang, and Jing Xin. 2018. An efficient design of reversible multi-bit quantum comparator via only a single ancillary bit. *International Journal of Theoretical Physics* 57 (2018), 3727–3744.

Håkan LS Younes, Marta Kwiatkowska, Gethin Norman, and David Parker. 2006. Numerical vs. statistical probabilistic model checking. *International Journal on Software Tools for Technology Transfer* 8 (2006), 216–228.

Håkan LS Younes and Reid G Simmons. 2002. Probabilistic verification of discrete event systems using acceptance sampling. In *Computer Aided Verification: 14th International Conference, CAV 2002 Copenhagen, Denmark, July 27–31, 2002 Proceedings 14*. Springer, 223–235.

Håkan LS Younes and Reid G Simmons. 2006. Statistical probabilistic model checking with a focus on time-bounded properties. *Information and Computation* 204, 9 (2006), 1368–1409.

Received 2024-04-06; accepted 2024-08-18