Static Analysis via Quantum Abstract Machines

An Operational Semantics Approach

MINGWEI ZHU and XINLU SHEN, University of Maryland, College Park, USA

1 INTRODUCTION

For decades, various models based on operational semantics, categorical semantics [Pagani et al. 2014] and game semantics [Clairambault et al. 2019][Clairambault and de Visme 2020] has been used to describe and specify the behavior of high-order program by quantum λ-calculus successfully. As the program grows larger, it is essential for programmers to have an approach to have a sound approximation of the result of the program ahead of the runtime so that we can check the correctness of a program and provide alternatives for optimizing the compiler for quantum program since we can know the shape of a program flow ahead of runtime. For classical programs, one way to perform sound approximation is to perform abstract interpretation over the program. As described by [Selinger and Valiron 2009], quantum λ-calculus combines quantum data such as quantum registers and special operations such as measure and qubit preparing with classical controls so that we can construct a program via a high-order perspective functionally. Owing to the classical control components residing in quantum λ-calculus, it is tempting to graft more classical control concepts and perform variants of classical control-flow analysis to achieve our goal. Inspired by Abstracting Abstract Machine Methodology [Van Horn and Might 2011], we proposed a solution to this problem by constructing several concrete quantum abstract machines gradually and abstracting a quantum abstract machine to reach the abstract domain govern the size of classical controls by bounding the abstract heap size in this work.

The paper which presents a derivation of an analyser by AAM approach is organized as follow. First, we include a brief introduction on an untyped variant of quantum lambda calclulus we use as a target language when performing abstract interpretation and few important properties we would like to enforece. Then, we describe a systematic progress of constructing a concrete abstract machine including QLM, QLCEK and QCESK*. After that, we present a $\widehat{\text{QCESK}}^*_t$ abstract machine where we use abstract timestamps and powerset construction to achieve the goal. In the last section, we talk about the limitation of the current machines and what a promising work in the future could be.

2 PREREQUISITES

2.1 Quantum Lambda Calculus

This section includes a brief summary of the literature *Quantum Lambda Calculus* [Selinger and Valiron 2009], which provides a definition of typed lambda calculus for quantum computation. In addition to the terms shared with classical call-by-value λ-calculus, terms supporting quantum data and controls were added into the language. For example, `new` is an operator used to simulate qubit preparation which means producing a quantum bit given a classical bit; `meas` is an operator for measurement. To express gate application from a high-order perspective, a set of constant unitary gates U which can representate specific matrix are also included into the language. These components enable us to define quantum programs. A simple fair coin can then be written as $\lambda * .\texttt{meas}(H\,(\texttt{new}\ 0))$, where $*$ is a unique 0-tuple representing "nothing" and H is the Hadamard gate.

Authors' address: Mingwei Zhu, mzhu1@terpmail.umd.edu; Xinlu Shen, xlshen@terpmail.umd.edu, University of Maryland, College Park, USA.

It is worth noting that in quantum λ-calculus, there is no actual classical data such as values of type nat in PCF settings. Truth and falseness—the foundation of logic—is represented as a syntactic sugar of $\mathtt{inj}_l *$ and $\mathtt{inj}_r *$. As a consequence, `if` is a syntactic form interpreted as an alias of pattern matching on $tinjl *$ and $tinjr *$. However, we can encode still encode natural numbers either in the style of Peano numbers or Church numerals.

Without a concrete semantics, one cannot interpret the meaning under the hood. In orignal work, both operational semantics which operate on a triary tuple $\langle q, l, M\rangle$, quantum closure, is used to develop a reduction semantics upon. The quantum closure treat states in a quantum program as the combination of quantum registers, which represent the all the qubits involved, linking functions, a mapping from variables to quantum registers, and terms standing for the form of the expression we are reducing. Due to the nature of meansurement and qubits, reductions on terms involving `meas` can results into 2 branches with probability p and $(1-p)$ so the probability is involved when generalizing such a step; let's say $\langle q, l, M\rangle \longmapsto_p \langle q', l', M'\rangle$ with probability p. Steps like this makes the multi-step reduction non-deterministic. However, if we consider all the possibility of the reduction, it is "deterministic" by techniques like superset construction.

Typing rules utilizing linearity was used to preserve the non-cloning theorem in [Selinger and Valiron 2009] and was seasoned later in [Pagani et al. 2014] so that the multiple access of quantum data is restricted but the one for the classical controls which are replicable is permitted.

2.2 Abstracting Abstract Machine Methodology

The abstracting abstract machine aims to provide a conservative and sound approximation for the behavior of programs. The approach by [Van Horn and Might 2011] relies on existing derivational techniques to transform high-level language semantics into low-level deterministic state-transition systems with potentially infinite state spaces. CEK machine is introduced as a state transition system that efficiently performs evaluation of a program. After having the idealized core of a realistic run-time system, a series of basic machine refactoring is then performed to obtain a non-deterministic state-transition system with a finite state space. In order to prepare for refactoring, a store component, which is a finite map from addresses to storable values, is introduced to machine states, and variable bindings and continuations are redirected through the store. Because the store is bound to a finite size, in order to avoid multiple values residing in a single store location, the store look-ups are required to be replaced by a non-deterministic choice among the multiple values.

The abstract interpreter is obtained by direct pointer optimization and structural abstraction that restrict the address space. By providing a computable analytic model that predicts intentional properties of the original machine, the derived machine computes a sound approximation of the machine, and thus forms an abstract interpretation of the machine and the high-level semantics. Such approach can be extended uniformly to richer language features.

3 ABSTRACT MACHINES

We will propose a 4-phase abstract machine construction on quantum λ-calculus from interpretation to abstraction.

3.1 QLM Abstract Machine

The QLM machine is our first attempt towards concrete quantum abstract machines where we directly adopt the definition of quantum closure and transform reduction rules into transitions among states in the mechine. Unlike deterministic abstract machines used to model simple λ-calculus, the `meas` operator doubles the size of a quantum closure since the possibility associated with current qubit determines two way how the post-measurement state looks like, which says steps related to measurement is non-deterministic. Since the reduction in QLM is non-deterministic,

$\langle \varsigma, p\rangle \longmapsto_{\mathrm{QLM}} \{\langle \varsigma', p'\rangle\}$	
$\langle\langle q, l, M\,N\rangle, p\rangle$	$\{\langle\langle q', l', M'\,N\rangle, p\rangle\}$
$\langle\langle q, l, V\,M\rangle, p\rangle$	$\{\langle\langle q', l', V\,M'\rangle, p\rangle\}$
$\langle\langle q, l, M \otimes N\rangle, p\rangle$	$\{\langle\langle q', l', M' \otimes N'\rangle, p\rangle\}$
$\langle\langle q, l, V \otimes M\rangle, p\rangle$	$\{\langle\langle q', l', V \otimes M'\rangle, p\rangle\}$
$\langle\langle q, l, \mathtt{inj}_l\, M\rangle, p\rangle$	$\{\langle\langle q', l', \mathtt{inj}_l\, M'\rangle, p\rangle\}$
$\langle\langle q, l, \mathtt{inj}_r\, M\rangle, p\rangle$	$\{\langle\langle q', l', \mathtt{inj}_r\, M'\rangle, p\rangle\}$
$\langle\langle q, l, \mathtt{let}\, x \otimes y = M\, \mathtt{in}\, N\rangle, p\rangle$	$\{\langle\langle q', l', \mathtt{let}\, x \otimes y = M'\, \mathtt{in}\, N\rangle, p\rangle\}$
$\langle\langle q, l, \mathtt{match}\, M\, \mathtt{with}\, (x : P \mid y : N)\rangle, p\rangle$	$\{\langle\langle q', l', \mathtt{match}\, M'\, \mathtt{with}\, (x : P \mid y : N)\rangle, p\rangle\}$
$\langle\langle q, l, (\lambda x.M)\, V\rangle, p\rangle$	$\{\langle\langle q, l, M\{V/x\}\rangle, p\rangle\}$
$\langle\langle q, l, \mathtt{let}\, x \otimes y = V \otimes W\, \mathtt{in}\, N\rangle, p\rangle$	$\{\langle\langle q, l, N\{V/x, W/y\}\rangle, p\rangle\}$
$\langle\langle q, l, \mathtt{match}\, (\mathtt{inj}_l V)\, \mathtt{with}\, (x : M \mid y : N)\rangle, p\rangle$	$\{\langle\langle q, l, M\{V/x\}\rangle, p\rangle\}$
$\langle\langle q, l, \mathtt{match}\, (\mathtt{inj}_r V)\, \mathtt{with}\, (x : M \mid y : N)\rangle, p\rangle$	$\{\langle\langle q, l, N\{V/y\}\rangle, p\rangle\}$
$\langle\langle q, l, \ \mathtt{letrec}\, f\, x = M\, \mathtt{in}\, N\rangle, p\rangle$	$\{\langle q, l, N\{(\lambda x.f\, x = M\, \mathtt{in}\, M)/f\}\rangle\}$
$\langle\langle q, l, U\,(x_1 \otimes \ldots \otimes x_k)\rangle, p\rangle$	$\{\langle\langle q', l, x_1 \otimes \ldots \otimes x_k\rangle, p\rangle\}$
$\langle\langle q, \varnothing, \mathtt{new}\, \mathtt{tt}\rangle, p\rangle$	$\{\langle\langle q \otimes \lvert 0\rangle, \{y \mapsto n+1\}, y\rangle, p\rangle\}$
$\langle\langle q, \varnothing, \mathtt{new}\, \mathtt{ff}\rangle, p\rangle$	$\{\langle\langle q \otimes \lvert 1\rangle, \{y \mapsto n+1\}, y\rangle, p\rangle\}$
$\langle\langle \alpha q_0 + \beta q_1, l\{x \mapsto i\}, \mathtt{meas}\, x\rangle, p\rangle$	$\{\langle\langle q_0', l, \mathtt{ff}\rangle, p \cdot \alpha^2 \neq 0\rangle, \langle\langle q_1', l, \mathtt{tt}\rangle, p \cdot \beta^2 \neq 0\rangle\}$

Fig. 1. Abstract Machine Reduction of QLM

we define the small-step transition as a relation (Figure 1) with type

$$\longmapsto_{\mathrm{QLM}} :: (\langle\langle Q, L, M\rangle, P\rangle, \mathcal{P}\,(\langle\langle Q, L, M\rangle, P\rangle)), \quad P = \mathbb{R} \cap [0, 1]$$

where each state and transition into several other states so that every possible execution is considered. In accord with this definition, we re-define the transitive-reflexive closure for QLM machines a là power set monad,

$$\begin{aligned}
\longmapsto\!\!\!\!\rightarrow_{\mathrm{QLM}} &:: (\mathcal{P}\,(\langle\langle Q, L, M\rangle, P\rangle), \mathcal{P}\,(\langle\langle Q, L, M\rangle, P\rangle))) \\
\longmapsto\!\!\!\!\rightarrow_{\mathrm{QLM}} &::= \left\{(\Sigma, \Sigma'') \mid \exists \Sigma',\ \left[\Sigma' = \left\{\varsigma' \mid \varsigma \longmapsto_{\mathrm{QLM}} \{\varsigma'\}, \varsigma \in \Sigma\right\}\right] \wedge \left[\Sigma' \longmapsto\!\!\!\!\rightarrow_{\mathrm{QLM}} \Sigma''\right]\right\} \\
&::= \{(\Sigma, \Sigma)\,.\}
\end{aligned}$$

Then, it is straightforward to define evaluation function,

$$eval_{\mathrm{QLM}}\,(M) = \left\{\varsigma \mid inj_{\mathrm{QLM}}(M) \longmapsto\!\!\!\!\rightarrow_{\mathrm{QLM}} \{\varsigma\}\right\}$$

where

$$inj_{\mathrm{QLM}}\,(M) = \langle\langle \varnothing, \varnothing, M\rangle, 1\rangle\,.$$

Limitation. This concrete abstract machine does a wonderful job in evaluating all the possible outcomes that could arise. However, there is a few limitations: first, the machines is in direct substitute style such that substitution are performed “on fly” while it is redundant to substitute each time an evaluation context is reduced. Moreover, since we are abstracting over an infinite state space, we cannot determeine the recursive structure of the program during runtime. If we are unable to split the execution process into eval, continue and apply, it is hard for us to construct such a program analyzer.

$\langle \varsigma, p\rangle$	$\longmapsto_{\text{QLCEK}} \{\langle \varsigma', p'\rangle\}$
$\langle\langle q, l, x, \rho, \kappa\rangle, p\rangle, x \in Dom(\rho)$	$\{\langle\langle q, l, v, \rho', \kappa\rangle, p\rangle\}, \rho(x) = (v, \rho')$
$\langle\langle q, l, x, \rho, \kappa\rangle, p\rangle, x \notin Dom(\rho)$	$\{\langle\langle q, l, l(x), \rho, \kappa\rangle, p\rangle\}$
$\langle\langle q, l, (e_1\, e_2), \rho, \kappa\rangle, p\rangle$	$\{\langle\langle q, l, e_1, \rho, \mathbf{ar}(e_2, \rho, \kappa)\rangle, p\rangle\}$
$\langle\langle q, l, \texttt{match}\, e_0\, \texttt{with}\, (x : e_1 \mid y : e_2)\rangle, \rho, \kappa, p\rangle$	$\{\langle\langle q, l, e_0, \rho, \mathbf{match}(x, e_1, y, e_2, \rho, \kappa)\rangle, p\rangle\}$
$\langle\langle q, l, \texttt{inj}_l\, v, \rho, \mathbf{match}(x, e_1, y, e_2, \rho, \kappa), p\rangle\rangle$	$\{\langle\langle q, l, e_1, \rho', \kappa\rangle, p\rangle\}, \rho' = \rho[x \mapsto (v, \rho)]$
$\langle\langle q, l, \texttt{inj}_r\, v, \rho, \mathbf{match}(x, e_1, y, e_2, \rho, \kappa), p\rangle\rangle$	$\{\langle\langle q, l, e_2, \rho', \kappa\rangle, p\rangle\}, \rho' = \rho[y \mapsto (v, \rho)]$
$\left\langle\left\langle \alpha q_{i_0} + \beta q_{i_1}, l[x \mapsto i], \texttt{meas}\, x, \rho, \kappa\right\rangle, p\right\rangle$	$\left\{\left\langle\left\langle \alpha q_{i_0} + \beta q_{i_1}, l[x \mapsto i], i, \rho, \mathbf{fn}(\texttt{meas}, \rho, \kappa)\right\rangle, p\right\rangle\right\}$
$\langle\langle q, l, v, \rho, \mathbf{ar}(e_2, \rho', \kappa)\rangle, p\rangle$	$\{\langle\langle q, l, e_2, \rho', \mathbf{fn}(e, \rho, \kappa)\rangle, p\rangle\}$
$\langle\langle q, l, v, \rho, \mathbf{fn}(\lambda x.e, \rho', \kappa)\rangle, p\rangle$	$\{\langle\langle q, l, e, \rho'[x \mapsto (v, \rho)], \kappa\rangle, p\rangle\}$
$\langle\langle q, l, v, \rho, \mathbf{fn}(\mu f.\lambda x.e, \rho', \kappa)\rangle, p\rangle$	$\{\langle\langle q, l, e, \rho'', \kappa\rangle, p\rangle\}$, $\rho'' = \rho'[f \mapsto (\mu f.\lambda x.e, \rho')][x \mapsto (v, \rho)]$
$\langle\langle q, l, v, \rho, \mathbf{fn}(U, \rho', \kappa)\rangle, p\rangle$	$\{\langle\langle q, l, [\![U\, v]\!], \rho, \kappa\rangle, p\rangle\}$
$\langle\langle q, l, \texttt{ff}, \rho, \mathbf{fn}(\texttt{new}, \rho', \kappa)\rangle, p\rangle$	$\{\langle\langle q \otimes \lvert 0\rangle, l[x \mapsto i], i, \rho, \kappa\rangle, p\rangle\}$
$\left\langle\left\langle \alpha q_{i_0} + \beta q_{i_1}, l[x \mapsto i], i, \rho, \mathbf{fn}(\texttt{meas}, \rho, \kappa)\right\rangle, p\right\rangle$	$\left\{\left\langle\left\langle q'_{i_0}, l, \texttt{ff}, \rho, \kappa\right\rangle, p \cdot \alpha^2\right\rangle, \left\langle\left\langle q'_{i_1}, l, \texttt{tt}, \rho, \kappa\right\rangle, p \cdot \beta^2\right\rangle\right\}$
$\langle\langle q, l, \texttt{let}\, x \otimes y = e_1\, \texttt{in}\, e_2, \rho, \kappa\rangle, p\rangle$	$\{\langle\langle q, l, e_1, \rho, \mathbf{let}(x, y, e_2, \rho, \kappa)\rangle, p\rangle\}$
$\langle\langle q, l, v_1 \otimes v_2, \rho, \mathbf{let}((x, y, e_2, \rho, c))\rangle, p\rangle$	$\{\langle\langle q, l, e, \rho[x \mapsto (v_1, \rho)][y \mapsto (v_2, \rho)], \kappa\rangle, p\rangle\}$
$\langle\langle q, l, \texttt{letrec}\, f\, x = e\, \texttt{in}\, e_2, \rho, \kappa\rangle, p\rangle$	$\{\langle\langle q, l, e_2, \rho[f \mapsto (\mu f.\lambda x.e, \rho)], \kappa\rangle, p\rangle\}$

Fig. 2. Transitions of concrete QLCEK machine

3.2 QLCEK **Abstract Machine**

In order to address the drawback in QLM machine, our approach, drawing inspiration from CEK machine [Felleisen and Friedman 1987], is to make a detour, QLCEK (Figure 2), by adding more classical controls in reduction. Specifically, substitutions are accomplished by using environments and closures, and evaluation contexts are extracted and modeled by continuations.

The specification is given as

$$
\begin{array}{rlcl}
\varsigma \in & \Sigma & :: & QRegs \times Link \times Exp \times Env \times Kont \\
\rho \in & Env & :: & Var \rightarrow_{\text{fin}} Val \times Env + Index \\
q \in & QRegs & :: & [qubit] \\
l \in & Link & :: & Index \longrightarrow QRegs \\
v \in & Val & = & \ldots^{1} + \mu f.\lambda x.e \\
\kappa \in & Kont & = & \mathbf{mt} \mid \mathbf{fn}(Exp, Env, Kont) \mid \mathbf{ar}(Exp, Env, Kont) \\
 & & \mid & \mathbf{let}\otimes(Var, Var, Exp, Env, Kont) \\
 & & \mid & \mathbf{match}(Var, Exp, Var, Env, Kont).
\end{array}
$$

Limitation. As the second attempt, we add continuation and environment into this machine QLCEK which allow us to abstract this program from these two perspectives. However, given that continuation and environemnt closure here are still recursive structures, if we perform a pairwise transformation α from concrete domain to abstract domain. The abstract space is still infinite. What's worse, QLCEK machine cannot enforce non-cloning property, which says that if a program is written in such way that a qubit is duplicated and used twice the machine will run nicely rather than getting stuck.

$\langle \varsigma, p\rangle \longmapsto_{\text{QCESK}^*} \{\langle \varsigma', p'\rangle\}$ where $\kappa = \sigma(a), b \notin dom(\sigma)$	
$\langle\langle q, x, \rho, \sigma, a\rangle, p\rangle$	$\{\langle\langle q, v, \rho', \sigma, a\rangle, p\rangle\}, \sigma(\rho(x)) = (v, \rho')$
$\langle\langle q, x, \rho[x \mapsto s], \sigma, a\rangle, p\rangle, \sigma(s) = i$	$\{\langle\langle q, i, \rho, \sigma, a\rangle, p\rangle\}$
$\langle\langle q, (e_1\, e_2), \rho, \sigma, a\rangle, p\rangle$	$\{\langle\langle q, e_1, \rho, \sigma[b \mapsto \mathbf{ar}(e_2, \rho, a)], b\rangle, p\rangle\}$
$\langle\langle q, \mathtt{match}\, e_0\, \mathtt{with}\, (x : e_1 \mid y : e_2)\rangle, \rho, \sigma, a, p\rangle$	$\{\langle\langle q, e_0, \rho, \sigma[b \mapsto \mathbf{match}(x, e_1, y, e_2, \rho, \sigma, a)], b\rangle, p\rangle\}$
$\left\langle\left\langle \alpha q_{i_0} + \beta q_{i_1}, \mathtt{meas}\, x, \rho, \sigma, a\right\rangle, p\right\rangle$	$\left\{\left\langle\left\langle \alpha q_{i_0} + \beta q_{i_1}, x, \rho, \sigma[b \mapsto \mathbf{fn}(\mathtt{meas}, \rho', c)], b\right\rangle, p\right\rangle\right\}$
$\langle\langle q, v, \rho, \sigma, a\rangle, p\rangle,$	
$\kappa = \mathbf{ar}(e_2, \rho', c)$	$\{\langle\langle q, e_2, \rho', \sigma[b \mapsto \mathbf{fn}(v, \rho, c)], b\rangle, p\rangle\}$
$v = \mathtt{inj}_l\, v_0, \kappa = \mathbf{match}(x, e_1, y, e_2, \rho', c)$	$\{\langle\langle q, e_1, \rho'[x \mapsto b], \sigma[b \mapsto (v_0, \rho')], c\rangle, p\rangle\}$
$v = \mathtt{inj}_r\, v_0, \kappa = \mathbf{match}(x, e_1, y, e_2, \rho', c)$	$\{\langle\langle q, e_2, \rho'[y \mapsto b], \sigma[b \mapsto (v_0, \rho')], c\rangle, p\rangle\}$
$\kappa = \mathbf{fn}(\lambda x.e, \rho', c)$	$\{\langle\langle q, e, \rho'[x \mapsto b], \sigma[b \mapsto (v, \rho')], c\rangle, p\rangle\}$
$\kappa = \mathbf{fn}(\mu f.\lambda x.e, \rho', c)$	$\{\langle\langle q, e, \rho'[x \mapsto b][f \mapsto d], \sigma', c\rangle, p\rangle\}$
	$\sigma' = \sigma[b \mapsto (v, \rho')][d \mapsto \mu f.\lambda x.e, \rho']$
$v = \mathtt{ff}, \kappa = \mathbf{fn}(\mathtt{new}, \rho', c)$	$\{\langle\langle q \otimes \lvert 0\rangle, x, \rho[x \mapsto b], \sigma[b \mapsto n], c\rangle, p\rangle\}$
$q = \alpha q_{i_0} + \beta q_{i_1}, \kappa = \mathbf{fn}(\mathtt{meas}, \rho', c)$	$\left\{ \begin{array}{l} \left\langle\left\langle [[q'_{i_0}]], \mathtt{ff}, \rho'', \sigma', c\right\rangle, p \cdot \alpha^2\right\rangle \\ \left\langle\left\langle [[q'_{i_1}]], \mathtt{tt}, \rho'', \sigma', c\right\rangle, p \cdot \beta^2\right\rangle \end{array} \right\} \begin{array}{l} \rho'' = \rho' \setminus [x \mapsto b] \\ \sigma' = \sigma \setminus [b \mapsto v] \end{array}$

Fig. 3. Concrete QCESK* abstract machine

3.3 QCESK* Abstract Machine

In order to resolve the problems in the previous machine, we come up with quantum version of CESK* machine [Van Horn and Might 2011]. In QCESK* (Figure 3), we unhinged the recursive structure under the hood of continuation and environment by heap-allocating the continuation and closures with a pointer which resembles to a pushdown abstract machine.

Formally, we alter the definition of QLCEK as

$$\begin{array}{rlcl} \varsigma \in & \Sigma & :: & \mathit{QRegs} \times \mathit{Exp} \times \mathit{Env} \times \mathit{Store} \times \mathit{Addr} \\ \rho \in & \mathit{Env} & :: & \mathit{Var} \rightharpoonup_{\text{fin}} \mathit{Addr} \\ \sigma \in & \mathit{Store} & :: & \mathit{Addr} \rightharpoonup_{\text{fin}} \mathit{Storable} \\ s \in & \mathit{Storable} & :: & \mathit{Val} \times \mathit{Env} + \mathit{Kont} + \mathit{Index} \times \mathit{Tag} \\ \kappa \in & \mathit{Cont} & & \mathbf{fn}(\mathit{Exp}, \mathit{Env}, \mathit{Addr}) \end{array}$$

to construct a concrete QCESK* machine. Now, we have successfully constructed a concrete machine which is observationally equivalent to QLM machine, $eval_{\text{QCESK}^*}(M) \triangleq eval_{\text{QLM}}(M)$. Rather than introducting an infinite biproduct to allow folding and unfolding of a n-fold tensor power into CPM category to accomplish the expressiveness of recurive function and distinguish classical and quantum linear components, we simply use a μ as a fix-point combinator and the concept of ownership transition to enforce linearity.

The evaluation function is defined as,

$$eval_{\text{QCESK}^*}(M) = \left\{\varsigma \mid inj_{\text{QCESK}^*}(M) \longmapsto\!\!\!\!\rightarrow_{\text{QCESK}^*} \{\langle \varsigma, p\rangle\}\right\}$$

where

$$inj_{\text{QCESK}^*}(M) = \langle\langle \emptyset, M, \emptyset, \emptyset, \mathbf{mt}\rangle, 1\rangle\,.$$

$$\hat{\varsigma} \longmapsto_{\widehat{\mathrm{QCESK}^*_{\mathrm{t}}}} \hat{\varsigma}' \text{ where } \langle \kappa, p' \rangle \in \hat{\sigma}(a), b = \widehat{alloc}(\hat{\varsigma}, \kappa), u = \widehat{tick}(t, \kappa)$$

$$\begin{array}{rl}
\langle Q, x, \rho, \hat{\sigma}, a, t, p \rangle,\ \langle (v, \rho'), p' \rangle \in \hat{\sigma}(\rho(x)) & \langle Q, v, \rho', \hat{\sigma}, a, u, p' \rangle \\
\langle Q, x, \rho[x \mapsto s], \hat{\sigma}, a, t, p \rangle, \hat{\sigma}(s) = \langle i, p \rangle & \langle Q, i, \rho, \hat{\sigma}, a, u, p \rangle \\
\langle Q, (e_1\, e_2), \rho, \hat{\sigma}, a, t, p \rangle & \big\langle Q, e_1, \rho, \hat{\sigma} \sqcup {}^{2} [b \mapsto \langle \mathbf{ar}(e_2, \rho, a), p \rangle], b, u, p \big\rangle \\
\langle Q, \mathtt{match}\, e_0\, \mathtt{with}\, (x : e_1 \mid y : e_2), \rho, \hat{\sigma}, a, t, p \rangle & \langle Q, e_0, \rho, \hat{\sigma} \sqcup [b \mapsto \langle \mathbf{match}(x, e_1, y, e_2, \rho, \sigma, a), p \rangle], b, u, p \rangle \\
\langle Q, \mathtt{meas}\, x, \rho, \hat{\sigma}, a, t, p \rangle & \langle Q, x, \rho, \hat{\sigma} \sqcup [b \mapsto \langle \mathbf{fn}(\mathtt{meas}, \rho', c), p \rangle], b, u, p \rangle \\
\langle Q, v, \rho, \hat{\sigma}, a, t, p \rangle, & \\
\kappa = \mathbf{ar}(e_2, \rho', c) & \langle Q, e_2, \rho', \hat{\sigma} \sqcup [b \mapsto \langle \mathbf{fn}(v, \rho, c), p \rangle], b, u, p \rangle \\
v = \mathtt{inj}_l\, v_0, \kappa = \mathbf{match}(x, e_1, y, e_2, \rho', c) & \langle Q, e_1, \rho'[x \mapsto b], \hat{\sigma} \sqcup [b \mapsto (v_0, \rho')], c, u, p \rangle \\
v = \mathtt{inj}_r\, v_0, \kappa = \mathbf{match}(x, e_1, y, e_2, \rho', c) & \langle Q, e_2, \rho'[y \mapsto b], \hat{\sigma} \sqcup [b \mapsto (v_0, \rho')], c, u, p \rangle \\
\kappa = \mathbf{fn}(\lambda x.e, \rho', c) & \langle q, e, \rho'[x \mapsto b], \hat{\sigma} \sqcup [b \mapsto \langle (v, \rho'), p \rangle], c, u, p \rangle \\
\kappa = \mathbf{fn}(\mu f.\lambda x.e, \rho', c) & \langle q, e, \rho'[x \mapsto b][f \mapsto d], \hat{\sigma}', c, u, p \rangle, \\
 & \hat{\sigma}' = \hat{\sigma} \sqcup [b \mapsto \langle (v, \rho'), p \rangle] \sqcup [d \mapsto \langle (\mu f.\lambda x.e, \rho'), p \rangle] \\
v = \mathtt{ff}, \kappa = \mathbf{fn}(\mathtt{new}, \rho', c) & \langle q \otimes |0\rangle, x, \rho[x \mapsto b], \hat{\sigma} \sqcup [b \mapsto \langle n, p \rangle], c, u, p \rangle \\
q = \alpha q_{i_0} + \beta q_{i_1}, \kappa = \mathbf{fn}(\mathtt{meas}, \rho', c) & \langle Q, *, \rho'', \sigma', c, u, p \rangle, \\
 & \hat{\sigma}'' = \hat{\sigma} \sqcup \big\langle b \mapsto \big\{ (\mathbf{meas}_{\mathtt{t}}\, (q'_{i0}, c), \alpha^2), (\mathbf{meas}_{\mathtt{f}}\, (q'_{i1}, c), \beta^2) \big\} \big\rangle, \\
 & \hat{\sigma}' = \hat{\sigma}'' \setminus [b \mapsto v], \\
 & \rho'' = \rho' \setminus [x \mapsto b] \\
\kappa = \mathbf{meas}_{\mathtt{tt}}\, (q_0, c) & \langle q_0, , \mathtt{tt}, \rho, \sigma, c, u, (p \cdot p') \rangle \\
\kappa = \mathbf{meas}_{\mathtt{ff}}\, (q_1, c) & \langle q_1, , \mathtt{ff}, \rho, \sigma, c, u, (p \cdot p') \rangle
\end{array}$$

Fig. 4. Abstract timestamp $\widehat{\mathrm{QCESK}^*_{\mathrm{t}}}$ machine

3.4 $\widehat{\mathrm{QCESK}^*_{\mathrm{t}}}$ Abstract Machine

Now, we have arrived at the final step of our machine derivation, $\widehat{\mathrm{QCESK}^*_{\mathrm{t}}}$. This is achieved by abstracting the timestamp version of our concrete abstract machine QCESK* to acquire a non-deterministic abstract machine with a finite-state space via powerset construction over Storable values and quantum registers.

Here, we gives out the seasoned definition for Store in $\widehat{\mathrm{QCESK}^*_{\mathrm{t}}}$.

$$\begin{array}{rlcl}
\hat{\varsigma} \in & \hat{\Sigma} & :: & \mathcal{P}\,(QRegs) \times Link \times (Exp \times P) \times Env \times Kont \times P \\
\sigma \in & Store & :: & Addr \rightarrow_{\text{fin}} \mathcal{P}\,(Storable \times P) \\
l \in & Link & :: & Index \longrightarrow QRegs \\
t, u \in & Time & :: & \mathbb{N}
\end{array}$$

4 CONCLUSION

With the attempt to provide a static analysis on quantum programs, we first presented the construction of several quantum abstract machines for quantum lambda calculus using the store-allocation strategy for continuation, where restrictions on quantum data (e.g. the no-cloning theorem) are satisfied, as an alternative to the model of completely positive maps over finite dimension. We then abstracted the time-stamped QCESK* machine, which is allowed to be nondeterministic, by constructing a set of storable values to bound the state space to be finite and adding an extra layer of continuation for probabilistic branches of outcomes.

5 A LOOK FORWARD

5.1 Soundness and Decidability

least fix point The introduction of quantum computation does give us computing power which cannot be achieved by classical computers, while its full potential is still under investigation. Uncertainty and undecidability remain unsolved under such field. Context and case analysis are required for the precision and soundness of our abstract interpretation.

5.2 Performance

exponential

5.3 Game Semantics and Abstract Machines

game semantics, full abstraction, game semantics abstract machine

REFERENCES

Pierre Clairambault and Marc de Visme. 2020. Full abstraction for the quantum lambda-calculus. *Proceedings of the ACM on Programming Languages* 4, POPL (Jan. 2020), 1–28. https://doi.org/10.1145/3371131

Pierre Clairambault, Marc De Visme, and Glynn Winskel. 2019. Game semantics for quantum programming. *Proceedings of the ACM on Programming Languages* 3, POPL (Jan. 2019), 1–29. https://doi.org/10.1145/3290345

Mattias Felleisen and D. P. Friedman. 1987. A calculus for assignments in higher-order languages. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages (POPL '87)*. Association for Computing Machinery, Munich, West Germany, 314. https://doi.org/10.1145/41625.41654

Michele Pagani, Peter Selinger, and Benoît Valiron. 2014. Applying quantitative semantics to higher-order quantum computing. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '14)*. Association for Computing Machinery, San Diego, California, USA, 647–658. https://doi.org/10.1145/2535838.2535879

Peter Selinger and Benoît Valiron. 2009. Quantum Lambda Calculus. In *Semantic Techniques in Quantum Computation*, Simon Gay and Ian Mackie (Eds.). Cambridge University Press, Cambridge, 135–172. https://doi.org/10.1017/CBO9781139193313.005

David Van Horn and Matthew Might. 2011. *Abstracting abstract machines*. Vol. 54. 100 pages. https://doi.org/10.1145/1995376.1995399