Abstraqt: Analysis of Quantum Circuits via Abstract Stabilizer Simulation

Benjamin Bichsel, Anouk Paradis, Maximilian Baader, and Martin Vechev

ETH Zurich, Switzerland

Stabilizer simulation can efficiently simulate an important class of quantum circuits consisting exclusively of Clifford gates. However, all existing extensions of this simulation to arbitrary quantum circuits including non-Clifford gates suffer from an exponential runtime.

To address this challenge, we present a novel approach for efficient stabilizer simulation on arbitrary quantum circuits, at the cost of lost precision. Our key idea is to compress an exponential sum representation of the quantum state into a single *abstract* summand covering (at least) all occurring summands. This allows us to introduce an *abstract stabilizer simulator* that efficiently manipulates abstract summands by *over-approximating* the effect of circuit operations including Clifford gates, non-Clifford gates, and (internal) measurements.

We implemented our abstract simulator in a tool called Abstraqt and experimentally demonstrate that Abstraqt can establish circuit properties intractable for existing techniques.

1 Introduction

Stabilizer simulation [1] is a promising technique for efficient classical simulation of quantum circuits consisting exclusively of *Clifford* gates. Unfortunately, generalizing stabilizer simulation to arbitrary circuits including non-Clifford gates requires exponential time [2, 3, 4, 5, 6, 7]. Specifically, the first such generalization by Aaronson and Gottesman [2, §VII-C] tracks the quantum state ρ at any point in the quantum circuit as a sum whose number of summands m grows exponentially with the number of non-Clifford gates:

$$\rho = \sum_{i=1}^{m} c_i P_i \prod_{j=1}^{n} \frac{\mathbb{I}+(-1)^{b_{ij}} Q_j}{2}. \tag{1}$$

Here, while c_i, P_i, b_{ij}, and Q_j can be represented efficiently (see §2), the overall representation is inefficient due to exponentially large m.

Abstraction. The key idea of Abstraqt is to avoid tracking the exact state ρ of a quantum system and instead only track key aspects of ρ. To this end, we rely on the established framework of abstract interpretation [8, 9], which is traditionally used to analyze classical programs [10, 11] or neural networks [12] by describing sets of possible states without explicitly enumerating all of them. Here, we use abstract interpretation to describe the set of quantum states that could occur at a specific point during execution of a circuit, by *over-approximating* the summands that could occur in any of those quantum states ρ.

Merging Summands. This allows us to curb the exponential blow-up of stabilizer simulation by merging multiple summands in Eq. (1) into an abstract single summand which over-approximates all summands, at the cost of lost precision. The key technical challenge addressed by our work is designing a suitable *abstract domain* to describe sets of summands, accompanied by the corresponding *abstract transformers* to over-approximate the actions performed by the original exponential stabilizer simulation on individual summands.

As a result, our approach is both efficient and exact on Clifford circuits, as these circuits never require merging summands. On non-Clifford circuits, merging summands trades precision for efficiency. Moreover, our approach naturally allows us to merge the possible outcomes of a measurement into a single abstract state, preventing an exponential path explosion when simulating multiple internal measurements.

Main Contributions. Our main contributions are:

- An abstract domain (§4) to over-approximate a quantum state represented by Eq. (1).
- Abstract transformers (§5) to simulate quantum circuits, including gate applications and measurements.
- An efficient implementation[1] of our approach in a tool called Abstraqt (§6), together with an evaluation showing that Abstraqt can establish circuit properties that are intractable for existing tools (§7).

Results. Overall, we find that Abstraqt is useful in scenarios where a full simulation of a given circuit is intractable, but establishing specific properties of the considered circuit is desirable.

For example, in our evaluation (§7), we demonstrate that Abstraqt can establish that a circuit ultimately restores some qubits to state $|0\rangle$. As precisely simulating the entire circuit is intractable, Abstraqt is typically the only existing tool able to establish this fact on 12 benchmarked circuits. In contrast, existing tools typically yield incorrect results, throw errors, run out of memory, time out, or are too imprecise to establish the resulting state is $|0\rangle$.

Outlook. Abstraqt trades precision for efficiency by abstracting the stabilizer simulation from Eq. (1), therefore allowing to establish properties of quantum circuit outputs when full simulation is intractable. Such results may be useful for tasks like (i) establishing that an internal circuit state allows for specific optimizations, (ii) debugging quantum computers by establishing invariants that can be checked at runtime, and more generally (iii) static analysis of quantum circuits, or (iv) verification of the correctness of quantum circuits.

Further, as discussed in §7.4, Abstraqt abstracts the very first stabilizer simulation generalized to non-Clifford gates by Aaronson and Gottesman [2, §VII-C]. We believe that our encouraging results pave the way to introduce analogous abstraction to various follow-up works which improve upon this simulation [4, 5, 6, 7]. As these more recent works scale better than [2, §VII-C], we expect that a successful application of abstract interpretation to them will yield even more favorable trade-offs between precision and efficiency.

2 Background

We first introduce the necessary mathematical concepts.

Basic Notation. We use $\mathbb{Z}_n := \mathbb{Z}/(n\mathbb{Z})$, define $\mathbb{B} := \mathbb{Z}_2$, and write 2^S for the power set of the set S.

We represent a pure n-qubit quantum state $\psi \in \mathbb{C}^{2^n}$ as a *density matrix* $\rho \in \mathbb{C}^{2^n \times 2^n}$, defined as $\rho = \psi\psi^\dagger$, where $\psi^\dagger$ denotes the conjugate transpose of ψ. For a mixed state, i.e., a distribution over pure states ψ_i with probability p_i, the corresponding density matrix is $\rho = \sum_i p_i \psi_i \psi_i^\dagger$. Because both ψ and ρ store exponentially many values, they cannot be represented explicitly for large n. We denote the embedding of a k-qubit gate $U \in \mathcal{U}(2^k)$ as an n-qubit gate by $U_{(i)} := \mathbb{I}_{2^i} \otimes U \otimes \mathbb{I}_{2^{n-i-k}}$, where $\mathbb{I}_l$ denotes the $l \times l$ identity matrix.

[1] Our implementation is available at https://github.com/eth-sri/abstraqt.

Stabilizer Simulation. The key idea of stabilizer simulation [1, 2] is representing quantum states $\rho = \psi\psi^\dagger$ implicitly, by *stabilizers* Q which stabilize the state ψ, that is $Q\psi = \psi$. As shown in [2], appropriately selecting n stabilizers Q_j then specifies a unique n-qubit state $\rho = \prod_{j=1}^{n} \frac{\mathbb{I}+Q_j}{2}$.

In stabilizer simulation, all Q_j are *Pauli elements* from $\mathcal{P}_n$ of the form $\mathrm{i}^v \cdot P^{(0)} \otimes \cdots \otimes P^{(n-1)}$, where $P^{(j)} \in \{X, Y, Z, \mathbb{I}_2\}$ and $\mathrm{v} \in \mathbb{Z}_4$. This directly implies that all stabilizers Q_i for the same state ψ commute, that is $Q_iQ_j = Q_jQ_i$, as elements from the Pauli group $\mathcal{P}_n$ either commute or anti-commute. These elements can be represented efficiently in memory by storing v and $P^{(0)}, \ldots, P^{(n-1)}$. In App. B, we list states stabilized by Pauli matrices (Tab. 7) and the results of multiplying Pauli matrices (Tab. 8). Further, in this work we use the functions *bare* $\mathfrak{b}\colon \mathcal{P}_n \to \mathcal{P}_n$ and *prefactor* $\mathfrak{f}\colon \mathcal{P}_n \to \mathbb{Z}_4$ which extract the Pauli matrices without the prefactor and the prefactor, respectively:

$$\mathfrak{f}(\mathrm{i}^v P^{(0)} \otimes \cdots \otimes P^{(n-1)}) = v, \tag{2}$$

$$\mathfrak{b}(\mathrm{i}^v P^{(0)} \otimes \cdots \otimes P^{(n-1)}) = P^{(0)} \otimes \cdots \otimes P^{(n-1)}. \tag{3}$$

Applying gate U to state ρ can be reduced to conjugating the stabilizers Q_j with U:

$$U\rho U^\dagger = U\Big(\prod_{j=1}^{n} \tfrac{\mathbb{I}+Q_j}{2}\Big)U^\dagger \overset{[13,\ \text{Sec. } 10.5]}{=} \prod_{j=1}^{n} \tfrac{\mathbb{I}+UQ_jU^\dagger}{2}. \tag{4}$$

While Eq. (4) holds for any gate U, stabilizer simulation can only exploit it if $UQ_jU^\dagger \in \mathcal{P}_n$. *Clifford gates* such as S, H, $CNOT$, $\mathbb{I}$, X, Y, and Z satisfy this for any $Q_j \in \mathcal{P}_n$.

To also support the application of non-Clifford gates such as T gates, we follow [2, §VII.C] and represent ρ more generally as

$$\rho = \sum_{i=1}^{m} c_i P_i \prod_{j=1}^{n} \tfrac{\mathbb{I}+(-1)^{b_{ij}}Q_j}{2},$$

for $c_i \in \mathbb{C}$, $P_i \in \mathcal{P}_n$, $b_{ij} \in \mathbb{B}$, and $Q_j \in \mathcal{P}_n$. Here, applying U to ρ amounts to replacing P_i by $UP_iU^\dagger$ and Q_j by $UQ_jU^\dagger$, which we can exploit if both $UP_iU^\dagger$ and $UQ_jU^\dagger$ lie in $\mathcal{P}_n$.

Otherwise, we decompose[2] U to the sum $\sum_{p=1}^{K} d_p R_p$, where $d_p \in \mathbb{C}$ and $R_p \in \mathfrak{b}(\mathcal{P}_n)$ are bare Pauli elements, which have a prefactor of $\mathrm{i}^0 = 1$. Then,

$$U\rho U^\dagger = \Big(\sum_{p=1}^{K} d_p R_p\Big)\Big(\sum_{i=1}^{m} c_i P_i \prod_{j=1}^{n} \tfrac{\mathbb{I}+(-1)^{b_{ij}}Q_j}{2}\Big)\Big(\sum_{q=1}^{K} d_q R_q\Big)^\dagger \tag{5}$$

$$\overset{[2,\ \S\text{VII.C}]}{=} \sum_{p=1}^{K}\sum_{i=1}^{m}\sum_{q=1}^{K} c_{piq} P_{piq} \prod_{j=1}^{n} \tfrac{\mathbb{I}+(-1)^{b_{ijq}}Q_j}{2}, \tag{6}$$

for $c_{piq} = d_p c_i d_q^* \in \mathbb{C}$, $P_{piq} = R_p P_i R_q \in \mathcal{P}_n$, and $b_{ijq} = b_{ij} + Q_j \diamond R_q \in \mathbb{B}$. Here, d_q^* denotes the complex conjugate of d_q, $+$ denotes addition modulo 2, and $Q_j \diamond R_q$ is the commutator defined as 0 if Q_j and R_q commute and 1 otherwise. Note that $\cdot \diamond \cdot\colon \mathcal{P}_n \times \mathcal{P}_n \to \mathbb{B}$ has the highest precedence.

Overall, the decomposition of a k-qubit non-Clifford gate results in at most $K = 4^k$ summands, thus blowing up the number of summands in our representation by at most $4^k \cdot 4^k = 16^k$. In practice, the blow-up is typically smaller, e.g., decomposing a T gate only requires 2 summands, while decomposing a $CCNOT$ gate requires 8 summands.

Measurement. Measuring in bare Pauli basis $P \in \mathfrak{b}(\mathcal{P}_n)$ yields one of two possible quantum states. They can be computed by applying the two *projections* $P_+ := \frac{\mathbb{I}+P}{2}$ and $P_- = \frac{\mathbb{I}-P}{2}$, resulting in states $\rho_+ = P_+\rho P_+$ and $\rho_- = P_-\rho P_-$, respectively. For example, collapsing the i^{th} qubit to $|0\rangle$ or $|1\rangle$ corresponds to measuring in Pauli basis $Z_{(i)}$. The probability of outcome ρ_+ is $\mathrm{tr}\,(\rho_+)$, and analogously for ρ_-. Note that we avoid renormalization for simplicity. We discuss in §5 how measurements are performed in stabilizer simulation [2, Sec. VII.C].

[2]This decomposition always exists and is unique, as bare Pauli elements span (more than) $\mathcal{U}(2^n)$.

Table 1: Transformers for the interval abstraction.

Function	Abstract Transformer	Efficient Closed form
$+$	$[l_1, u_1] +^\sharp [l_2, u_2] = [l', u']$	$l' = l_1 + l_2$ and $u' = u_1 + u_2$
$\cdot$	$[l_1, u_1] \cdot^\sharp [l_2, u_2] = [l', u']$	$l' = \min(l_1 l_2, l_1 u_2, u_1 l_2, u_1 u_2)$, u' defined analogously with max
exp	$\exp^\sharp([l, u]) = [l', u']$	$l' = \exp(l)$ and $u' = \exp(u)$
cos	$\cos^\sharp([l, u]) = [l', u']$	exists, several case distinctions necessary
$\cup$	$[l_1, u_1] \sqcup [l_2, u_2] = [l', u']$	$l' = \min(l_1, l_2)$ and $u' = \max(u_1, u_2)$

Abstract Interpretation. Abstract interpretation [8] is a framework for formalizing approximate but sound calculation. An *abstraction* consists of ordered sets $(2^{\mathcal{X}}, \subseteq)$ and $(\boldsymbol{\mathcal{X}}, \leq)$, where $\mathcal{X}$ and $\boldsymbol{\mathcal{X}}$ are called *concrete set* and *abstract set* respectively together with a *concretization function* $\gamma\colon \boldsymbol{\mathcal{X}} \to 2^{\mathcal{X}}$ which indicates which concrete elements $x = \gamma(\boldsymbol{x}) \subseteq \mathcal{X}$ are represented by the abstract element $\boldsymbol{x}$. Additionally, $\bot \in \boldsymbol{\mathcal{X}}$ refers to $\emptyset = \gamma(\bot) \subseteq \mathcal{X}$ and $\top \in \boldsymbol{\mathcal{X}}$ refers to $\mathcal{X} = \gamma(\top)$.

An abstract transformer $f^\sharp\colon \boldsymbol{\mathcal{X}} \to \boldsymbol{\mathcal{X}}$ of a function $f\colon \mathcal{X} \to \mathcal{X}$ satisfies $\gamma \circ f^\sharp(\boldsymbol{x}) \supseteq f \circ \gamma(\boldsymbol{x})$ for all $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}$, where f was lifted to operate on subsets of $\mathcal{X}$. This ensures that $f^\sharp$ (over-)approximates f, a property referred to as *soundness* of $f^\sharp$. Abstract transformers can analogously be defined for functions $f\colon \mathcal{X}^n \to \mathcal{X}$. Further, we introduce *join* $\sqcup\colon \boldsymbol{\mathcal{X}} \times \boldsymbol{\mathcal{X}} \to \boldsymbol{\mathcal{X}}$, satisfying $\gamma(\boldsymbol{x}) \cup \gamma(\boldsymbol{y}) \subseteq \gamma(\boldsymbol{x} \sqcup \boldsymbol{y})$. Throughout this work, we distinguish abstract objects $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}$ and concrete objects $x \in \mathcal{X}$ by stylizing them in bold or non-bold respectively.

As an example, a common abstraction is the interval abstraction with $\mathcal{X} = \mathbb{R}$. The abstract set is the set of intervals

$$\boldsymbol{\mathcal{X}} = \{(l, u) \mid l, u \in \mathbb{R} \cup \{\pm\infty\}\},$$

where $\boldsymbol{x} = (l, u)$ is a tuple. The concretization function $\gamma\colon \boldsymbol{\mathcal{X}} \to \mathcal{X}$ maps these tuples to sets:

$$\gamma(\boldsymbol{x}) = [l, u] = \{y \in \mathbb{R} \mid l \leq y \leq u\}.$$

Further, $\top = (-\infty, \infty)$ and $\bot = (l, u)$ for $l > u$. Common abstract transformers for the interval abstraction are shown in Tab. 1.

The transformers in Tab. 1 are *precise*, meaning that for $f\colon \mathbb{R} \to \mathbb{R}$, we have that $f^\sharp((l, u)) = (\min_{l \leq v \leq u} f(v), \max_{l \leq v \leq u} f(v))$ and analogously for $f\colon \mathbb{R}^n \to \mathbb{R}$. An abstract transformer for a composition of functions $f \circ g$ is the composition of the abstract transformers. Although this is sound, it is not necessarily precise: let $g\colon \mathbb{R} \to \mathbb{R}^2$ with $g(x) = \binom{x}{x}$ and $f\colon \mathbb{R}^2 \to \mathbb{R}$ with $f(x, y) = x \cdot y$, then $f \circ g(x) = x^2$, but $f^\sharp \circ g^\sharp((-2, 2)) = (-4, 4)$ whereas a precise transformer would map $(-2, 2)$ to $(0, 4)$.

Notational Convention. In slight abuse of notation, throughout this work we may write the concretization of abstract elements instead of the abstract element itself. For example, for $(0, 1) \in \boldsymbol{R}$, we write $[0, 1]$ defined as $\{v \in \mathbb{R} \mid 0 \leq v \leq 1\}$ to indicate that it represents an interval. Where clear from context, we omit $\sharp$ and write f for $f^\sharp$. For example, we write $[l_1, u_1] + [l_2, u_2]$ for $[l_1, u_1] +^\sharp [l_2, u_2]$.

3 Overview

In this section, we showcase Abstraqt by applying it to the example circuit in Fig. 1. Overall, Abstraqt proceeds analogously to [2, §VII-C], but operates on abstract summands representing many concrete summands.

Example Circuit. We first discuss the circuit in Fig. 1. Both qubits are initialized to $|0\rangle$. The circuit then applies a succession of gates. The abstract representation of the state after the application of each gate is shown in the gray boxes below the circuit. On the final state, the circuit collapses the upper qubit to $|-\rangle$ by applying the projection $M_- = \frac{\mathbb{1} - X_{(0)}}{2}$. Precise circuit simulation shows that the probability of obtaining $|-\rangle$ is 0, in this case. In the following, we demonstrate how Abstraqt computes an over-approximation of this probability.

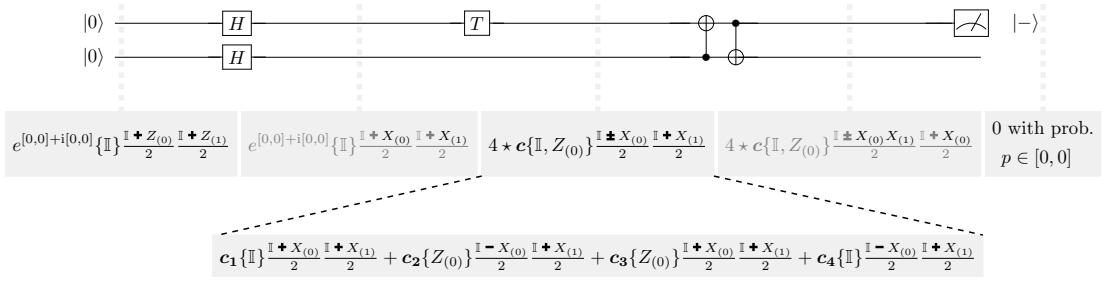


Figure 1: Overview of ABSTRAQT, where we define $\boldsymbol{c}$ and $\boldsymbol{c_1}$–$\boldsymbol{c_4}$ in §3.

Initial State. The density matrix for the initial state $|0\rangle \otimes |0\rangle$ can be represented as (see [2]):

$$\rho_A = 1\mathbb{I}\frac{\mathbb{I}+(-1)^0 Z_{(0)}}{2}\frac{\mathbb{I}+(-1)^0 Z_{(1)}}{2}.$$

To translate this to an abstract density matrix, we simply replace some elements by abstract representations. This gives the following initial abstract state:

$$\boldsymbol{\rho_A} = e^{[0,0]+[0,0]\mathrm{i}}\{\mathbb{I}\}\frac{\mathbb{I}+(-1)^{\{0\}} Z_{(0)}}{2}\frac{\mathbb{I}+(-1)^{\{0\}} Z_{(1)}}{2}. \tag{7}$$

Here we abstract booleans as sets, for instance $\{0\}$. For conciseness, in Fig. 1 we write $x \boldsymbol{+} y$, $x \boldsymbol{-} y$, and $x \boldsymbol{\pm} y$ for $x + (-1)^{\{0\}} y$, $x + (-1)^{\{1\}} y$, and $x + (-1)^{\{0,1\}} y$. Further, we represent abstract complex numbers in polar form with logarithmic length, using real intervals: 1 is represented as $e^{[0,0]+[0,0]\mathrm{i}}$, while we can over-approximate the set of complex numbers $\{1, \mathrm{i}\}$ as $e^{[0,0]+[0,\frac{\pi}{2}]\mathrm{i}}$. Finally, we abstract Pauli elements as sets, such as $\{\mathbb{I}\}$ in Fig. 1 and Eq. (7). In §4, we will clarify how we store these sets efficiently, for example representing $\{\mathbb{I}\}$ as $\mathrm{i}^{\{0\}} \cdot \{\mathbb{I}\} \otimes \{\mathbb{I}\}$ and $\{\mathrm{i} \cdot \mathbb{I}, \mathrm{i} \cdot Z_{(0)}\}$ as $\mathrm{i}^{\{1\}} \cdot \{\mathbb{I}, Z\} \otimes \{\mathbb{I}\}$.

We now explain how each operation in the circuit modifies this abstract state.

Clifford Gate Application. First, the circuit applies one Hadamard gate H to each qubit. This corresponds to the unitary operator $H_{(0)}H_{(1)}$, yielding updated abstract density matrix $\boldsymbol{\rho_B} = (H_{(0)}H_{(1)})\boldsymbol{\rho_A}(H_{(0)}H_{(1)})^\dagger$. Just as for concrete density matrices (see §2), this amounts to replacing

$$\begin{aligned}
\{\mathbb{I}\} \;&\text{ by }\; (H_{(0)}H_{(1)})\{\mathbb{I}\}(H_{(0)}H_{(1)})^\dagger = \{\mathbb{I}\},\\
Z_{(0)} \;&\text{ by }\; (H_{(0)}H_{(1)})Z_{(0)}(H_{(0)}H_{(1)})^\dagger = X_{(0)}, \text{ and}\\
Z_{(1)} \;&\text{ by }\; (H_{(0)}H_{(1)})Z_{(1)}(H_{(0)}H_{(1)})^\dagger = X_{(1)}.
\end{aligned}$$

We hence get $\boldsymbol{\rho_B} = e^{[0,0]+[0,0]\mathrm{i}}\{\mathbb{I}\}\frac{\mathbb{I}+(-1)^{\{0\}} X_{(0)}}{2}\frac{\mathbb{I}+(-1)^{\{0\}} X_{(1)}}{2}$.

Non Clifford Gate Application. Next, the circuit applies gate T on the upper qubit. To this end, we again follow the simulation described in §2. We first decompose T into Pauli elements: $T_{(0)} = d_1\mathbb{I} + d_2 Z_{(0)}$, where $d_1 \approx e^{-0.1+0.4\mathrm{i}}$ and $d_2 \approx e^{-1.0-1.2\mathrm{i}}$. Replacing T with its decomposition, we can then write $\boldsymbol{\rho_T} = T\boldsymbol{\rho_B}T^\dagger$, using Eq. (6), as:

$$\begin{aligned}
\boldsymbol{\rho_T} = \big(d_1\mathbb{I} + d_2 Z_{(0)}\big)\Big(e^{[0,0]+[0,0]\mathrm{i}}\{\mathbb{I}\}\frac{\mathbb{I}+(-1)^{\{0\}} X_{(0)}}{2}\frac{\mathbb{I}+(-1)^{\{0\}} X_{(1)}}{2}\Big)\\
\big(d_1\mathbb{I} + d_2 Z_{(0)}\big)^\dagger.
\end{aligned}$$

Analogously to §2, we can rewrite this to:

$$\begin{aligned}
&\boldsymbol{c_1}\{\mathbb{I}\}\frac{\mathbb{I}+(-1)^{\{0\}} X_{(0)}}{2}\frac{\mathbb{I}+(-1)^{\{0\}} X_{(1)}}{2}\\
+&\boldsymbol{c_2}\{Z_{(0)}\}\frac{\mathbb{I}+(-1)^{\{1\}} X_{(0)}}{2}\frac{\mathbb{I}+(-1)^{\{0\}} X_{(1)}}{2}\\
+&\boldsymbol{c_3}\{Z_{(0)}\}\frac{\mathbb{I}+(-1)^{\{0\}} X_{(0)}}{2}\frac{\mathbb{I}+(-1)^{\{0\}} X_{(1)}}{2}\\
+&\boldsymbol{c_4}\{\mathbb{I}\}\frac{\mathbb{I}+(-1)^{\{1\}} X_{(0)}}{2}\frac{\mathbb{I}+(-1)^{\{0\}} X_{(1)}}{2},
\end{aligned}$$

where

$$\begin{aligned}
\boldsymbol{c_1} &= d_1 e^{[0,0]+[0,0]\mathrm{i}} d_1^* \approx e^{[-0.2,-0.2]+[0,0]\mathrm{i}},\\
\boldsymbol{c_2} &= d_1 e^{[0,0]+[0,0]\mathrm{i}} d_2^* \approx e^{[-1.1,-1.1]+[1.6,1.6]\mathrm{i}},\\
\boldsymbol{c_3} &= d_2 e^{[0,0]+[0,0]\mathrm{i}} d_1^* \approx e^{[-1.1,-1.1]+[-1.6,-1.6]\mathrm{i}},\\
\boldsymbol{c_4} &= d_2 e^{[0,0]+[0,0]\mathrm{i}} d_2^* \approx e^{[-2.0,-2.0]+[0,0]\mathrm{i}}.
\end{aligned}$$

Merging Summands. Unfortunately, simply applying T gates as shown above may thus increase the number of summands in the abstract density matrix by a factor of 4. To counteract this, our key idea is to merge summands, by allowing a single abstract summand to represent multiple concrete ones, resulting in reduced computation overhead at the cost of lost precision. Our abstract representation allows for a straightforward merge: we take the union of sets and join intervals. Specifically, for complex numbers, we join the intervals in their representation, obtaining:

$$\boldsymbol{c} := \boldsymbol{c_1} \sqcup \boldsymbol{c_2} \sqcup \boldsymbol{c_3} \sqcup \boldsymbol{c_4} = e^{[-2.0,-0.2]+[-1.6,1.6]\mathrm{i}}.$$

Finally, we introduce the symbol $\star$ to denote how many concrete summands an abstract summand represents. Altogether, merging the summands in $\boldsymbol{\rho_T}$ yields:

$$\boldsymbol{\rho_C} = 4 \star e^{[-2.0,-0.2]+[-1.6,1.6]\mathrm{i}} \{\mathbb{I}, Z_{(0)}\} \tfrac{\mathbb{I}+(-1)^{\{0,1\}} X_{(0)}}{2} \tfrac{\mathbb{I}+(-1)^{\{0\}} X_{(1)}}{2}.$$

Note that for an abstract element $\boldsymbol{x}$, $r \star \boldsymbol{x}$ is not equivalent to $r \cdot \boldsymbol{x}$. For example, $2 \star \{0,1\} = \{0,1\} + \{0,1\} = \{0,1,2\}$, while $2 \cdot \{0,1\} = \{0,2\}$. [3]

Measurement. After the T gate, the circuit applies two additional $CNOT$ gates, resulting in the updated density matrix:

$$\boldsymbol{\rho_D} = 4 \star e^{[-2.0,-0.2]+[-1.6,1.6]\mathrm{i}} \{\mathbb{I}, Z_{(0)}\} \tfrac{\mathbb{I}+(-1)^{\{0,1\}} X_{(0)}X_{(1)}}{2} \tfrac{\mathbb{I}+(-1)^{\{0\}} X_{(0)}}{2}.$$

Finally, the circuit applies the projection $M_- = \frac{\mathbb{I}-X_{(0)}}{2}$. To update the density matrix accordingly, we closely follow [2], which showed that measurement can be reduced to simple state updates through a case distinction on M_- and the state ρ. If (i) the measurement Pauli (here $-X_{(0)}$) commutes with the product Paulis (here $(-1)^{\{0,1\}}X_{(0)}X_{(1)}$ and $(-1)^{\{0\}}X_{(1)}$) and (ii) the measurement Pauli cannot be written as a product of the product Paulis, the density matrix after measurement is 0. We will explain in §5.2 how our abstract domain allows both of these checks to be performed efficiently.

Here, both conditions are satisfied, and we hence get the final state $\boldsymbol{\rho_{M1}} = 0$. We can then compute the probability of such an outcome by $p = \mathrm{tr}\,(\boldsymbol{\rho_{M1}}) = 0$. Thus, our abstract representation was able to provide a fully precise result.

Imprecise Measurement. Suppose now that instead of the measurement in Fig. 1, we had collapsed the lower qubit to $|0\rangle$ by applying projection $M_0 = \frac{\mathbb{I}+Z_{(1)}}{2}$.

To derive the resulting state, we again follow [2] closely. We note that the measurement Pauli $+Z_{(1)}$ (i) anticommutes with the first product Pauli $(-1)^{\{0,1\}}X_{(0)}X_{(1)}$ and commutes with the second one $(-1)^{\{0\}}X_{(0)}$ and (ii) commutes with the initial Paulis $\{\mathbb{I}, Z_{(0)}\}$. In this case, we get that the density matrix is unchanged, thus $\boldsymbol{\rho_{M2}} = \boldsymbol{\rho_D}$. To compute the trace of this matrix, we follow the procedure outlined in §5.4. We omit intermediate steps here and get: [4]

$$p = \mathrm{tr}\,(\boldsymbol{\rho_{M2}}) = 4\,\mathrm{Re}(\boldsymbol{c}) \approx [0, 1.7].$$

Thus, our abstraction here is highly imprecise and does not yield any information on the measurement result (we already knew that the probability must lie in $[0,1]$).

[3] We implicitly lift concrete elements to abstract elements: $2 \cdot \{0,1\} = \{2\} \cdot \{0,1\} = \{0,2\}$.

[4] We used the precise interval bounds for $\boldsymbol{c}$ here, not the rounded values provided earlier.

Table 2: Example elements on abstract domains.

Dom.	Example element	Concretization	
$\boldsymbol{B}$	$\{0,1\}$	$\{0,1\}$	
$\boldsymbol{Z}_4$	$\{0,3\}$	$\{0,3\}$	
$\boldsymbol{R}$	$(0,1)$	$[0,1]$	$= \{r \mid 0 \leq r \leq 1\}$
$\boldsymbol{C}$	$(0,1,\pi,2\pi)$	$e^{[0,1]+[\pi,2\pi]\mathrm{i}}$	$= \{e^{r+\varphi\mathrm{i}} \mid 0 \leq r \leq 1, \pi \leq \varphi \leq 2\pi\}$
$\boldsymbol{P}_2$	$(\{0,3\},\{Z,Y\},\{X\})$	$\mathrm{i}^{\{0,3\}} \cdot \{Z,Y\} \otimes \{X\}$	$= \left\{ \mathrm{i}^b \cdot P^{(1)} \otimes P^{(2)} \;\middle|\; \begin{array}{l} b \in \{0,3\}, \\ P^{(1)} \in \{Z,Y\}, P^{(2)} \in \{X\} \end{array} \right\}$

Table 3: Summary of abstract transformers.

Transformers	Domains	Definition
$\boldsymbol{b}+\boldsymbol{c} \in \boldsymbol{B}, \boldsymbol{b}\cdot\boldsymbol{c} \in \boldsymbol{B}$	$\boldsymbol{b},\boldsymbol{c} \in \boldsymbol{B}$	Lifting to sets, Eq. (8)
$\boldsymbol{b} \sqcup \boldsymbol{c} \in \boldsymbol{B}$	$\boldsymbol{b},\boldsymbol{c} \in \boldsymbol{B}$	$\boldsymbol{b} \cup \boldsymbol{c}$
$\boldsymbol{b}+\boldsymbol{c} \in \boldsymbol{Z}_4, \boldsymbol{b}-\boldsymbol{c} \in \boldsymbol{Z}_4, \boldsymbol{b}\cdot\boldsymbol{c} \in \boldsymbol{Z}_4$	$\boldsymbol{b},\boldsymbol{c} \in \boldsymbol{Z}_4$	Lifting to sets
$\boldsymbol{b} \sqcup \boldsymbol{c} \in \boldsymbol{Z}_4$	$\boldsymbol{b},\boldsymbol{c} \in \boldsymbol{Z}_4$	$\boldsymbol{b} \cup \boldsymbol{c}$
$\boldsymbol{b} \in \boldsymbol{Z}_4$	$\boldsymbol{b} \in \boldsymbol{B}$	Embedding
$\boldsymbol{c}\cdot\boldsymbol{d} \in \boldsymbol{C}$	$\boldsymbol{c},\boldsymbol{d} \in \boldsymbol{C}$	Eq. (9)
$\boldsymbol{c} \sqcup \boldsymbol{d} \in \boldsymbol{C}$	$\boldsymbol{c},\boldsymbol{d} \in \boldsymbol{C}$	Eq. (10)
$\mathrm{Re}(\boldsymbol{c}) \in \boldsymbol{R}$	$\boldsymbol{c} \in \boldsymbol{C}^n$	Eq. (11)
$\mathrm{i}^{\boldsymbol{b}} \in \boldsymbol{C}$	$\boldsymbol{b} \in \boldsymbol{B}$	Eq. (12)
$\boldsymbol{P}\boldsymbol{Q} \in \boldsymbol{P}_n$	$\boldsymbol{P},\boldsymbol{Q} \in \boldsymbol{P}_n$	Eq. (13)
$\mathrm{f}(\boldsymbol{P}\boldsymbol{Q}) \in \boldsymbol{Z}_4$	$\boldsymbol{P},\boldsymbol{Q} \in \boldsymbol{P}_n$	Eq. (14)
$U_{(i)}\boldsymbol{P}U_{(i)}^\dagger \in \boldsymbol{P}_n$	$U \in \mathcal{U}(2^k), \boldsymbol{P} \in \boldsymbol{P}_n$	Eq. (15)
$\boldsymbol{P} \diamond \boldsymbol{Q} \in \boldsymbol{B}$	$\boldsymbol{P},\boldsymbol{Q} \in \boldsymbol{P}_n$	Eq. (16)
$\boldsymbol{P} \sqcup \boldsymbol{Q} \in \boldsymbol{P}_n$	$\boldsymbol{P},\boldsymbol{Q} \in \boldsymbol{P}_n$	Eq. (17)
$(-1)^{\boldsymbol{b}} \cdot \boldsymbol{P}$	$\boldsymbol{b} \in \boldsymbol{B}, \boldsymbol{P} \in \boldsymbol{P}_n$	Eq. (18)

4 Abstract Domains

In the following, we formalize all abstract domains (Tab. 2) underlying our abstract representation of density matrices $\boldsymbol{\rho}$ along with key abstract transformers operating on them (Tab. 3). We note that all abstract transformers introduced here naturally also support (partially) concrete arguments.

Example Elements. Tab. 2 provides an example element $\boldsymbol{x}$ of each abstract domain, along with an example of its concretization $\gamma(\boldsymbol{x})$, where $\gamma\colon \boldsymbol{\mathcal{X}} \to 2^{\mathcal{X}}$. While Tab. 2 correctly distinguishes abstract elements from their concretization, in the following, when describing operators we write concretizations instead of abstract elements (as announced in §2).

Booleans and $\mathbb{Z}_4$. Abstract booleans $\boldsymbol{b} \in \boldsymbol{B} = 2^{\mathbb{B}}$ are subsets of $\mathbb{B}$, as exemplified in Tab. 2. The addition of two abstract booleans naturally lifts boolean addition to sets and is clearly sound:

$$\boldsymbol{b} + \boldsymbol{c} = \{b + c \mid b \in \boldsymbol{b}, c \in \boldsymbol{c}\}. \tag{8}$$

We define multiplication of abstract booleans analogously. Further, we define the join of two abstract booleans as their set union.

Analogously to booleans, our abstract domain $\boldsymbol{Z}_4$ consists of subsets of $\mathbb{Z}_4$, where addition, subtraction, multiplication, and joins works analogously to abstract booleans. Further, we can straight-forwardly embed abstract booleans into $\boldsymbol{Z}_4$ by mapping 0 to 0 and 1 to 1.

Real Numbers. We abstract real numbers by intervals of the form $[\underline{a}, \overline{a}] \subseteq \mathbb{R} \cup \{\pm\infty\}$, and denote the set of such intervals by $\boldsymbol{R}$. Here, $\underline{a}$ and $\overline{a}$ indicate the lower and upper bounds of the interval, respectively. Interval addition, interval multiplication, and the cosine and exponential transformer on intervals are defined in their standard way, see §2.

Complex Numbers. We parametrize complex numbers $c \in \mathbb{C}$ in polar coordinates (with magnitude in log-space), as $c = e^{r+\varphi \mathrm{i}}$ for $r, \varphi \in \mathbb{R}$. For example, we parametrize 0 as $e^{-\infty+0\mathrm{i}}$.

Based on this parametrization, we abstract complex numbers using two real intervals for r and φ respectively, as exemplified in Tab. 2. Formally, we interpret $\boldsymbol{c} \in \boldsymbol{C}$ as the set of all possible outcomes when instantiating both intervals:

$$\gamma(\boldsymbol{c}) = e^{[\underline{r},\overline{r}]+[\underline{\varphi},\overline{\varphi}]\mathrm{i}} = \left\{ e^{r+\varphi\mathrm{i}} \;\middle|\; r \in [\underline{r},\overline{r}], \varphi \in [\underline{\varphi},\overline{\varphi}] \right\}.$$

We can compute the multiplication and join of two abstract complex numbers $\boldsymbol{c} = e^{[\underline{r},\overline{r}]+[\underline{\varphi},\overline{\varphi}]\mathrm{i}}$ and $\boldsymbol{c'} = e^{[\underline{r}',\overline{r}']+[\underline{\varphi}',\overline{\varphi}']\mathrm{i}}$ as

$$\boldsymbol{c} \cdot \boldsymbol{c'} = e^{[\underline{r}+\underline{r}',\overline{r}+\overline{r}']+[\underline{\varphi}+\underline{\varphi}',\overline{\varphi}+\overline{\varphi}']\mathrm{i}} \text{ and} \tag{9}$$

$$\boldsymbol{c} \sqcup \boldsymbol{c'} = e^{[\min(\underline{r},\underline{r}'),\max(\overline{r},\overline{r}')]+[\min(\underline{\varphi},\underline{\varphi}'),\max(\overline{\varphi},\overline{\varphi}')]\mathrm{i}}. \tag{10}$$

Again, simple arithmetic shows that Eqs. (9)–(10) are sound. We note that to increase precision, we could map complex numbers to a canonical representation before joining them, by exploiting $e^{r+\phi\mathrm{i}} = e^{r+(\phi+2\pi)\mathrm{i}}$ to ensure that $\underline{\varphi}$ lies in $[0, 2\pi]$.

We compute the real part of an abstract complex number $\boldsymbol{c} = e^{[\underline{r},\overline{r}]+[\underline{\varphi},\overline{\varphi}]\mathrm{i}}$ as

$$\mathrm{Re}(\boldsymbol{c}) = \exp([\underline{r},\overline{r}]) \cdot \cos\big([\underline{\varphi},\overline{\varphi}]\big), \tag{11}$$

where we rely on interval transformers to evaluate the right-hand side. The soundness of Eq. (11) follows from the standard formula to extract the real part from a complex number in polar coordinates. We will later use Eq. (11) to compute $\mathrm{tr}\,(\boldsymbol{\rho})$. To this end, we also need the transformer

$$\mathrm{i}^{\boldsymbol{b}} = \bigsqcup_{b \in \boldsymbol{b}} \{\mathrm{i}^b\} \in \boldsymbol{C}. \tag{12}$$

Pauli Elements. Recall that a Pauli element $P \in \mathcal{P}_n$ has the form $P = \mathrm{i}^v \cdot P^{(0)} \otimes \cdots \otimes P^{(n-1)}$, for v in $\mathbb{Z}_4$ and $P^{(k)} \in \{\mathbb{I}, X, Y, Z\}$. We therefore parametrize P as a prefactor v (in $\log_{\mathrm{i}}$ space) and n bare Paulis $P^{(k)}$.

Accordingly, we parametrize abstract Pauli elements $\boldsymbol{P} \in \boldsymbol{P}_n$ as $\mathrm{i}^{\boldsymbol{v}} \cdot \boldsymbol{P^{(0)}} \otimes \cdots \otimes \boldsymbol{P^{(n-1)}}$, where $\boldsymbol{v} \in \boldsymbol{Z}_4$ is a set of possible prefactors and $\boldsymbol{P^{(k)}} \subseteq \{X, Y, Z, \mathbb{I}_2\}$ are sets of possible Pauli matrices. Formally, we interpret $\boldsymbol{P}$ as the set of all possible outcomes when instantiating all sets:

$$\gamma(\boldsymbol{P}) = \left\{ \mathrm{i}^v \cdot \bigotimes_{i=0}^{n-1} P^{(i)} \;\middle|\; v \in \boldsymbol{v}, P^{(i)} \in \boldsymbol{P^{(i)}} \right\}.$$

We define the product of two abstract Pauli elements as:

$$\boldsymbol{PQ} = \mathrm{i}^{\mathfrak{f}(\boldsymbol{PQ})} \bigotimes_{i=0}^{n-1} \mathfrak{b}\Big(\boldsymbol{P^{(i)}Q^{(i)}}\Big). \tag{13}$$

To this end, we evaluate the prefactor induced by multiplying Paulis as

$$\mathfrak{f}(\boldsymbol{PQ}) = \mathfrak{f}(\boldsymbol{P}) + \mathfrak{f}(\boldsymbol{Q}) + \sum_{i=1}^{n} \mathfrak{f}(\boldsymbol{P^{(i)}Q^{(i)}}), \tag{14}$$

where we can evaluate the summands in the right-hand side of Eq. (14) by precomputing them for all possible sets of Pauli matrices $\boldsymbol{P^{(i)}}$ and $\boldsymbol{Q^{(i)}}$. Then, we compute the sum using Eq. (8). Analogously, we can evaluate $\mathfrak{b}\Big(\boldsymbol{P^{(i)}Q^{(i)}}\Big)$ by precomputation. The soundness of Eq. (13) follows from applying the multiplication component-wise, and then separating out prefactors from bare Paulis.

We also define the conjugation of an abstract Pauli element $\boldsymbol{P}$ with k-qubit gate U padded to n qubits as:

$$\begin{aligned} U_{(i)}\boldsymbol{P}U_{(i)}^\dagger &= U_{(i)}\Big(\mathrm{i}^{\boldsymbol{v}} \cdot \boldsymbol{P^{(0:i)}} \otimes \boldsymbol{P^{(i:i+k)}} \otimes \boldsymbol{P^{(i+k:n)}}\Big)U_{(i)}^\dagger \\ &= \mathrm{i}^{\boldsymbol{v}+\mathfrak{f}(U\boldsymbol{P^{(i:i+k)}}U^\dagger)} \cdot \boldsymbol{P^{(0:i)}} \otimes \mathfrak{b}(U\boldsymbol{P^{(i:i+k)}}U^\dagger) \otimes \boldsymbol{P^{(i+k:n)}}, \end{aligned} \tag{15}$$

where $\boldsymbol{P^{(i:j)}}$ denotes $\boldsymbol{P^{(i)}} \otimes \cdots \otimes \boldsymbol{P^{(j-1)}}$. Because k is typically small, and all possible gates U are known in advance, we can efficiently precompute $\mathfrak{f}(U\boldsymbol{P^{(i:i+k)}}U^\dagger)$ and $\mathfrak{b}(U\boldsymbol{P^{(i:i+k)}}U^\dagger)$. We note that this only works if the result of conjugation is indeed an (abstract) Pauli element—if not, this operation throws an error[5]. The soundness from Eq. (15) follows from applying U to qubits i through $i+k$, and then separating out prefactors from bare Paulis.

We define the commutator $\boldsymbol{P} \diamond \boldsymbol{Q}$ of two abstract Pauli elements $\boldsymbol{P}$ and $\boldsymbol{Q}$ as

$$\left(\mathrm{i}^{\boldsymbol{v}} \cdot \bigotimes_{i=0}^{n-1} \boldsymbol{P^{(i)}}\right) \diamond \left(\mathrm{i}^{\boldsymbol{w}} \cdot \bigotimes_{i=0}^{n-1} \boldsymbol{Q^{(i)}}\right) = \sum_{i=1}^{n} \boldsymbol{P^{(i)}} \diamond \boldsymbol{Q^{(i)}}. \tag{16}$$

Here, we evaluate the sum using Eq. (8), and efficiently evaluate $\boldsymbol{P^{(i)}} \diamond \boldsymbol{Q^{(i)}} \in \boldsymbol{B}$ by precomputing:

$$\boldsymbol{P^{(i)}} \diamond \boldsymbol{Q^{(i)}} = \left\{ P^{(i)} \diamond Q^{(i)} \;\middle|\; P^{(i)} \in \boldsymbol{P^{(i)}}, Q^{(i)} \in \boldsymbol{Q^{(i)}} \right\}.$$

The soundness of Eq. (16) can be derived from the corresponding concrete equation, which can be verified using standard linear algebra.

We define the join of abstract Pauli elements as

$$\left(\mathrm{i}^{\boldsymbol{v}} \bigotimes_{i=0}^{n-1} \boldsymbol{P^{(i)}}\right) \sqcup \left(\mathrm{i}^{\boldsymbol{w}} \bigotimes_{i=0}^{n-1} \boldsymbol{Q^{(i)}}\right) = \mathrm{i}^{\boldsymbol{v} \sqcup \boldsymbol{w}} \bigotimes_{i=0}^{n-1} \left(\boldsymbol{P^{(i)}} \cup \boldsymbol{Q^{(i)}}\right), \tag{17}$$

where $\boldsymbol{P^{(i)}} \cup \boldsymbol{Q^{(i)}} \subseteq \{\mathbb{I}, X, Y, Z\}$. Clearly, this join is sound.

Finally, we define an abstract transformer for modifying the sign of an abstract Pauli element $\boldsymbol{P}$ by:

$$(-1)^{\boldsymbol{b}} \cdot \left(\mathrm{i}^{\boldsymbol{v}} \cdot \bigotimes_{i=0}^{n-1} \boldsymbol{P^{(i)}}\right) = \mathrm{i}^{\boldsymbol{v}+2\cdot\boldsymbol{b}} \cdot \bigotimes_{i=0}^{n-1} \boldsymbol{P^{(i)}} \tag{18}$$

The soundness of Eq. (18) follows directly from $(-1)^v = \mathrm{i}^{2v}$.

Abstract Density Matrices. The concrete and abstract domains introduced previously allow us to represent an abstract density matrix $\boldsymbol{\rho} \in \boldsymbol{D}$ as follows:

$$\boldsymbol{\rho} = r \star \boldsymbol{c} \cdot \boldsymbol{P} \cdot \prod_{j=1}^{n} \tfrac{\mathbb{I}+(-1)^{\boldsymbol{b}_j} Q_j}{2}. \tag{19}$$

Here, $r \in \mathbb{N}$, $\boldsymbol{c} \in \boldsymbol{C}$, $\boldsymbol{P} \in \boldsymbol{P}_n$, $\boldsymbol{b}_j \in \boldsymbol{B}$, and $Q_j \in \mathcal{P}_n$. Note that Q_j are concrete Pauli elements, while $\boldsymbol{P}$ is abstract. Further, both $\boldsymbol{P}$ and Q_j can have a prefactor, i.e., are not necessarily bare Paulis. Here, the integer counter r records how many concrete summands were abstracted. Specifically, $r \star \boldsymbol{x}$ is defined as $\sum_{i=1}^{r} \boldsymbol{x}$. Overall, we interpret $\boldsymbol{\rho}$ as:

$$\gamma(\boldsymbol{\rho}) = \left\{ \sum_{i=1}^{r} c_i P_i \prod_{j=1}^{n} \tfrac{\mathbb{I}+(-1)^{b_{ij}} Q_j}{2} \;\middle|\; c_i \in \gamma(\boldsymbol{c}), P_i \in \gamma(\boldsymbol{P}), b_{ij} \in \gamma(\boldsymbol{b}_j) \right\}, \tag{20}$$

relying on the previously discussed interpretations of $\mathbb{C}$, $\mathcal{P}_n$, and $\mathbb{B}$.

5 Abstract Transformers

We now formalize the abstract transformers used by Abstraqt to simulate quantum circuits. The soundness of all transformers is straightforward, except for the trace transformer (§5.4) which we discuss in App. A.

[5] We can recover from this error by decomposing U as a sum of bare Pauli elements, as mentioned in §2, see also Eqs. (21)–(22).

Initialization. We start from initial state $\otimes_{i=1}^{n} |0\rangle$, which corresponds to density matrix

$$\boldsymbol{\rho} = \prod_{j=1}^{n} \tfrac{\mathbb{I}+Z_{(j)}}{2} = 1 \star e^{[0,0]+\mathrm{i}[0,0]} \cdot \mathrm{i}^{\{0\}} \{\mathbb{I}\} \prod_{j=1}^{n} \tfrac{\mathbb{I}+(-1)^{\{0\}} Z_{(j)}}{2},$$

as established in [2, Sec. III]. We note that we can prepare other starting states by applying appropriate gates to the starting state $\otimes_{i=1}^{n} |0\rangle$.

5.1 Gate Application

Analogously to the concrete case discussed in §2, applying a unitary gate U to $\boldsymbol{\rho}$ yields:

$$U\boldsymbol{\rho}U^\dagger = r \star \boldsymbol{cP'} \prod_{j=1}^{n} \tfrac{\mathbb{I}+(-1)^{\boldsymbol{b}_j} Q'_j}{2}, \tag{21}$$

for $\boldsymbol{P'} = U\boldsymbol{P}U^\dagger$ and $Q'_j = UQ_jU^\dagger$.

If either $U\boldsymbol{P}U^\dagger \notin \mathcal{P}_n$ or $UQ_jU^\dagger \notin \mathcal{P}_n$, Eq. (21) still holds, but we cannot represent the resulting matrices efficiently. In this case, again analogously to §2, we instead decompose the offending gate as $U = \sum_p d_p R_p$, with $R_p \in \mathcal{P}_n$ and obtain

$$U\boldsymbol{\rho}U^\dagger = \sum_{pq} r \star \boldsymbol{c_{pq}} \boldsymbol{P_{pq}} \prod_{j=1}^{n} \tfrac{\mathbb{I}+(-1)^{\boldsymbol{b_{jq}}} Q_j}{2}, \tag{22}$$

for $\boldsymbol{c}_{pq} = d_p \boldsymbol{c} d_q^*$, $\boldsymbol{P'}_{pq} = R_p \boldsymbol{P} R_q$, and $\boldsymbol{b_{jq}} = \boldsymbol{b_j} + Q_j \diamond R_q$.

Overall, we can evaluate Eqs. (21)–(22) by relying on the abstract transformers from §4.

Compression. To prevent an exponential blow-up of the number of summands and to adhere to the abstract domain of $\boldsymbol{\rho}$ which does not include a sum, we compress all summands to a single one. Two summands can be joined as follows:

$$\begin{aligned}
&\left(r_1 \star \boldsymbol{c}_1 \boldsymbol{P}_1 \prod_{j=1}^{n} \tfrac{\mathbb{I}+(-1)^{\boldsymbol{b}_{1j}} Q_j}{2} \right) \sqcup \left(r_2 \star \boldsymbol{c}_2 \boldsymbol{P}_2 \prod_{j=1}^{n} \tfrac{\mathbb{I}+(-1)^{\boldsymbol{b}_{2j}} Q_j}{2} \right) \\
&= r \star \boldsymbol{cP} \prod_{j=1}^{n} \tfrac{\mathbb{I}+(-1)^{\boldsymbol{b}_j} Q_j}{2},
\end{aligned}$$

where $r = r_1 + r_2$, $\boldsymbol{c} = \boldsymbol{c}_1 \sqcup \boldsymbol{c}_2$, $\boldsymbol{b}_j = \boldsymbol{b}_{1j} \sqcup \boldsymbol{b}_{2j}$, and $\boldsymbol{P} = \boldsymbol{P}_1 \sqcup \boldsymbol{P}_2$. The key observation here is that the concrete Q_j are independent of the summand, and thus need not be joined.

We note that we could also only merge *some* summands and leave the others precise—investigating the effect of more flexible merging strategies could be interesting future research.

5.2 Measurement

We now describe how to perform Pauli measurements, by extending the (concrete) stabilizer simulation to abstract density matrices. The correctness of the concrete simulation was previously established in [2, Sec. VII.C], while the correctness of the abstraction is immediate.

Simulating Measurement. Applying a Pauli measurement in basis $R \in \mathfrak{b}(\mathcal{P}_n)$ has a probabilistic outcome and transforms ρ to $\rho_+ = \frac{\mathbb{I}+R}{2}\rho\frac{\mathbb{I}+R}{2}$ with probability $\mathrm{tr}(\rho_+)$ or $\rho_- = \frac{\mathbb{I}-R}{2}\rho\frac{\mathbb{I}-R}{2}$ with probability $\mathrm{tr}(\rho_-)$. We describe how to compute ρ_+. Computing ρ_- works analogously by using $-R$ instead of R.

In the following, we will consider a concrete state ρ as defined in §2 and an abstract state $\boldsymbol{\rho}$ as defined in Eq. (19):

$$\rho = \sum_{i=1}^{m} c_i P_i \prod_{j=1}^{n} \tfrac{\mathbb{I}+(-1)^{b_{ij}} Q_j}{2} \quad \text{and} \quad \boldsymbol{\rho} = r \star \boldsymbol{cP} \prod_{j=1}^{n} \tfrac{\mathbb{I}+(-1)^{\boldsymbol{b}_j} Q_j}{2}. \tag{23}$$

Concrete simulation of measurement distinguishes two cases: either (i) R commutes with all Q_j or (ii) R anti-commutes with at least one Q_j. Note that as the Q_j are concrete in an abstract state $\boldsymbol{\rho}$, those two cases translate directly to the abstract setting. We now describe both cases for concrete and abstract simulation.

Background: Concrete Case (i). In this case, we assume R commutes with all Q_j. Focusing on a single summand ρ_i of ρ, measurement maps it to (see [2]):

$$\rho_{i,+} = c_i \tfrac{\mathbb{I}+R}{2} P_i \tfrac{\mathbb{I}+R}{2} \prod_{j=1}^{n} \tfrac{\mathbb{I}+(-1)^{b_{ij}} Q_j}{2}. \tag{24}$$

Let us first introduce the notation $\{(-1)^{b_{ij}} Q_j\} \rightsquigarrow R$, denoting that R can be written as a product of selected Pauli elements from $\{(-1)^{b_{ij}} Q_j\}$. Symmetrically, we write $\{(-1)^{b_{ij}} Q_j\} \not\rightsquigarrow R$ if R cannot be written as such a product. As shown in [2], if $\{(-1)^{b_{ij}} Q_j\} \rightsquigarrow R$ then $\frac{\mathbb{I}+R}{2} \prod_{j=1}^{n} \frac{\mathbb{I}+(-1)^{b_{ij}} Q_j}{2}$ is equal to $\prod_{j=1}^{n} \frac{\mathbb{I}+(-1)^{b_{ij}} Q_j}{2}$ and if $\{(-1)^{b_{ij}} Q_j\} \not\rightsquigarrow R$ then $\frac{\mathbb{I}+R}{2} \prod_{j=1}^{n} \frac{\mathbb{I}+(-1)^{b_{ij}} Q_j}{2}$ is null. Further, using that $R^2 = \mathbb{I}$, we get from Eq. (24) that if P_i commutes with R, $\rho_{i,+}$ is equal to ρ_i, otherwise, P_i anti-commutes with R and $\rho_{i,+}$ is null. Putting it all together, we finally get:

$$\rho_+ = \sum_{i=1}^{m} \rho_{i,+} = \sum_{i=1}^{m} \begin{cases} c_i P_i \prod_{j=1}^{n} \frac{\mathbb{I}+(-1)^{b_{ij}} Q_j}{2} & \text{if } \{(-1)^{b_{ij}} Q_j\} \rightsquigarrow R \text{ and } R \diamond P_i = 0, \\ 0 & \text{if } \{(-1)^{b_{ij}} Q_j\} \not\rightsquigarrow R \text{ or } R \diamond P_i = 1. \end{cases} \tag{25}$$

Abstract Case (i). Let us first define $\rightsquigarrow^u$ and $\not\rightsquigarrow^u$ for a concrete R, concrete Q_j and abstract $\boldsymbol{b_j}$. We say $\{(-1)^{\boldsymbol{b_j}} Q_j\} \rightsquigarrow^u R$ if for all j, for all $b_j \in \gamma(\boldsymbol{b_j})$, we have $\{(-1)^{b_j} Q_j\} \rightsquigarrow R$. Similarly, we say $\{(-1)^{\boldsymbol{b_j}} Q_j\} \not\rightsquigarrow^u R$ if for all j, for all $b_j \in \gamma(\boldsymbol{b_j})$, we have $\{(-1)^{b_j} Q_j\} \not\rightsquigarrow R$. Note that $\rightsquigarrow^u$ and $\not\rightsquigarrow^u$ are under-approximations, and there can exist some R and $\{(-1)^{\boldsymbol{b_j}} Q_j\}$ such that neither apply. Using those two abstract relations, we get the abstract transformer for $\boldsymbol{\rho_+}$:

$$r \star \begin{cases} \boldsymbol{c P} \prod_{j=1}^{n} \frac{\mathbb{I}+(-1)^{\boldsymbol{b_j}} Q_j}{2} & \text{if } \{(-1)^{\boldsymbol{b_j}} Q_j\} \rightsquigarrow^u R \text{ and } R \diamond \boldsymbol{P} = \{0\}, \\ 0 & \text{if } \{(-1)^{\boldsymbol{b_j}} Q_j\} \not\rightsquigarrow^u R \text{ or } R \diamond \boldsymbol{P} = \{1\}, \\ (\boldsymbol{c} \sqcup \{0\})\, \boldsymbol{P} \prod_{j=1}^{n} \frac{\mathbb{I}+(-1)^{\boldsymbol{b_j}} Q_j}{2} & \text{otherwise.} \end{cases} \tag{26}$$

We can evaluate Eq. (26) by relying on the abstract transformers from Tab. 3 and by evaluating $\rightsquigarrow^u$ as discussed shortly.

Background: Concrete Case (ii). We now suppose R anti-commutes with at least one Q_j. In this case, we can rewrite ρ such that R anti-commutes with Q_1, and commutes with all other Q_j. Specifically, we can select any Q_{j^*} which anti-commutes with R, swap b_{ij^*} and Q_{j^*} with b_{i1} and Q_1, and replace all other Q_j anti-commuting with R by $Q_1 Q_j$ (and analogously b_{ij} by $b_{ij} + b_{i1}$), which leaves ρ invariant (see [2]). Assuming ρ is the result after this rewrite, we have:

$$\rho_+ = \sum_i \tfrac{1}{2} c_i P'_i \tfrac{\mathbb{I}+(-1)^0 R}{2} \prod_{j=2}^{n} \tfrac{\mathbb{I}+(-1)^{b_{ij}} Q_j}{2}, \tag{27}$$

$$\text{where } P'_i = \begin{cases} P_i & \text{if } R \diamond P_i = 0, \\ (-1)^{b_{i1}} P_i Q_1 & \text{if } R \diamond P_i = 1. \end{cases}$$

Overall, after rewriting ρ as above, Eq. (27) replaces c_i by $\frac{1}{2} c_i$, P_i by P'_i, b_{i1} by 0, and Q_1 by R.

Abstract Case (ii). In the abstract case, we first apply the same rewrite as in the concrete case, where we pick j^* as the first j for which Q_j anti-commutes with R. [6] Then, directly abstracting

[6] We could also consider other strategies than picking the first possible j, for example picking a j for which $\boldsymbol{b_j}$ is precise whenever possible, to increase precision.

Eq. (27) yields:

$$\boldsymbol{\rho}_+ = r \star \tfrac{1}{2}\boldsymbol{c}\boldsymbol{P'}\tfrac{\mathbb{I}+(-1)^{\{0\}}R}{2}\prod_{j=2}^{n}\tfrac{\mathbb{I}+(-1)^{\boldsymbol{b_j}}Q_j}{2}, \tag{28}$$

$$\text{where } \boldsymbol{P'} = \begin{cases} \boldsymbol{P} & \text{if } R \diamond \boldsymbol{P} = \{0\}, \\ (-1)^{\boldsymbol{b_1}}\boldsymbol{P}Q_1 & \text{if } R \diamond \boldsymbol{P} = \{1\}, \\ \boldsymbol{P} \sqcup (-1)^{\boldsymbol{b_1}}\boldsymbol{P}Q_1 & \text{otherwise.} \end{cases}$$

Here, we replace $\boldsymbol{c}$ by $\frac{1}{2}\boldsymbol{c}$, $\boldsymbol{P}$ by $\boldsymbol{P'}$, $\boldsymbol{b_1}$ by $\{0\}$, and Q_1 by R. When defining $\boldsymbol{P'}$, we follow the two cases from Eq. (27) when our abstraction is precise enough to indicate which case we should choose, or join the results of both cases otherwise. Again, we can evaluate Eq. (28) by relying on the abstract transformers from Tab. 3.

Joining Both Measurement Results. For measurements occurring within a quantum circuit, stabilizer simulation generally requires randomly selecting either ρ_+ or ρ_- with probability $\mathrm{tr}(\rho_+)$ and $\mathrm{tr}(\rho_-)$, respectively, and then continues only with the selected state. In contrast, Abstraqt can join both measurement outcomes into a single abstract state $\boldsymbol{\rho}_+ \sqcup \boldsymbol{\rho}_-$, as the Q_j are the same in both. This allows us to pursue both measurement outcomes simultaneously, as we demonstrate in §7.

5.3 Efficiently computing $\rightsquigarrow$

To simulate the result of a measurement, we introduced the new operator $\{(-1)^{b_j}Q_j\} \rightsquigarrow R$, denoting that some Pauli R can be written as a product of $\{(-1)^{b_j}Q_j\}$. We now show how to compute $\rightsquigarrow$ efficiently.

Background: Concrete case. We first note that $\{(-1)^{b_j}Q_j\} \rightsquigarrow R$ holds if and only if there exist some $x \in \mathbb{B}^n$ such that:

$$R \stackrel{!}{=} \prod_{j=1}^{n}\left((-1)^{b_j}Q_j\right)^{x_j}. \tag{29}$$

Further, this solution x would satisfy:

$$\mathfrak{b}(R) \stackrel{!}{=} \mathfrak{b}\left(\prod_{j=1}^{n}\left((-1)^{b_j}Q_j\right)^{x_j}\right) \tag{30}$$

Eq. (30) has a solution if and only if R commutes with all the Q_j, in which case this solution x is unique (see [2]). Hence, to check if $\{(-1)^{b_j}Q_j\} \rightsquigarrow R$, we can first verify whether $R \diamond Q_j = 0$ for all j, and if so, check if the unique x satisfying Eq. (30) also satisfies Eq. (29).

Background: Finding x for Eq. (30). To compute this solution x, the stabilizer simulation relies critically on an isomorphism $\mathfrak{g}$ between Pauli matrices $\{\mathbb{I}, X, Y, Z\}$ and $\mathbb{B}^2$.

Specifically, $\mathfrak{g}$ maps I to $\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)$, X to $\left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$, Y to $\left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$, and Z to $\left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$. Further, $\mathfrak{g}$ extends naturally to bare Pauli elements $R \in \mathfrak{b}(\mathcal{P}_n)$ and tuples $Q = (Q_1, \ldots, Q_n) \in \mathfrak{b}(\mathcal{P}_n)^n$ by:

$$\mathfrak{g}(R) = \begin{pmatrix} \mathfrak{g}(R^{(0)}) \\ \vdots \\ \mathfrak{g}(R^{(n-1)}) \end{pmatrix} \text{ and } \mathfrak{g}(Q) = \begin{pmatrix} \mathfrak{g}(Q_1^{(0)}) & \cdots & \mathfrak{g}(Q_n^{(0)}) \\ \vdots & \ddots & \vdots \\ \mathfrak{g}(Q_1^{(n-1)}) & \cdots & \mathfrak{g}(Q_n^{(n-1)}) \end{pmatrix},$$

where $\mathfrak{g}(R) \in \mathbb{B}^{2n\times 1}$ and $\mathfrak{g}(Q) \in \mathbb{B}^{2n\times n}$. We can naturally extend $\mathfrak{g}$ to $\mathcal{P}_n$, by defining $\mathfrak{g}(R) = \mathfrak{g}(\mathfrak{b}(R))$.

This isomorphism $\mathfrak{g}$ is designed so that the product of bare Pauli elements ignoring prefactors corresponds to a component-wise addition of encodings:

$$\mathfrak{g}(P_1P_2) = \mathfrak{g}(P_1) + \mathfrak{g}(P_2). \tag{31}$$

Using Eq. (31), we can obtain solution candidates x for Eq. (30) by solving a system of linear equations using Gaussian elimination modulo 2:

$$\mathfrak{g}(R) \stackrel{!}{=} \mathfrak{g}\left(\prod_{j=1}^{n} Q_j^{x_j}\right) = \sum_{j=1}^{n} \mathfrak{g}(Q_j) x_j = \mathfrak{g}(Q) x. \tag{32}$$

Because in our case, $\mathfrak{g}(Q)$ is over-determined and has full rank, Eq. (32) either has no solution, or a unique solution x.

Background: Checking prefactors. Once we have found the unique x (if it exists) satisfying Eq. (30) as described above, we need to check if it also satisfies Eq. (29). It is enough to check if the prefactors match:

$$\mathfrak{f}(R) \stackrel{!}{=} \mathfrak{f}\left(\prod_j (-1)^{b_j x_j} Q_j^{x_j}\right),$$

or equivalently:

$$\mathfrak{f}(R) - \mathfrak{f}\left(\prod_j Q_j^{x_j}\right) - 2\sum_j b_j x_j \stackrel{!}{=} 0,$$

where the subtraction and sum operations are over $\mathbb{Z}_4$.

Putting it all together, we can define $\mathfrak{F}\colon \mathcal{P}_n \times \mathcal{P}_n^n \times \mathbb{B}^n \to \mathbb{Z}_4 \cup \{\lightning\}$ with

$$\mathfrak{F}(R, Q, b) = \begin{cases} \lightning & \text{if } \exists j, R \diamond Q_j = 1, \\ \mathfrak{f}(R) - \mathfrak{f}\left(\prod_{j=1}^{n} Q_j^{x_j}\right) - 2\sum_{j=1}^{n} x_j b_j & \text{otherwise,} \end{cases} \tag{33}$$

where x is the unique value such that $\mathfrak{g}(R) = \mathfrak{g}(Q)x$ and $\lightning$ indicates there is no such x. We then have that $\{(-1)^{b_j} Q_j\} \rightsquigarrow R$ if and only if $\mathfrak{F}(R, Q, b) = 0$, or equivalently, $\{(-1)^{b_j} Q_j\} \not\rightsquigarrow R$ if and only if $\mathfrak{F}(R, Q, b) \neq 0$.

$\mathfrak{F}$ for abstract $\boldsymbol{b_j}$. For abstract values $\boldsymbol{b_j}$, we define $\mathfrak{F}\colon \mathcal{P}_n \times \mathcal{P}_n^n \times \boldsymbol{B}^n \to 2^{\mathbb{Z}_4 \cup \{\lightning\}}$ as follows:

$$\mathfrak{F}(R, Q, \boldsymbol{b}) = \begin{cases} \{\lightning\} & \text{if } \exists j, R \diamond Q_j = 1, \\ \mathfrak{f}(R) - \mathfrak{f}\left(\prod_{j=1}^{n} Q_j^{x_j}\right) - 2\sum_{j=1}^{n} x_j \boldsymbol{b_j} & \text{otherwise.} \end{cases} \tag{34}$$

Following the same reasoning as above, we have $\{(-1)^{\boldsymbol{b_j}} Q_j\} \rightsquigarrow^u R$ if and only if $\mathfrak{F}(R, Q, \boldsymbol{b}) = \{0\}$ and $\{(-1)^{\boldsymbol{b_j}} Q_j\} \not\rightsquigarrow^u R$ if and only if $\mathfrak{F}(R, Q, \boldsymbol{b}) \cap \{0\} = \emptyset$.

$\mathfrak{F}$ for abstract $\boldsymbol{b_j}$ and $\boldsymbol{R}$. To compute the trace of a state (see §5.4), we further extend Eq. (33) to abstract $\boldsymbol{b_j}$ and abstract $\boldsymbol{R}$, and define $\mathfrak{F}\colon \boldsymbol{P}_n \times \mathcal{P}_n^n \times \boldsymbol{B}^n \to 2^{\mathbb{Z}_4 \cup \{\lightning\}}$ as:

$$\mathfrak{F}(\boldsymbol{R}, Q, \boldsymbol{b}) = \begin{cases} \{\lightning\} & \text{if } \exists j. \boldsymbol{R} \diamond Q_j = \{1\}, \\ \mathfrak{f}(\boldsymbol{R}) - \mathfrak{f}\left(\prod_{j=1}^{n} Q_j^{\boldsymbol{x_j}}\right) - 2\sum_{j=1}^{n} \boldsymbol{x_j b_j} & \text{if } \forall j. \boldsymbol{R} \diamond Q_j = \{0\}, \\ \mathfrak{f}(\boldsymbol{R}) - \mathfrak{f}\left(\prod_{j=1}^{n} Q_j^{\boldsymbol{x_j}}\right) - 2\sum_{j=1}^{n} \boldsymbol{x_j b_j} \cup \{\lightning\} & \text{otherwise,} \end{cases} \tag{35}$$

$$\text{for } \mathfrak{g}(\boldsymbol{R}) = \mathfrak{g}(Q)\boldsymbol{x}. \tag{36}$$

Here, evaluating Eq. (35) requires evaluating $Q_j^{\boldsymbol{b}}$ for an abstract boolean $\boldsymbol{b}$, which we define naturally as

$$Q_j^{\boldsymbol{b}} := \begin{cases} \{Q_j\} & \text{if } \boldsymbol{b} = \{1\}, \\ \{\mathbb{I}\} & \text{if } \boldsymbol{b} = \{0\}, \\ \{Q_j, \mathbb{I}\} & \text{if } \boldsymbol{b} = \{0, 1\}. \end{cases}$$

Further, Eq. (36) requires over-approximating all x which satisfy the linear equation $\mathfrak{g}(\boldsymbol{R}) = \mathfrak{g}(Q)x$. Here, we naturally extend $\mathfrak{g}$ to abstract Paulis by joining their images. For instance, we have that $\mathfrak{g}(\{X, Y\}) = \{\binom{1}{0}\} \sqcup \{\binom{1}{1}\} = \binom{\{1\}}{\{0,1\}}$. We then view $\mathfrak{g}(\boldsymbol{R}) = \mathfrak{g}(Q)x$ as a system of linear equations $\boldsymbol{b} = Ax$, where the left-hand side consists of abstract booleans $\boldsymbol{b} \in \boldsymbol{B}^{2n}$. We then drop all equations in this equation system where the left-hand side is $\{0, 1\}$, as they do not constrain the solution space. This updated system is fully concrete, hence we can solve it using Gaussian elimination. We get either no solution, or a solution space $y + \sum_{k=1}^{p} \lambda_k u_k$, where y is a possible solution and $u_1, ..., u_p$ is a possibly empty basis of the null solution space. In the case of no solution, $\boldsymbol{x}$ is not needed in Eq. (35). Otherwise, we can compute $\boldsymbol{x_j}$ as $\{y_j + \sum_{k=1}^{m} \lambda_k u_{k,j} \mid \lambda_k \in \mathbb{B}\}$.

5.4 Trace

Recall that the probability of obtaining state ρ_+ when measuring ρ is $\mathrm{tr}\,(\rho_+)$. We now describe how to compute this trace using $\mathfrak{S}$ defined above.

Background: Concrete Trace. Following [2], we compute the trace of a density matrix ρ by:

$$\mathrm{tr}\,(\rho) = \sum_{i=1}^{m} \mathrm{Re}\left(c_i \mathrm{i}^{\mathfrak{S}(P,Q,b_i)}\right), \tag{37}$$

where we define $\mathrm{i}^{\not\in} := 0$. Because the trace of a density matrix is always real, $\mathrm{Re}(\cdot)$ is redundant, but will be convenient to avoid complex traces in our abstraction.

Abstract Trace. For an abstract state ρ, we define:

$$\mathrm{tr}\,(\boldsymbol{\rho}) = r \cdot \mathrm{Re}\left(\boldsymbol{c}\mathrm{i}^{\mathfrak{S}(\boldsymbol{P},Q,\boldsymbol{b})}\right), \tag{38}$$

where we use $\mathfrak{S}(\cdot)$ as defined in Eq. (35).

6 Implementation

In the following, we discuss our implementation of the abstract transformers from §4 and §5 in Abstraqt.

Language and Libraries. We implemented Abstraqt in Python 3.8, relying on Qiskit 0.40.0 [14] for handling quantum circuits, and a combination of NumPy 1.20.0 [15] and Numba 0.54 [16] to handle matrix operations.

Bit Encodings. An abstract density matrix $\boldsymbol{\rho} = r \star \boldsymbol{c} \cdot \boldsymbol{P} \cdot \prod_{j=1}^{n} \frac{\mathbb{I}+(-1)^{\boldsymbol{b}_j} Q_j}{2}$ is encoded as a tuple $(r, \boldsymbol{c}, \boldsymbol{P}, \boldsymbol{b_1}, ..., \boldsymbol{b_n}, Q_1, \ldots, Q_n)$. To encode the concrete Pauli matrices Q_j, we follow concrete stabilizer simulation encodings such as [17] and encode Pauli matrices P using two bits $\mathfrak{g}(P)$ (see §5.3). To encode abstract elements of a finite set we use bit patterns. For example, we encode $\boldsymbol{b_1} = \{1, 0\} \in \boldsymbol{B}$ as 11_2, where the least significant bit (i.e. the right-most bit) indicates that $0 \in \boldsymbol{b_1}$. Analogously, we encode $\boldsymbol{v} = \{3, 0\} \in \boldsymbol{Z}_4$ as 1001_2. Further, we encode $\{Z, Y\}$ as 1100_2, where the indicator bits correspond to Z, Y, X, and $\mathbb{I}$, respectively, from left to right. Hence the abstact Pauli $\boldsymbol{P} = (\{0, 3\}, \{Z, Y\}, \{X\})$ would be represented as $(1001_2, 1100_2, 0010_2)$.

Implementing Transformers. The abstract transformers on abstract density matrices can be implemented using operations in $\boldsymbol{B}, \boldsymbol{Z}_4, \boldsymbol{C}$, and $\boldsymbol{P}_1$. As $\boldsymbol{B}, \boldsymbol{Z}_4$, and $\boldsymbol{P}_1$ are small finite domains, we can implement operations in these domains using lookup tables, which avoids the need for bit manipulation tricks. While such tricks are applicable in our context (e.g., [2] uses bit manipulations to compute $H_{(i)} P H_{(i)}^{\dagger}$ for $P \in \mathcal{P}_n$), they are generally hard to come up with [18]. In contrast, the efficiency of our lookup tables is comparable to that of bit manipulation tricks, without requiring new insights for new operations.

For example, to evaluate $\{\} + \{0\}$ over $\boldsymbol{B}$ using Eq. (8), we encode the first argument $\{\}$ as 00 and the second argument $\{0\}$ as 01. Looking up entry $(00, 01)$ in a two-dimensional pre-computed table then yields 00, the encoding of the correct result $\{\}$. We note that we cannot implement this operation directly using a XOR instruction on encodings, as this would yield incorrect results: 00 XOR 01 = 01 $\simeq \{0\}$, which is incorrect.

Gaussian Elimination. To efficiently solve equations modulo two as discussed in §5, we implemented a custom Gaussian elimination relying on bit-packing (i.e., storing 32 boolean values in a single 32-bit integer). In the future, it would be interesting to explore if Gaussian elimination could be avoided altogether, as suggested by previous works [2, 17].

Testing. To reduce the likelihood of implementation errors, we have complemented Abstraqt with extensive automated tests. We test that abstract transformers $f^\sharp$ are sound with respect to concrete functions f, that is to say that

$$\forall x_1 \in \gamma(\boldsymbol{x_1}) \cdots \forall x_k \in \gamma(\boldsymbol{x_k}).f(x_1, \ldots, x_n) \in f^\sharp(\boldsymbol{x_1}, \ldots, \boldsymbol{x_k}).$$

We check this inclusion for multiple selected samples of $\boldsymbol{x_i}$ and $x_i \in \boldsymbol{x_i}$ (typically corner cases).

This approach is highly effective at catching implementation errors, which we have found in multiple existing tools as shown in §7.

7 Evaluation

We now present our evaluation of Abstraqt, demonstrating that it can establish circuit properties no existing tool can establish.

7.1 Benchmarks

To evaluate Abstraqt, we generated 12 benchmark circuits, summarized and visualized in Tab. 4.

Benchmark Circuit Generation. Each circuit operates on 62 qubits, partitioned into 31 *upper* qubits and 31 *lower* qubits. We picked the limit of 62 qubits because our baseline ESS (discussed shortly) only supports up to 63 qubits; Abstraqt is not subject to such a limitation.

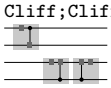
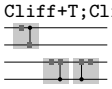
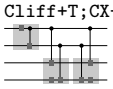
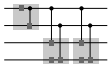
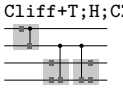
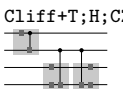
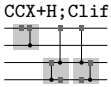
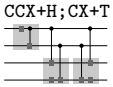
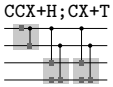
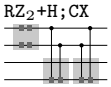
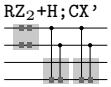
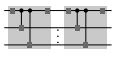
Each circuit operates on initial state $|0\rangle$ and is constructed to ensure that all lower qubits are eventually reverted to state $|0\rangle$. We chose this invariant as it can be expressed for most of the evaluated tools, as we will discuss in §7.2. Further, as some tools can only check this for one qubit at a time, we only check if the very last qubit is reverted to $|0\rangle$, instead of running 31 independent checks (which would artificially slow down some baselines). Note that this check is of equivalent difficulty for all lower qubits.

Benchmark Details. Tab. 4 details how each benchmark circuit was generated. Most of the circuits are built from three concatenated subcircuits. First, c_1 modifies the upper qubits, then c_2 modifies the lower qubits (potentially using gates controlled by the upper qubits) and finally c_3 reverts all lower qubits to $|0\rangle$, but in a non-trivial way. Circuit `CCX+H;Cliff` slightly deviates from this pattern, as it also modifies the upper qubits using gates controller by lower qubits. Further, circuits `Cliff+T;H;CZ+RX` and `Cliff+T;H;CZ+RX'` additionally apply two layers of H gates to the lower qubits. Finally, circuit `MeasureGHZ` applies internal measurements, as discussed below.

The majority of circuits revert the lower qubits to $|0\rangle$ by applying c_3, the inverse of c_2 but optimized using PyZX [19]—this obfuscates the fact that c_2 and c_3 cancel out. Four circuits, marked with a trailing prime (`'`), generate c_3 by optimizing the un-inverted c_2. They still reset all lower qubits to $|0\rangle$, but establishing this requires advanced reasoning. Specifically, `RZ2+H;CX'` flips each lower qubit an even number of times. [7] Similarly, `Cliff+T;CX+T'` and `CCX+H;CX+T'` additionally modify the phase but still flip each lower qubit an even number of times. Finally,

[7] More precisely, when representing the quantum state as a sum over computational basis states, an even number of flips are applied to each qubit of each summand.

Table 4: Description of benchmark circuits, where upper $= \{1, \ldots, 31\}$ and lower $= \{32, \ldots, 62\}$.

Circuit	**Generation**	**Gates** (approx.)
`Cliff;Cliff` 	$c_1 \in \left(\{o(q) \mid o \in \{H, S\}, q \in \text{upper}\} \cup \{CX(q_1, q_2) \mid q_1, q_2 \in \text{upper}\}\right)^{10^4}$ $c_2 \in \left(\{o(q) \mid o \in \{H, S\}, q \in \text{lower}\} \cup \{CX(q_1, q_2) \mid q_1, q_2 \in \text{lower}\}\right)^{10^4}$ return $c_1; c_2; \text{opt}(c_2^\dagger)$	26k × Clifford
`Cliff+T;Cliff` 	$c_1 \in \left(\{o(q) \mid o \in \{H, S, T\}, q \in \text{upper}\} \cup \{CX(q_1, q_2) \mid q_1, q_2 \in \text{upper}\}\right)^{10^4}$ $c_2 \in \left(\{o(q) \mid o \in \{H, S\}, q \in \text{lower}\} \cup \{CX(q_1, q_2) \mid q_1, q_2 \in \text{lower}\}\right)^{10^4}$ return $c_1; c_2; \text{opt}(c_2^\dagger)$	23k × Clifford, 2.5k × T
`Cliff+T;CX+T` 	$c_1 \in \left(\{o(q) \mid o \in \{H, S, T\}, q \in \text{upper}\} \cup \{CX(q_1, q_2) \mid q_1, q_2 \in \text{upper}\}\right)^{10^4}$ $c_2 \in \left(\{CX(q_1, q_2) \mid q_1 \in \text{upper}, q_2 \in \text{lower}\} \cup \{T(q) \mid q \in \text{lower}\}\right)^{10^4}$ return $c_1; c_2; \text{opt}(c_2^\dagger)$	18k × Clifford, 9k × T, 40 × $T^\dagger$
`Cliff+T;CX+T'` 	$c_1 \in \left(\{o(q) \mid o \in \{H, S, T\}, q \in \text{upper}\} \cup \{CX(q_1, q_2) \mid q_1, q_2 \in \text{upper}\}\right)^{10^4}$ $c_2 \in \left(\{CX(q_1, q_2) \mid q_1 \in \text{upper}, q_2 \in \text{lower}\} \cup \{T(q) \mid q \in \text{lower}\}\right)^{10^4}$ return $c_1; c_2; \text{opt}(c_2)$	18k × Clifford, 9k × T, 40 × $T^\dagger$
`Cliff+T;H;CZ+RX` 	$c_1 \in \left(\{o(q) \mid o \in \{H, S, T\}, q \in \text{upper}\} \cup \{CX(q_1, q_2) \mid q_1, q_2 \in \text{upper}\}\right)^{10^4}$ $c_h = H(32); \ldots; H(62)$ $c_2 \in \left(\{CZ(q_1, q_2) \mid q_1 \in \text{upper}, q_2 \in \text{lower}\} \cup \{RX_{\frac{\pi}{4}}(q) \mid q \in \text{lower}\}\right)^{10^4}$ return $c_1; c_h; c_2; \text{opt}(c_2^\dagger); c_h$	18k × Clifford, 5k × $RX_{\frac{\pi}{4}}$, 3k × T, 1k × $T^\dagger$
`Cliff+T;H;CZ+RX'` 	$c_1 \in \left(\{o(q) \mid o \in \{H, S, T\}, q \in \text{upper}\} \cup \{CX(q_1, q_2) \mid q_1, q_2 \in \text{upper}\}\right)^{10^4}$ $c_h = H(32); \ldots; H(62)$ $c_2 \in \left(\{CZ(q_1, q_2) \mid q_1 \in \text{upper}, q_2 \in \text{lower}\} \cup \{RX_{\frac{\pi}{4}}(q) \mid q \in \text{lower}\}\right)^{10^4}$ return $c_1; c_h; c_2; \text{opt}(c_2); c_h$	18k × Clifford, 5k × $RX_{\frac{\pi}{4}}$, 4k × T, 40 × $T^\dagger$
`CCX+H;Cliff` 	$c_1 \in \left(\{CCX(q_1, q_2, q_3) \mid q_1, q_2, q_3 \in \text{upper}\} \cup \{H(q) \mid q \in \text{upper}\}\right)^{10^4}$ $c_2 \in \left(\{o(q) \mid o \in \{H, S\}, q \in \text{lower}\} \cup \{CX(q_1, q_2) \mid q_1 \in \text{lower}, q_2 \in \text{lower} \cup \text{upper}\}\right)^{10^4}$ return $c_1; c_2; \text{opt}(c_2^\dagger)$	22k × Clifford, 5k × CCX
`CCX+H;CX+T` 	$c_1 \in \left(\{CCX(q_1, q_2, q_3) \mid q_1, q_2, q_3 \in \text{upper}\} \cup \{H(q) \mid q \in \text{upper}\}\right)^{10^4}$ $c_2 \in \left(\{CX(q_1, q_2) \mid q_1 \in \text{upper}, q_2 \in \text{lower}\} \cup \{T(q) \mid q \in \text{lower}\}\right)^{10^4}$ return $c_1; c_2; \text{opt}(c_2^\dagger)$	16k × Clifford, 5k × CCX, 5k × T, 1k × $T^\dagger$
`CCX+H;CX+T'` 	$c_1 \in \left(\{CCX(q_1, q_2, q_3) \mid q_1, q_2, q_3 \in \text{upper}\} \cup \{H(q) \mid q \in \text{upper}\}\right)^{10^4}$ $c_2 \in \left(\{CX(q_1, q_2) \mid q_1 \in \text{upper}, q_2 \in \text{lower}\} \cup \{T(q) \mid q \in \text{lower}\}\right)^{10^4}$ return $c_1; c_2; \text{opt}(c_2)$	16k × Clifford, 5k × CCX, 7k × T, 30 × $T^\dagger$
`RZ`$_2$`+H;CX` 	$c_1 \in \left(\{o(q) \mid o \in \{RZ_2, H\}, q \in \text{upper}\}\right)^{10^4}$ $c_2 \in \left(\{CX(q_1, q_2) \mid q_1 \in \text{upper}, q_2 \in \text{lower}\}\right)^{10^4}$ return $c_1; c_2; \text{opt}(c_2^\dagger)$	16k × Clifford, 5k × RZ_2
`RZ`$_2$`+H;CX'` 	$c_1 \in \left(\{o(q) \mid o \in \{RZ_2, H\}, q \in \text{upper}\}\right)^{10^4}$ $c_2 \in \left(\{CX(q_1, q_2) \mid q_1 \in \text{upper}, q_2 \in \text{lower}\}\right)^{10^4}$ return $c_1; c_2; \text{opt}(c_2)$	16k × Clifford, 5k × RZ_2
`MeasureGHZ` 	$c_1 = CX(1, 2); \ldots; CX(1, 62)$ $c_2 = H(1); c_1; \text{measure}(1), c_1$ return $c_2; \ldots; c_2$ (100 times)	12k × Clifford, 100 × measure

`Cliff+T;H;CZ+RX'` flips between states $|+\rangle$ and $|-\rangle$ an even number of times, where $RX_{\frac{\pi}{4}}$ only modifies the phase.

The last benchmark `MeasureGHZ` first generates a GHZ state $\frac{1}{\sqrt{2}}|0\cdots0\rangle + \frac{1}{\sqrt{2}}|1\cdots1\rangle$, and collapses it to $|0\cdots0\rangle$ or $|1\cdots1\rangle$ by measuring the first qubit. Then, it resets all qubits to $|0\rangle$ except for the first one. It then repeats this process, with the first qubit starting in either $|0\rangle$ or $|1\rangle$. Thus, the state before measurement is either $\frac{1}{\sqrt{2}}|0\cdots0\rangle + \frac{1}{\sqrt{2}}|1\cdots1\rangle$ or $\frac{1}{\sqrt{2}}|0\cdots0\rangle - \frac{1}{\sqrt{2}}|1\cdots1\rangle$, but every repetition still resets all lower qubits to $|0\rangle$.

Discussion. Our benchmark covers a wide variety of gates, with all applying Clifford gates, seven applying T gates, three applying CCX gates, two applying $RX_{\frac{\pi}{4}}$ gates (one qubit gate, rotation around the X axis of $\frac{\pi}{4}$ radians), and two applying RZ_2 gates (one qubit gate, rotation around the Z axis of 2 radians).

All benchmarks are constructed to revert the lower qubits to $|0\rangle$, but in a non-obvious way. As fully precise simulation of most benchmarks is unrealistic, we expect that over-approximation is typically necessary to establish this fact.

7.2 Baselines

We now discuss how we instantiated existing tools to establish that a circuit c evolves a qubit q to state $|0\rangle$. Overall, we considered two tools based on stabilizer simulation (ESS [5] and QuiZX [4]), one tool based on the Feynman path integral (Feynman [20]), one tool based on abstract interpretation (YP21 [21], in two different modes), and one tool based on state vectors (Statevector as implemented by Qiskit [14]).

ESS. Qiskit [14] provides an extended stabilizer simulator implementing the ideas published in [5] which (i) decomposes quantum circuits into Clifford circuits, (ii) simulates these circuits separately, and (iii) performs measurements by an aggregation across these circuits. To check if a circuit c consistently evolves a qubit q to $|0\rangle$, we check if c extended by a measurement of q always yields 0. To run our simulation, we used default parameters.

QuiZX. QuiZX [4] improves upon [5] by alternating between decomposing circuits (splitting non-Clifford gates into Clifford gates) and optimizing the decomposed circuits (which may further reduce non-Clifford gates). We can use QuiZX to establish that a qubit is in state $|0\rangle$ by "plugging" output q as $|1\rangle$ and establishing that the probability of this output is zero. [8]

Feynman. Feynman [20] allows to verify quantum circuits based on the Feynman path integral. Its implementation[9] supports two main use cases, namely optimization and checking the equivalence of two circuits. While these use cases cannot prove that a circuit resets a qubit to $|0\rangle$, we can use Feynman's equivalence check to check whether the circuits in Tab. 4 are equivalent to a simplified version which performs no operation at all on lower qubits. We check this equivalence for all circuits, even for those where we know it does not hold (namely all whose name ends with a prime), allowing us to confirm that Feynman cannot scale to any of our benchmarks (see §7.3).

We note that Feynman currently does not support internal measurements. [10]

YP21. Like Abstraqt, YP21 [21] also uses abstract interpretation, but relies on projectors instead of stabilizer simulation. Specifically, it encodes the abstract state of selected (small) subsets of qubits as projectors $\{P_j\}_{j\in\mathcal{J}}$, which constrain the state of these qubits to the range of P_j.

To check if a qubit q is in state $|0\rangle$, we check if the subspace resulting from intersecting the range of all P_j is a subset of the range of $\mathbb{I} + Z_{(q)}$—an operation which is natively supported by YP21.

[8] The use of plugging is described on https://github.com/Quantomatic/quizx/issues/9.

[9] Tool available at https://github.com/meamy/feynman

[10] https://github.com/meamy/feynman/issues/8

Table 5: Success rates when running simulators on benchmarks from Tab. 4.

Label	Abstraqt	QuiZX	ESS	Feynman	YP21 (mode 1)	YP21 (mode 2)	Statevec.
`Cliff;Cliff`	**100%**	0% (E)	0% (I)	0% (T,M)	0% (T,P)	0% (I)	0% (M)
`Cliff+T;Cliff`	**100%**	70% (T)	0% (I)	0% (T,M)	0% (T,P)	0% (I)	0% (M)
`Cliff+T;CX+T`	**100%**	80% (M)	0% (M)	0% (T)	0% (T,E,P)	0% (I)	0% (M)
`Cliff+T;CX+T'`	**100%**	0% (M)	0% (M)	0% (T)	0% (T,E,P)	0% (I)	0% (M)
`Cliff+T;H+CZ+RX`	**100%**	60% (M)	0% (M)	0% (T)	0% (T,P)	0% (I)	0% (M)
`Cliff+T;H+CZ+RX'`	**100%**	0% (T,M)	0% (M)	0% (T)	0% (T,P)	0% (I)	0% (M)
`CCX+H;Cliff`	**100%**	0% (T)	0% (M)	0% (M)	0% (T)	0% (T)	0% (M)
`CCX+H;CX+T`	**100%**	50% (T)	0% (M)	0% (T)	0% (T)	0% (T)	0% (M)
`CCX+H;CX+T'`	**100%**	0% (T,M)	0% (M)	0% (T)	0% (T)	0% (T)	0% (M)
RZ_2`+H;CX`	**100%**	0% (E)	0% (T,M)	0% (U)	0% (U)	0% (U)	0% (M)
RZ_2`+H;CX'`	**100%**	0% (E)	0% (M)	0% (U)	0% (U)	0% (U)	0% (M)
`MeasureGHZ`	**100%**	0% (U)	0% (U)	0% (U)	0% (U)	0% (U)	0% (U)
Overall success	**100%**	22%	0%	0%	0%	0%	0%

T: timeout (6h), M: out of memory, U: unsupported operation in the circuit,
I: incorrect simulation results, P: too imprecise, E: internal error

When running YP21, we used the two execution modes suggested in its original evaluation [21]. The first mode tracks the state of all pairs of qubits, while the second considers subsets of 5 qubits that satisfy a particular condition (for details, see [21, §9]). Because [21] does not discuss which execution mode to pick for new circuits, we evaluated all circuits in both modes.

We note that because YP21 does not support $CX(a, b)$ for $a > b$, we instead encoded such gates as $H(a); H(b); CX(b, a); H(b); H(a)$.

Statevector. Qiskit [14] further provides a simulator based on state vectors, which we also used for completeness.

Abstraqt. In Abstraqt, we can establish that a qubit is in state $|0\rangle$ by measuring the final abstract state $\boldsymbol{\rho}$ in basis $Z_{(i)}$ and checking if the probability of obtaining $|1\rangle$ is 0.

Experimental Setup. We executed all experiments on a machine with 110 GB RAM and 56 cores at 2.6 GHz, running Ubuntu 22.04. Because some tools consumed excessive amounts of memory, we limited them to 12 GB of RAM. This was not necessary for Abstraqt, which never required more than 600 MB of RAM. We limited each tool to a single thread.

7.3 Results

Tab. 5 summarizes the results when using all tools discussed in §7.2 to establish that the last qubit in 10 randomly selected instantiations of each benchmark from Tab. 4 is in state $|0\rangle$. Overall, it demonstrates that while Abstraqt can establish this for all benchmarks within minutes, QuiZX can only establish it for a few instances, and all other tools cannot establish it for any benchmark. Further, we found that for some circuits the established simulation tool ESS yields incorrect results. We now discuss the results of each tool in more details.

MeasureGHZ. Importantly, no baseline tool except Abstraqt can simultaneously simulate both outcomes of a measurement, without incurring an exponential blow-up. Therefore, for `MeasureGHZ`, we consider internal measurements as an unsupported operation in these tools. We note that we could randomly select one measurement outcome and simulate the remainder of the circuit for it, but then we can only establish that the final state is $|0\rangle$ *for a given sequence of measurement outcomes*. In contrast, a single run of Abstraqt can establish that the final state is $|0\rangle$ for all possible measurement outcomes (see also §5.2).

QuiZX. As QuiZX is the only baseline tool solving some of our benchmark instances, we provide a detailed comparison to it in Tab. 6.

Overall, QuiZX cannot consistently handle any of the benchmarks from Tab. 4, Instead, it often either times out or runs out of memory. Further, QuiZX consistently runs into an internal error

Table 6: Detailed comparison of outcomes from Abstraqt and QuiZX, including runtimes of successful runs.

Label	Abstraqt			QuiZX		
	Outcomes	min [s]	max [s]	Outcomes	min [s]	max [s]
`Cliff;Cliff`	$10\times$✓	24	33	$0\times$✓, $10\times$E	-	-
`Cliff+T;Cliff`	$10\times$✓	32	47	$7\times$✓, $3\times$T	$5.5\cdot10^3$	$2.0\cdot10^4$
`Cliff+T;CX+T`	$10\times$✓	46	63	$8\times$✓, $2\times$M	$2.0\cdot10^3$	$9.4\cdot10^3$
`Cliff+T;CX+T'`	$10\times$✓	47	65	$0\times$✓,$10\times$T	-	-
`Cliff+T;H+CZ+RX`	$10\times$✓	58	69	$6\times$✓, $4\times$M	$3.6\cdot10^3$	$1.4\cdot10^4$
`Cliff+T;H+CZ+RX'`	$10\times$✓	52	71	$0\times$✓, $1\times$T, $9\times$M	-	-
`CCX+H;Cliff`	$10\times$✓	143	155	$0\times$✓,$10\times$T	-	-
`CCX+H;CX+T`	$10\times$✓	155	173	$5\times$✓, $5\times$T	$5.9\cdot10^3$	$7.9\cdot10^3$
`CCX+H;CX+T'`	$10\times$✓	155	173	$0\times$✓, $1\times$T, $9\times$M	-	-
RZ_2`+H;CX`	$10\times$✓	37	47	$0\times$✓, $10\times$E	-	-
RZ_2`+H;CX'`	$10\times$✓	37	46	$0\times$✓, $10\times$E	-	-
`MeasureGHZ`	$10\times$✓	23	32	$0\times$✓, $10\times$U	-	-

T: timeout (6h), M: out of memory, U: unsupported operation in the circuit, E: internal error

when simulating RZ_2`+H;CX` and RZ_2`+H;CX'`. Surprisingly, QuiZX also consistently fails to simulate `Cliff;Cliff`, which we conjecture is due to a bug for circuits that do not contain non-Clifford gates. After adding a single T gate, simulation is successful.

Importantly, even when QuiZX succeeds, it is significantly slower than Abstraqt, sometimes by more than two orders of magnitude.

ESS. Surprisingly, ESS simulates circuits `Cliff;Cliff` and `Cliff+T;Cliff` incorrectly. Specifically, it samples the impossible measurement of 1 around 50% of cases. Interestingly, smaller circuits generated with the same process are handled correctly. It is reassuring to see that Abstraqt allows us to discover such instabilities in established tools.

It may be surprising that ESS returns an incorrect result for `Cliff+T;Cliff` instead of timing out, although the circuit contains many T gates—this is because Qiskit can establish that the Clifford+T part of the circuit is irrelevant when measuring the last qubit. For all remaining circuits, ESS runs out of memory or times out, as it decomposes the circuit into exponentially many Clifford circuits.

Feynman. Feynman consistently either times out, runs out of memory, or does not support a relevant operation (namely measurement and RZ_2).

YP21. YP21 typically either times out, throws an internal error, does not support a relevant operation (e.g., measurements or RZ_2), or returns incorrect results. The latter is because on some circuits, mode 2 choses an empty set of projectors, which leads to trivially unsound results. When YP21 does terminate, it is too imprecise to establish that the last qubit is in state $|0\rangle$.

Statevector. Unsurprisingly, statevector simulation cannot handle the circuits in Tab. 5. This is because it requires space exponential in the number of qubits, which precludes simulating any of the benchmarks.

7.4 Limitations and Discussion

We note that our benchmarks are designed to showcase successful applications of Abstraqt where it outperforms existing tools. Of course, Abstraqt is not precise on all circuits—e.g., Abstraqt quickly loses precision on general Clifford+T circuits (analogously to the imprecise measurement discussed in §3).

Future Abstractions. We expect that for many real-world circuits, existing approaches work better than the current implementation of Abstraqt. However, as Abstraqt only abstracts the first stabilizer simulation generalized to non-Clifford gates [2, §VII-C], we believe it paves the way to also abstract more recent stabilizer simulators. For example, ESS [5] operates on so-called *CH-forms* which, like the generalized stabilizer simulation underlying Abstraqt, can be encoded using bits and complex numbers. Hence, it seems plausible that our ideas could be adapted to abstract ESS. QuiZX operates on *ZX-diagrams* consisting of graphs whose nodes are parametrized by rotation angles α. Again, a promising direction for future research is introducing abstract ZX-diagrams that support abstract rotation angles. This is particularly promising because both ESS and QuiZX scale better in number of T gates than [2, §VII-C]: with 2^n instead of 4^n.

We note however that not all concrete simulation techniques are directly amenable to abstraction. For example, when naively abstracting the Clifford simulation by Aaronson and Gottesmann, applying a measurement requires selecting an entry in an boolean matrix that definitively equals one [2, Case I in §III]—it is unclear how to generalize this to abstract boolean matrices whose entries may be $\{0, 1\}$.

Improving Abstraqt. Another promising route towards better abstractions in incrementally improving Abstraqt itself. For example, it would be interesting to consider the effect of keeping more than one abstract summand, abstracting P_i or b_{ij} using a custom relational domain (which retains information about the relationship between different values) [22], or a more precise abstraction for complex numbers by taking into account that restricted gate sets such as Clifford+T only induce matrices over finite sets of values.

Summary. Overall, we believe that all tools in Tab. 5 are valuable to analyze quantum circuits. We are hoping that addressing some limitations of the considered baselines (e.g., fixing bugs in QuiZX and ESS) and cross-pollinating ideas (e.g., extending QuiZX by abstract interpretation) will allow the community to benefit from the fundamentally different mathematical foundations of all tools.

8 Related Work

Here, we discuss works related to the goal and methods of Abstraqt.

Quantum Abstract Interpretation. Some existing works have investigated abstract interpretation for simulating quantum circuits [21, 23, 24]. As [21] is not specialized for Clifford circuits, it is very imprecise on the circuits investigated in §7: it cannot derive that the lower qubits are $|0\rangle$ for any of them. While [23, 24] are inspired by stabilizer simulation, they only focus on determining if certain qubits are entangled or not, whereas Abstraqt can extract more precise information about the state. Further, both tools are inherently imprecise on non-Clifford gates—in contrast, a straight-forward extension of Abstraqt can treat some non-Clifford gates precisely at the exponential cost of not merging summands.

Stabilizer Simulation. The Gottesman-Knill theorem [1] established that stabilizers can be used to efficiently simulate Clifford circuits. Stim [17] is a recent implementation of such a simulator, which only supports Clifford gates and Pauli measurements.

Stabilizer simulation was extended to allow for non-Clifford gates at an exponential cost, while still allowing efficient simulation of Clifford gates [2, §VII-C]. Various works build upon this insight, handling Clifford gates efficiently but suffering from an exponential blow-up on non-Clifford gates [3, 4, 5, 6, 7]. In our evaluation, we demonstrate that Abstraqt extends the reach of state-of-the-art stabilizer simulation by comparing to two tools from this category, ESS [5] (chosen because it is implemented in the popular Qiskit library) and QuiZX [4] (chosen because it is a recent tool reporting favorable runtimes).

Verifying Quantum Programs. Another approach to establishing circuit properties is end-to-end formal program verification, as developed in [25] for instance. However, this approach often requires new insights for each program it is applied to. Even though recent works have greatly improved verification automation, proving even the simplest programs still requires a significant time investment [26], whereas our approach can analyze it without any human time investment.

The work [27] automatically generates rich invariants, but is exponential in the number of qubits, limiting its use to small circuits. Finally, [20] can automatically verify the equivalence of two given circuits, but times out on the benchmarks considered in §7.

9 Conclusion

In this work, we have demonstrated that combining abstract interpretation with stabilizer simulation allows to establish circuit properties that are intractable otherwise.

Our key idea was to over-approximate the behavior of non-Clifford gates in the generalized stabilizer simulation of Aaronson and Gottesman [2] by merging summands in the sum representation of the quantum states density matrix. Our carefully chosen abstract domain allows us to define efficient abstract transformers that approximate each of the concrete stabilizer simulation functions, including measurement.

References

[1] Daniel Gottesman. "The Heisenberg Representation of Quantum Computers". Technical Report arXiv:quant-ph/9807006. arXiv (1998).

[2] Scott Aaronson and Daniel Gottesman. "Improved Simulation of Stabilizer Circuits". Physical Review A **70**, 052328 (2004).

[3] Robert Rand, Aarthi Sundaram, Kartik Singhal, and Brad Lackey. "Extending gottesman types beyond the clifford group". In The Second International Workshop on Programming Languages for Quantum Computing (PLanQC 2021). (2021). url: https://pldi21.sigplan.org/details/planqc-2021-papers/9/Extending-Gottesman-Types-Beyond-the-Clifford-Group.

[4] Aleks Kissinger and John van de Wetering. "Simulating quantum circuits with ZX-calculus reduced stabiliser decompositions". Quantum Science and Technology **7**, 044001 (2022).

[5] Sergey Bravyi, Dan Browne, Padraic Calpin, Earl Campbell, David Gosset, and Mark Howard. "Simulation of quantum circuits by low-rank stabilizer decompositions". Quantum **3**, 181 (2019).

[6] Hakop Pashayan, Oliver Reardon-Smith, Kamil Korzekwa, and Stephen D. Bartlett. "Fast estimation of outcome probabilities for quantum circuits". PRX Quantum **3**, 020361 (2022).

[7] "Classical simulation of quantum circuits with partial and graphical stabiliser decompositions". Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022).

[8] Patrick Cousot and Radhia Cousot. "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints". In Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. Pages 238–252. POPL '77New York, NY, USA (1977). ACM.

[9] Patrick Cousot and Radhia Cousot. "Abstract interpretation frameworks". Journal of logic and computation **2**, 511–547 (1992).

[10] Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérome Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. "A static analyzer for large safety-critical software". ACM SIGPLAN Notices **38**, 196–207 (2003).

[11] Francesco Logozzo and Manuel Fähndrich. "Pentagons: A weakly relational abstract domain for the efficient validation of array accesses". Science of Computer Programming **75**, 796–807 (2010).

[12] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. "AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation". In 2018 IEEE Symposium on Security and Privacy (SP). Pages 3–18. San Francisco, CA (2018). IEEE.

[13] Michael A. Nielsen and Isaac L. Chuang. "Quantum computation and quantum information: 10th anniversary edition". Cambridge University Press. (2010).

[14] Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, Luciano Bello, Yael Ben-Haim, David Bucher, Francisco Jose Cabrera-Hernández, Jorge Carballo-Franquis, Adrian Chen, Chun-Fu Chen, Jerry M. Chow, Antonio D. Córcoles-Gonzales, Abigail J. Cross, Andrew Cross, Juan Cruz-Benito, Chris Culver, Salvador De La Puente González, Enrique De La Torre, Delton Ding, Eugene Dumitrescu, Ivan Duran, Pieter Eendebak, Mark Everitt, Ismael Faro Sertage, Albert Frisch, Andreas Fuhrer, Jay Gambetta, Borja Godoy Gago, Juan Gomez-Mosquera, Donny Greenberg, Ikko Hamamura, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Hiroshi Horii, Shaohan Hu, Takashi Imamichi, Toshinari Itoko, Ali Javadi-Abhari, Naoki Kanazawa, Anton Karazeev, Kevin Krsulich, Peng Liu, Yang Luh, Yunho Maeng, Manoel Marques, Francisco Jose Martín-Fernández, Douglas T. McClure, David McKay, Srujan Meesala, Antonio Mezzacapo, Nikolaj Moll, Diego Moreda Rodríguez, Giacomo Nannicini, Paul Nation, Pauline Ollitrault, Lee James O'Riordan, Hanhee Paik, Jesús Pérez, Anna Phan, Marco Pistoia, Viktor Prutyanov, Max Reuter, Julia Rice, Abdón Rodríguez Davila, Raymond Harry Putra Rudy, Mingi Ryu, Ninad Sathaye, Chris Schnabel, Eddie Schoute, Kanav Setia, Yunong Shi, Adenilton Silva, Yukio Siraichi, Seyon Sivarajah, John A. Smolin, Mathias Soeken, Hitomi Takahashi, Ivano Tavernelli, Charles Taylor, Pete Taylour, Kenso Trabing, Matthew Treinish, Wes Turner, Desiree Vogt-Lee, Christophe Vuillot, Jonathan A. Wildstrom, Jessica Wilson, Erick Winston, Christopher Wood, Stephen Wood, Stefan Wörner, Ismail Yunus Akhalwaya, and Christa Zoufal. "Qiskit: An open-source framework for quantum computing" (2019).

[15] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. "Array programming with NumPy". Nature **585**, 357–362 (2020).

[16] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. "Numba: a LLVM-based Python JIT compiler". In Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC. Pages 1–6. LLVM '15New York, NY, USA (2015). Association for Computing Machinery.

[17] Craig Gidney. "Stim: a fast stabilizer circuit simulator". Quantum **5**, 497 (2021).

[18] Henry S. Warren. "Hacker's delight". Addison-Wesley Professional. (2012). 2nd edition.

[19] Aleks Kissinger and John van de Wetering. "PyZX: Large Scale Automated Diagrammatic Reasoning". In Bob Coecke and Matthew Leifer, editors, Proceedings 16th International Conference on Quantum Physics and Logic, Chapman University, Orange, CA, USA., 10-14 June 2019. Volume 318 of Electronic Proceedings in Theoretical Computer Science, pages 229–241. Open Publishing Association (2020).

[20] Matthew Amy. "Towards Large-scale Functional Verification of Universal Quantum Circuits". Electronic Proceedings in Theoretical Computer Science **287**, 1–21 (2019).

[21] Nengkun Yu and Jens Palsberg. "Quantum abstract interpretation". In Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. Pages 542–558. PLDI 2021New York, NY, USA (2021). Association for Computing Machinery.

[22] Antoine Miné. "Weakly Relational Numerical Abstract Domains". PhD Thesis (2004). url: https://www-apr.lip6.fr/~mine/these/these-color.pdf.

[23] Simon Perdrix. "Quantum Entanglement Analysis Based on Abstract Interpretation". In Proceedings of the 15th International Symposium on Static Analysis. Pages 270–282. SAS '08Berlin, Heidelberg (2008). Springer-Verlag.

[24] Kentaro Honda. "Analysis of Quantum Entanglement in Quantum Programs using Stabilizer Formalism". Electronic Proceedings in Theoretical Computer Science**195** (2015).

[25] Kesha Hietala, Robert Rand, Shih-Han Hung, Liyi Li, and Michael Hicks. "Proving Quantum Programs Correct". Leibniz International Proceedings in Informatics (LIPIcs) **193**, 21:1–21:19 (2021).

[26] Christophe Chareton, Sébastien Bardin, François Bobot, Valentin Perrelle, and Benoît Valiron. "An automated deductive verification framework for circuit-building quantum programs". In Programming Languages and Systems. Pages 148–177. Springer International Publishing (2021).

[27] Mingsheng Ying, Shenggang Ying, and Xiaodi Wu. "Invariants of quantum programs: Characterisations and generation". SIGPLAN Not. **52**, 818–832 (2017).

A Abstract Transformers Soundness

Here, we prove the soundness of the trace transformer Eq. (38):

Theorem A.1. *Trace. For all* $\boldsymbol{\rho} \in \mathbb{D}$ *we have*

$$\gamma \circ \mathrm{tr}\,(\boldsymbol{\rho}) \supseteq \mathrm{tr} \circ \gamma(\boldsymbol{\rho}).$$

Proof. The over-approximation $\mathfrak{S}^{\#}$ follows closely the form of $\mathfrak{S}$, where the first term $\mathfrak{f}(\boldsymbol{P})$ over-approximates the prefactors of $\boldsymbol{P}$ and second term over-approximates the prefactors originating from the solution space for y of $\mathfrak{b}(\boldsymbol{P}) = \mathfrak{b}(\prod_{j=1}^{n} Q_j^{y_j})$. Overall, we have:

$$\begin{aligned}
\mathrm{tr} \circ \gamma\,(\boldsymbol{\rho}) &= \mathrm{tr}\left(\left\{\sum_{i=1}^{r} c_i P_i \prod_{j=1}^{n} \tfrac{1}{2}\left(\mathbb{I} + (-1)^{b_{ij}} Q_j\right) \;\middle|\; c_i \in \boldsymbol{c}, P_i \in \boldsymbol{P}, b_{ij} \in \boldsymbol{b}_j \right\}\right) \\
&= \left\{\mathrm{tr}\left(\sum_{i=1}^{r} c_i P_i \prod_{j=1}^{n} \tfrac{1}{2}\left(\mathbb{I} + (-1)^{b_{ij}} Q_j\right)\right) \;\middle|\; c_i \in \boldsymbol{c}, P_i \in \boldsymbol{P}, b_{ij} \in \boldsymbol{b}_j \right\} \\
&= \left\{\sum_{i=1}^{r} \mathrm{Re}\left(c_i \mathrm{i}^{\mathfrak{S}(P,Q,b_{ij})}\right) \;\middle|\; c_i \in \boldsymbol{c}, P_i \in \boldsymbol{P}, b_{ij} \in \boldsymbol{b}_j \right\} && \text{Concrete trace, §5.4} \\
&= \sum_{i=1}^{r} \mathrm{Re}\left(\{c_i \in \boldsymbol{c}\} \cdot \mathrm{i}^{\mathfrak{S}(\{P_i \in \boldsymbol{P}\},Q,\{b_{ij} \in \boldsymbol{b}_j\})}\right) \\
&\subseteq \gamma\left(\sum_{i=1}^{r} \mathrm{Re}\left(\boldsymbol{c} \cdot \mathrm{i}^{\mathfrak{S}^{\#}(\boldsymbol{P},Q,\boldsymbol{b}_j)}\right)\right) && \text{Soundness of transf.} \\
&= \gamma\left(r \cdot \mathrm{Re}\left(\boldsymbol{c} \cdot \mathrm{i}^{\mathfrak{S}^{\#}(\boldsymbol{P},Q,\boldsymbol{b}_j)}\right)\right) && \text{Property of intervals} \\
&= \gamma \circ \mathrm{tr}\,(\boldsymbol{\rho})\,.
\end{aligned}$$

□

Table 7: States stabilized by Pauli matrices P and also $-P$, where $X := \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, $Y := \left(\begin{smallmatrix} 0 & -\mathrm{i} \\ \mathrm{i} & 0 \end{smallmatrix}\right)$, $Z := \left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$, and $\mathbb{I}_2 := \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$.

Stab.	**State vec.**	**Dens. mat.**	**Stab.**	**State vec.**	**Dens. mat.**
X	$\frac{1}{\sqrt{2}}\left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right) \mathrel{\widehat{=}} \lvert+\rangle$	$\frac{1}{2}\left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$	$-X$	$\frac{1}{\sqrt{2}}\left(\begin{smallmatrix} 1 \\ -1 \end{smallmatrix}\right) \mathrel{\widehat{=}} \lvert-\rangle$	$\frac{1}{2}\left(\begin{smallmatrix} 1 & -1 \\ -1 & 1 \end{smallmatrix}\right)$
Y	$\frac{1}{\sqrt{2}}\left(\begin{smallmatrix} 1 \\ \mathrm{i} \end{smallmatrix}\right)$	$\frac{1}{2}\left(\begin{smallmatrix} 1 & \mathrm{i} \\ -\mathrm{i} & 1 \end{smallmatrix}\right)$	$-Y$	$\frac{1}{\sqrt{2}}\left(\begin{smallmatrix} 1 \\ -\mathrm{i} \end{smallmatrix}\right)$	$\frac{1}{2}\left(\begin{smallmatrix} 1 & -\mathrm{i} \\ \mathrm{i} & 1 \end{smallmatrix}\right)$
Z	$\left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right) \mathrel{\widehat{=}} \lvert0\rangle$	$\left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right)$	$-Z$	$\left(\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right) \mathrel{\widehat{=}} \lvert1\rangle$	$\left(\begin{smallmatrix} 0 & 0 \\ 0 & 1 \end{smallmatrix}\right)$
$\mathbb{I}_2$	(any vec.)	-	$-\mathbb{I}_2$	(no vec.)	-

Table 8: Multiplication of Pauli matrices.

$\mathbb{I}\mathbb{I} = \mathbb{I}$	$\mathbb{I}X = X$	$\mathbb{I}Y = Y$	$\mathbb{I}Z = Z$
$X\mathbb{I} = X$	$XX = \mathbb{I}$	$XY = \mathrm{i}Z$	$XZ = -\mathrm{i}Y$
$Y\mathbb{I} = Y$	$YX = -\mathrm{i}Z$	$YY = \mathbb{I}$	$YZ = \mathrm{i}X$
$Z\mathbb{I} = Z$	$ZX = \mathrm{i}Y$	$ZY = -\mathrm{i}X$	$ZZ = \mathbb{I}$

B Stabilizers and Pauli Matrices

Tab. 7 shows the states stabilized by each Pauli matrix, together with the density matrix of the stabilized state. Further, Tab. 8 shows the multiplication table for the Pauli matrices.