SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

December 2019, Vol. 62 222501:1–222501:8
https://doi.org/10.1007/s11432-018-9847-0

Implementing termination analysis on quantum programming

Shusen LIU[1,2,3*], Kan HE[4*] & Runyao DUAN[2]

[1]*School of Data and Computer Science, Sun Yat-sen University, Guangzhou* 510006*, China;*
[2]*Institute for Quantum Computing, Baidu, Beijing* 100193*, China;*
[3]*Institute for Advanced Study, Tsinghua University, Beijing* 100084*, China;*
[4]*College of Mathematics & College of Information and Computer Science, Taiyuan University of Technology, Taiyuan* 030024*, China*

Received 1 June 2018/Revised 24 December 2018/Accepted 12 March 2019/Published online 11 November 2019

Abstract Termination analysis is an essential part in programming. Especially quantum programming concerning measurement, entanglement and even superposition are the foundations of bizarre behaviours in quantum programs. In this paper, we analyse and extend the theoretical theorems on termination analysis proposed by Ying et al. into computational theorems and algorithms. The new algorithm without the Jordan decomposition process has a significant acceleration with polynomial complexity both on terminating and almost-surely terminating programs. Moreover, the least upper bound of termination programs steps is studied and utilized to output the substituted matrix representation of quantum programs. We also implement four groups of experiments to illustrate the advantages of the new algorithm in case of processing a simplified quantum walk example comparing with the original counterpart.

Keywords termination analysis, quantum while-language, quantum programming, quantum experiment, terminating steps

Citation Liu S S, He K, Duan R Y. Implementing termination analysis on quantum programming. Sci China Inf Sci, 2019, 62(12): 222501, https://doi.org/10.1007/s11432-018-9847-0

1 Introduction

Termination analysis is a crucial task in classical programming language and a necessary part of modern programming environment. As with the software quality surging, more relative tools are required for debugging the programs. A very beginning tool called 'Syntox' was mentioned by Bourdoncle [1]. It turned on the abstract debugging of the Pascal language by using scalar variables of programs. In 2006, Cook et al. [2], as a team of Microsoft, developed an automatic termination tool which provided a capacity for large program fragments. After a decade, also from Microsoft, Brockschmidt et al. [3] proposed a new open-source tool T2 which supported automatic temporal-logic proving techniques and handled a general class of user-provided liveness and safety properties.

Considering that quantum programming is quite counter-intuitive due to the measurement, superposition, and entanglement, the termination of quantum programming receives more attention than its classical counterpart. For any practical quantum system with the classical loop structure, the information of termination is always useful since the quantum interpreter is not able to guarantee a finite number of steps within the quantum programs. Moreover, Ying [4] formally introduced a quantum language with a loop structure namely quantum **while**-language. The operational semantics, denotational semantics

* Corresponding author (email: shusen88.liu@gmail.com, hk19830310@163.com)

and related characterization were systematically studied. Now, the quantum **while**-language had become a powerful language to describe some complicated quantum algorithms such as quantum walk. It was simple to be used and made quantum programming relatively easier. Meanwhile, the termination analysis problem was also introduced. Termination of quantum programs with unitary transformation in quantum programming was firstly studied by Ying and Feng [5] where the Jordan decomposition process was employed to develop the analysis algorithm. In 2013, they extended the analysis method to the termination of quantum programs with super-operator transformation [6]. Moreover, the termination of non-deterministic and concurrent quantum programs was carefully studied by Li et al. [7] and Yu and Ying [8] as a problem of quantum Markov systems. Recently, by using realisability and synthesis of linear ranking super-martingale (LRSMs), Li and Ying [9] examined the termination problems of quantum programs and reduced them into semi-definite programming (SDP) problems.

In this paper, we first review the definitions and theorems of quantum termination followed by two algorithms for analysing. One is extended directly from the propositions in [6] into computable conditions while the other is an improved algorithm without the Jordan decomposition to reach high reliable results and significant acceleration. Lastly, we provide an example – Qloop – to display the efficiency of both algorithms. It also should be noted that the algorithms we proposed are only fit for finite-dimensional quantum programs. The infinite scenario can be found in [9].

2 Definitions and theorems

A general quantum loop can be shortened in the form,

$$\textbf{while}(M[\bar{q}] \in X)\{\bar{q} = \mathcal{E}(\bar{q});\},$$

where $\bar{q}$ is a quantum variable, M is a measurement that satisfies $M_0^\dagger M_0 + M_1^\dagger M_1 = I$, where $\dagger$ indicates the conjugate transpose operation. X is a set of measurement results, $\mathcal{E}$ is a complete positive and trace-preserving (CPTP) map and a denotation for a **while**'s sub-program, and $\mathcal{E}(\cdot) = \sum_i E_i(\cdot)E_i^\dagger$. We also define the super-operator $\mathcal{E}_i(\rho) = M_i \rho M_i^\dagger$ for any density operator ρ, and $i = 0, 1$.

We denote $\mathcal{G}(\rho) = \sum_i E_i M_1 \rho M_1^\dagger E_i^\dagger$, G in the text below is the matrix representation of $\mathcal{G}$, that is $G = \sum_i (E_i M_1) \otimes (E_i M_1)^*$, where $*$ indicates the transpose operation. $|\Phi\rangle$ is the maximally entangled state.

Definition 1. (I) We say that the program terminates if the non-termination probability in the nth step is

$$p_n^{\mathrm{NT}} = \mathrm{tr}([\mathcal{E}_1 \circ (\mathcal{E} \circ \mathcal{E}_1)^{n-1}](\rho_0)) = 0$$

for some positive integer n.

(II) We say that a program almost-surely terminates if $\lim_{n\to\infty} p_n^{\mathrm{NT}} = 0$, where p_n^{NT} is as per the finite termination definition.

Prior to outlining the full results, some elementary results are provided as follows. Lemma 1 is obvious.

Lemma 1. (I) The program terminates if and only if $G^n(\rho_0 \otimes I)\,|\Phi\rangle = 0$ for some integer $n \geqslant 0$;

(II) The program almost-surely terminates if and only if $\lim_{n\to\infty} G^n(\rho_0 \otimes I)\,|\Phi\rangle = 0$.

Then we could have Lemma 2 in termination analysis [6] which only concerns the program operations other than the inputs.

Lemma 2. (I) The program scheme is terminating if and only if $G^n\,|\Phi\rangle = 0$ for some integer $n \geqslant 0$;

(II) The program scheme is almost-surely terminating if $\lim_{n\to\infty} G^n\,|\Phi\rangle = 0$.

However, it is not easy to check whether a quantum program scheme terminates or almost-surely terminates with the above definition or lemmas. But, by splitting the vectors, these assertions can be converted into a computable proposition that assesses the correct termination type through some operational algorithms (see Section 3).

Recall that G is the matrix representation of the quantum program scheme $\mathcal{G}$. Therefore, the eigenvalues of G will have norms of 1 or less. Denoted as H_λ, the eigensubspace of G's eigenvalue λ gives

the space decomposition $H = H_0 \oplus H_{(0,1)} \oplus H_1$, where $H_0 = \bigvee_{|\lambda|=0} H_\lambda$, $H_{(0,1)} = \bigvee_{|\lambda|\in(0,1)} H_\lambda$ and $H_1 = \bigvee_{|\lambda|=1} H_\lambda$.

Theorem 1. Supposing the Jordan decomposition of G is $G = SJ(G)S^{-1}$ with an invertible S and

$$S^{-1}|\Phi\rangle = \begin{pmatrix} |u\rangle \\ |v\rangle \\ |w\rangle \end{pmatrix}$$

based on the space decomposition $H = H_0 \oplus H_{(0,1)} \oplus H_1$, the following statements hold true:

(I) If $|v\rangle$ and $|w\rangle$ are both 0, the program scheme G is terminating;

(II) If $|w\rangle$ is nonzero, the program scheme G is non-terminating;

(III) The program scheme G is almost-surely terminating if and only if $|w\rangle$ is 0.

Proof. The proofing technique is similar to that of Proposition 5.1 in [6].

Suppose $G = SJ(G)S^{-1}$ where $J(G) = \mathrm{diag}(J_{k_1}(\lambda_1), J_{k_2}(\lambda_2), \ldots, J_{k_l}(\lambda_l))$ with J_{k_s} being a $k_s \times k_s$-Jordan block of eigenvalue λ_s $(1 \leqslant s \leqslant l)$ [10]. All of the eigenvalues $0 \leqslant |\lambda_i| \leqslant 1$ [11], without loss of generality, we can assume $\lambda_1 = \cdots = \lambda_s = 0$, $0 < |\lambda_{s+1}| \leqslant \cdots \leqslant |\lambda_t| < 1$ and $|\lambda_{t+1}| = \cdots = |\lambda_l| = 1$. As $G^n = SJ(G)^n S^{-1}$, $G^n|\Phi\rangle = SJ(G)^n S^{-1}|\Phi\rangle$. Based on the space decomposition $H = H_0 \oplus H_{(0,1)} \oplus H_1$, we write

$$J(G) = \begin{pmatrix} A & 0 & 0 \\ 0 & B & 0 \\ 0 & 0 & C \end{pmatrix} \quad \text{and} \quad S^{-1}|\Phi\rangle = \begin{pmatrix} |u\rangle \\ |v\rangle \\ |w\rangle \end{pmatrix},$$

where

$$A = \mathrm{diag}(J_{k_1}(\lambda_1), \ldots, J_{k_s}(\lambda_s)), \quad B = \mathrm{diag}(J_{k_{s+1}}(\lambda_{s+1}), \ldots, J_{k_t}(\lambda_t)),$$
$$C = \mathrm{diag}(J_{k_{t+1}}(\lambda_{t+1}), \ldots, J_{k_l}(\lambda_l)),$$

$|u\rangle$ is an s-dimensional vector, $|v\rangle$ is a $(t-s)$-dimensional vector, $|w\rangle$ is a $(l-t)$-dimensional vector and $S^{-1}|\Phi\rangle$ is a combination of $|u\rangle$, $|v\rangle$ and $|w\rangle$. Following that, $J(G)^n S^{-1}|\Phi\rangle$ can be written as

$$J(G)^n S^{-1}|\Phi\rangle = \begin{pmatrix} A^n|u\rangle \\ B^n|v\rangle \\ C^n|w\rangle \end{pmatrix}. \tag{1}$$

Let us first check (I). Noting that $\lambda_1, \ldots, \lambda_s = 0$ and $\lambda_{s+1}, \ldots, \lambda_l \neq 0$, $J_{k_{s+1}}(\lambda_{s+1}), \ldots, J_{k_l}(\lambda_l)$ are non-singular in (1), if $|v\rangle$ and $|w\rangle$ are both 0, then according to (1), we have

$$J(G)^n S^{-1}|\Phi\rangle = \begin{pmatrix} A^n|u\rangle \\ 0 \\ 0 \end{pmatrix}.$$

Now, with an n greater than the maximal size of the Jordan blocks $J_{k_j}(0)$, $j = 1, 2, \ldots, s$, $A^n = 0$. It follows that $J(G)^n S^{-1}|\Phi\rangle = 0$. Thus $G^n|\Phi\rangle = 0$, and G is therefore terminating. (I) is proved. Moreover, if $|w\rangle \neq 0$, then $J(G)^n S^{-1}|\Phi\rangle \neq 0$ for any n since C is non-singular. So $G^n|\Phi\rangle \neq 0$ for any n, and (II) is proved.

Next, let us turn to prove (III). By noting that $\lim_{n\to\infty} A^n = 0$ and $\lim_{n\to\infty} B^n = 0$, we can conclude that $|w\rangle$ is 0, if and only if, $C^n|w\rangle = 0$ since C is non-singular. It follows that $|w\rangle$ is 0 if and only if,

$$\lim_{n\to\infty} J(G)^n S^{-1}|\Phi\rangle = \begin{pmatrix} A^n|u\rangle \\ B^n|v\rangle \\ C^n|w\rangle \end{pmatrix} = 0.$$

Therefore, (III) holds true.

The following proposition concerns the output matrix representation of terminating program scheme and almost-surely terminating program scheme which can be used for nested and concurrent program analysis.

Proposition 1. Supposing that G is defined in Theorem 1 and $N_0 = M_0 \otimes M_0^*$.

(I) If the program scheme G terminates after k-steps, then the matrix representation of the program scheme becomes $G' = \sum_{n=0}^{k-1} N_0 G^n$.

(II) If the program scheme G almost-surely terminates, then the matrix representation of the program scheme becomes $G' = \sum_{n=0}^{\infty} N_0 G^n = N_0 (I - G)^{-1}$.

Concluding Proposition 5.1 in [6], we can draw two other sufficient conditions.

Proposition 2. (I) $G^d |\Phi\rangle = 0$, where d is the rank of G, if and only if the program scheme G terminates.

(II) If $|\Phi\rangle$ is orthogonal to all eigenvectors of $G^\dagger$ corresponding to the eigenvalues in module 1, then the program scheme G almost-surely terminates.

Theorem 2 is the least upper bound of the terminating steps, i.e., for any terminating program, when the executing step $n_{\text{any}} \geqslant n_{\text{sup}}$, the program must terminate. Following Proposition 1, it determines the fixed output matrix representation of a program.

Theorem 2. If the program scheme G terminates, then the least upper bound of terminating steps

$$n_{\text{sup}} = \max_{J_{k_1},\ldots,J_{k_s}} \min\{k'_j, n'_j\},$$

where k'_j is the size of the jth Jordan block J_{k_j} of G with $0 \leqslant j \leqslant s$, and n'_j is the greatest index of non-zero elements in the vector block corresponding to the jth Jordan block of G.

Proof. Let $B_1 = J_{k_1}(0)$, …, $B_s = J_{k_s}(0)$ and $S^{-1}|\Phi\rangle = |y\rangle = (|y_1\rangle, \ldots, |y_s\rangle)^{\mathrm{T}}$. Therefore, G is terminating if and only if

$$B|y\rangle = \begin{pmatrix} B_1 & & \\ & \ddots & \\ & & B_s \end{pmatrix} \begin{pmatrix} |y_1\rangle \\ \vdots \\ |y_s\rangle \end{pmatrix} = \begin{pmatrix} B_1|y_1\rangle \\ \vdots \\ B_s|y_s\rangle \end{pmatrix}.$$

Further, if the program scheme is terminating, then there is an n, such that

$$B^n|y\rangle = \begin{pmatrix} B_1^n|y_1\rangle \\ \vdots \\ B_s^n|y_s\rangle \end{pmatrix} = 0. \tag{2}$$

To complete the proof, we only need to show that for each block under the condition $\min\{k'_j, n'_j\}$, every block must ensure $B_j^{n'_j}|y_j\rangle = 0$. $n_{\text{sup}} = \max \min\{k'_j, n'_j\}$ is enough to ensure all blocks are 0, i.e., $B|y\rangle = 0$.

Now, with (2), $B_1^n|y_1\rangle = 0$, …, $B_s^n|y_s\rangle = 0$ for each block. It is clear that after k'_j times with the the size of k'_j, $B_j^{k'_j} = 0$ where $0 \leqslant j \leqslant s$ as follows:

$$B_j = \begin{pmatrix} 0 & 1 & & & \\ & 0 & 1 & & \\ & & \ddots & & \\ & & & 0 & 1 \\ & & & & 0 \end{pmatrix}, \quad B_j^{k'_j-1} = \begin{pmatrix} 0 & 0 & & 0 & 1 \\ & 0 & 0 & & 0 \\ & & \ddots & & \\ & & & 0 & 0 \\ & & & & 0 \end{pmatrix}, \quad B_j^{k'_j} = \begin{pmatrix} 0 & 0 & & 0 & 0 \\ & 0 & 0 & & 0 \\ & & \ddots & & \\ & & & 0 & 0 \\ & & & & 0 \end{pmatrix}.$$

However, supposing $|y\rangle = (y_1, y_2, \ldots, y_j)^{\mathrm{T}}$ for each block $J_{k_j}(0)$, we have

$$B_j^{n'_j}|y_j\rangle = B_j^{n'_j-1}B_j|y_j\rangle = B_j^{n'_j-1}\begin{pmatrix} 0 & 1 & & & \\ & 0 & 1 & & \\ & & \ddots & & \\ & & & 0 & 1 \\ & & & & 0 \end{pmatrix}\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_j \end{pmatrix} = B_j^{n'_j-1}\begin{pmatrix} y_2 \\ y_3 \\ \vdots \\ y_j \\ 0 \end{pmatrix}$$
$$= B_j^{n'_j-1}\left|y_j^1\right\rangle = B_j^{n'_j-2}\left|y_j^2\right\rangle = \cdots = \left|y_j^n\right\rangle.$$

If the first n'_j number of $|y_j\rangle = (y_1, y_2, \ldots, y_j)^{\mathrm{T}}$ is not 0, then the first $n'_j - 1$ number of $\left|y_j^1\right\rangle = (y_2, y_3, \ldots, y_j)^{\mathrm{T}}$ is not 0. After n'_j times, $|y_j^{n'_j}\rangle = 0$.

Thus we can conclude that, for each block under the condition $\min\{k'_j, n'_j\}$, it ensures every block $B_j^{n'_j}|y_j\rangle = 0$. $n_{\sup} = \max\min\{k'_j, n'_j\}$ is enough to ensure all blocks are 0, i.e., $B|y\rangle = 0$. This completes the proof.

3 Algorithms for termination analysis

In Section 2, we outline several useful theorems and propositions for termination analysis. Here, we provide two computational algorithms for termination analysis. Algorithm 1 corresponds to the Jordan decomposition process and is only considered as efficient complicity from a theoretical perspective. Algorithm 2 is more flexible and efficient theoretically and practically.

Algorithm 1 The algorithm for checking termination with the Jordan decomposition

Input: G;
Result: State, Instead.
1: **Function** [State, Instead] = CheckTermination(G);
2: $|\Phi\rangle \leftarrow$ MaxEntangledState where $d(|\Phi\rangle) = d(G)$;
3: $[J, S] \leftarrow$ Jordan(G) s.t. $G = SJS^{-1}$, $|\phi\rangle' \leftarrow S^{-1}|\Phi\rangle$;
4: $[|u\rangle\ \ |v\rangle\ \ |w\rangle]^{\mathrm{T}} \leftarrow |\phi\rangle'$ where $d(|u\rangle)$ = number of eignvalues with value 0, $d(|w\rangle)$ = number of eignvalues with module 1, $d(|v\rangle) = d(|\phi\rangle') - d(|u\rangle) - d(|w\rangle)$;
5: **if** $|v\rangle = 0$ and $|w\rangle = 0$ **then**
6: $k \leftarrow$ CalSteps(G);
7: State $\leftarrow$ Finite termination;
8: Instead $\leftarrow \sum_{n=0}^{k-1} N_0 G^n$;
9: **else if** $|w\rangle = 0$ **then**
10: State $\leftarrow$ Almost-surely termination;
11: Instead $\leftarrow \sum_{n=0}^{\infty} N_0 G^n = N_0(I - G)^{-1}$;
12: **else**
13: State $\leftarrow$ Non-termination or unknown;
14: Instead $\leftarrow$ NULL;
15: **end if**

Algorithm 1 (Algo 1) re-illustrates the process we used to prove Theorem 1. The input of the function CheckTermination is G which is the matrix representation of $\mathcal{G}$. As the termination is defined on the program scheme that the program excludes the input quantum state, our concern, $\mathcal{G}$ with the measurement M and the loop body $\mathcal{E}$, are sufficient to check termination. The outputs of Algorithm 1 are: State which is byte that represents termination, almost-surely termination, or non-termination and Instead which is also a matrix representation for substituting the program segment as a new super-operator $\sum_{n=0}^{k-1} N_0 G^n$ where N_0 is the matrix representation of measurement operator M_0. It can be easily inferred that for a quantum program scheme with k-steps, the first $k-1$ steps can be treated as G^n and the last step must be M_0 which shows the after-measurement effect. Lines 10–12 are for cases where the program almost-surely terminates. According to Proposition 1, the effects of the infinite steps can be substituted

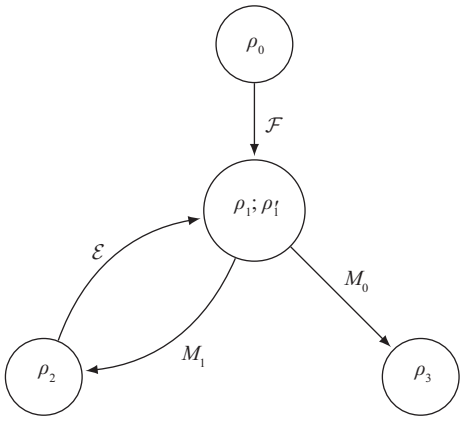

Figure 1 The flowchart of Qloop.

by $N_0(I-G)^{-1}$. The last scenario is the non-termination or the unknown type of programs. The most significant difficulty for implementing this algorithm is the Jordan decomposition on line 3. Even though the Jordan decomposition can achieve a time complexity of $\mathcal{O}(n^3)$ [12], general Jordan decomposition toolkit often consumes unacceptable time as it invokes symbolic calculation to avoid the discontinuous of entries in the process.

Compared to Algorithm 1, Algorithm 2 (Algo 2) can achieve a far more reasonable time complexity without relying on the Jordan decomposition. Input G and the outputs State and Instead are defined as per Algorithm 1. Line 2 generates a maximally entangled state.

Algorithm 2 The algorithm for checking termination without the Jordan decomposition

Input: G;
Result: State, Instead.
1: **Function** [State, Instead] = CheckTermination(G);
2: $|\Phi\rangle \leftarrow$ MaxEntangledState;
3: $[V, D] \leftarrow \text{eig}(G^\dagger)$;
4: $D_1 \leftarrow D$ where their eigenvalues with module 1;
5: **if** $G^d|\Phi\rangle = 0$ where d is the dimension of G **then**
6: $k \leftarrow$ CalSteps(G);
7: State $\leftarrow$ Finite termination;
8: Instead $\leftarrow \sum_{n=0}^{k-1} N_0 G^n$;
9: **else if** $|\Phi\rangle \perp D_1$ **then**
10: State $\leftarrow$ Almost-surely termination;
11: Instead $\leftarrow \sum_{n=0}^{\infty} N_0 G^n = N_0(I-G)^{-1}$;
12: **else**
13: State $\leftarrow$ Non-termination or unknown;
14: Instead $\leftarrow$ NULL;
15: **end if**

The key difference is in lines 3–5 where we focus on Proposition 2 and use the eigenvector set of $G^\dagger$ to check the termination. This avoids the need for the Jordan decomposition. It is worth noting that even though we have proven the two algorithms to be equivalent, calculating the eigenvalues and eigenvectors is a theoretically and practically polynomial process.

One of the necessary steps in both Algorithms 1 and 2 is CalSteps which decides k to output Instead. k can be decided easily with Theorem 2.

4 Termination examples — Qloop

A Qloop is a simplified version of a quantum random walk — a powerful quantum computation algorithm. The flow chart of a Qloop is shown in Figure 1.

Unlike a 1-dimensional quantum walk, a Qloop does not include a direction translation operator. The

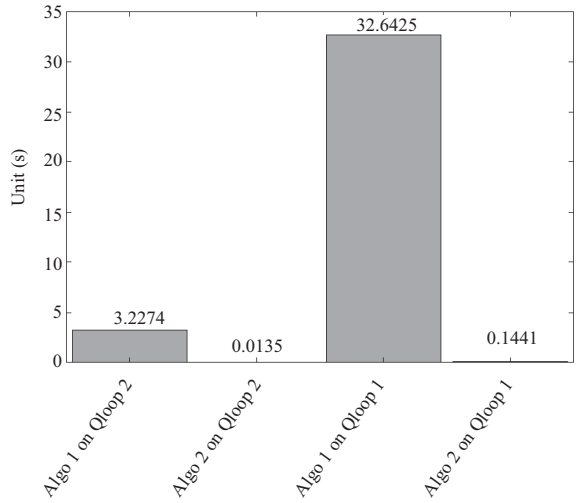

Figure 2 Algorithms 1 and 2 are programmed onto Matlab 2017b. The experiment is conducted on the PC with Core i7 and 16 GB memory. Qloops 1 and 2 are executed repeated 1000 times for one setting and statistics over the total running time. The figure clearly illustrates Algorithm 2 without the Jordan decomposition process has a significant acceleration on analysing of the termination. For both termination and almost-surely scenario, Algorithm 2 speeds up almost 300 times than Algorithm 1.

Qloop 1 Qloop with Hadamard gate as a sub-program

Input: $\rho = \rho_1 = |1\rangle\langle 1|$, $M = \{|0\rangle\langle 0|, |1\rangle\langle 1|\}$, HGate $= \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$;
Result: ρ.
1: **while** $M(\rho)$ **do**
2: HGate(ρ);
3: **end while**
4: **return** ρ.

Qloop 2 Qloop with bit flip gate as a sub-program

Input: $\rho = \rho_1 = |1\rangle\langle 1|$, $M = \{|0\rangle\langle 0|, |1\rangle\langle 1|\}$, XGate $= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$;
Result: ρ.
1: **while** $M(\rho)$ **do**
2: XGate(ρ);
3: **end while**
4: **return** ρ.

direction Left or Right of ρ_1 is only determined by the measurement result. If the result is 0, the quantum state moves to ρ_3, whereas if the result is 1, the quantum state moves to ρ_2. After an operator $\mathcal{E}$ which represents a simple 1-step quantum operation or a more complicated sub-program, the current state moves to ρ_1' and a measurement M is projected onto it. The new measurement result may lead to a Left or a Right routine. The following two termination examples show a fixed input with different loop bodies.

Although Qloops 1 and 2 only have a small section of differences in the sub-program, these programs have completely different behaviours. After the first measurement, the quantum state $\rho = |1\rangle\langle 1|$, but after the sub-programs it would be $\rho = \frac{1}{2}(|0\rangle - |1\rangle)(\langle 0| - \langle 1|)$ or $\rho = |0\rangle\langle 0|$ respectively. This could lead to different results when ρ is measured again. For instance, in Qloop 1 which is an almost-surely terminating program, the program would immediately terminate if the measurement result is 0 with a probability 50%. However, if the result is 1 with a probability of 50%, the subprogram would continue executing and change the state to $\rho = \frac{1}{2}(|0\rangle - |1\rangle)(\langle 0| - \langle 1|)$ in the next loop. In Qloop 2 which is a terminating program, the program would immediately terminate with a probability of 100%.

We implemented Algorithms 1 and 2 in Matlab and tested the speed of both with: a Qloop case with a Hadamard gate and a Qloop case with a bit-flip gate respectively. Figure 2 shows the results.

5 Conclusion and discussion

This paper extended the results in [6] (see also Chapter 5 in [4]). Two computational termination analysis algorithms for quantum **while**-language were presented, along with the least upper bound of the terminating steps for outputting matrix representation with terminating quantum programs. The termination analysis experiments illustrated that the algorithm without the Jordan decomposition process can significantly speed up the calculation. Moreover, both algorithms have been integrated into QSI's termination module [13] to provide extra information about the characteristics of quantum programs for the quantum compiler.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant Nos. 61672007, 11771011, 11647140). We were grateful to Ying DONG for his helpful dicussions.

References

1 Bourdoncle F. Abstract debugging of higher-order imperative languages. SIGPLAN Not, 1993, 28: 46–55
2 Cook B, Podelski A, Rybalchenko A. Termination proofs for systems code. In: Proceedings of the 27th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2006. 415–426
3 Brockschmidt M, Cook B, Ishtiaq S, et al. T2: temporal property verification. In: Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2016. 387–393
4 Ying M S. Foundations of quantum programming. In: Proceedings of the 8th Asian Conference on Programming Languages and Systems, 2010
5 Ying M S, Feng Y. Quantum loop programs. Acta Inform, 2010, 47: 221–250
6 Ying M S, Yu N K, Feng Y, et al. Verification of quantum programs. Sci Comput Program, 2013, 78: 1679–1700
7 Li Y J, Yu N K, Ying M S. Termination of nondeterministic quantum programs. Acta Inform, 2014, 51: 1–24
8 Yu N K, Ying M S. Reachability and termination analysis of concurrent quantum programs. In: Proceedings of International Conference on Concurrency Theory, 2012. 69–83
9 Li Y J, Ying M S. Algorithmic analysis of termination problems for quantum programs. In: Proceedings of ACM on Programming Languages, 2017
10 Horn A R, Johnson R C. Matrix Analysis. Cambridge: Cambridge University Press, 1990
11 Paulsen V. Completely Bounded Maps and Operator Algebras. Cambridge: Cambridge University Press, 2002
12 Beelen T, van Dooren P M. Computational aspects of the Jordan canonical form. In: Reliable Numerical Computation. Oxford: Clarendon Press, 1990. 57–72
13 Liu S S, Zhou L, Guan J, et al. $Q|SI\rangle$: a quantum programming environment (in Chinese). Sci Sin Inform, 2017, 47: 1300–1315