Notes on Stack Machines and Quantum Stack Machines

Daowen Qiu

School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510006, China
{E-mail:issqdw@mail.sysu.edu.cn(D.Qiu)}

Abstract. Multi-stack machines and Turing machines can simulate to each other. In this note, we give a succinct definition of multi-stack machines, and from this definition it is clearly seen that pushdown automata and deterministic finite automata are special cases of multi-stack machines. Also, with this mode of definition, pushdown automata and deterministic pushdown automata are equivalent and recognize all context-free languages. In addition, we are motivated to formulate concise definitions of quantum pushdown automata and quantum stack machines.

Keywords: Multi-Stack Machines, Pushdown Automata; Finite Automata; Turing Machines; Quantum Pushdown Automata; Quantum Multi-Stack Machines.

In the theory of classical computation [5], both multi-stack machines, as a generalization of pushdown automata, and multi-counter machines can efficiently simulate Turing machines (two-stack machines and two-counter machines can efficiently simulate Turing machines.) [8,3,5].

A different definition of pushdown automata was given in [14], and in this note we will further show this definition is equivalent to the traditional pushdown automata [5]. One-way nondeterministic stack automata were studied from the point of view of space complexity [6]. Quantum pushdown automata and quantum stack machines as well quantum counter machines were studied in [4,11,10,7,15]. However, these quantum computing models seem somewhat complex in practice from the viewpoint of physical realizability.

Now we give a new definition of two-stack machines (multi-stack machines are similar). First, we need some notations. Let Γ denote a finite set of stacks, and let ε denote an empty string. Denote $\Gamma_i(\downarrow) = \{X(i,\downarrow) : X \in \Gamma\}$ and $\Gamma_i(\uparrow) = \{X(i,\uparrow) : X \in \Gamma\}$, $i = 1, 2$, where $X(i,\downarrow)$ and $X(i,\uparrow)$ denote "push" X and "pop" X in stack i, respectively. In addition, we denote $\Gamma_i(\updownarrow) = \Gamma_i(\downarrow) \cup \Gamma_i(\uparrow)$, and

$$\Gamma_{1,2}(\updownarrow) = (\Gamma_1(\updownarrow) \times \Gamma_2(\updownarrow)) \cup (\Gamma_1(\updownarrow) \times \{\varepsilon\}) \cup (\{\varepsilon\} \times \Gamma_2(\updownarrow)). \tag{1}$$

More exactly, we can represent $\Gamma_{1,2}(\updownarrow)$ as follows.

$$\begin{aligned}\Gamma_{1,2}(\updownarrow) = &\{(A,B) : A \in \Gamma_1(\updownarrow), B \in \Gamma_2(\updownarrow)\} \\ &\cup \{(A,\varepsilon) : A \in \Gamma_1(\updownarrow)\} \cup \{(\varepsilon,B) : B \in \Gamma_2(\updownarrow)\}.\end{aligned} \tag{2}$$

Without the number of stack, $\Gamma(\downarrow) = \{X(\downarrow) : X \in \Gamma\}$ and $\Gamma(\uparrow) = \{X(\uparrow) : X \in \Gamma\}$, where $X(\downarrow)$ and $X(\uparrow)$ denote "push" X and "pop" X, respectively.

Definition 1. *Let Γ be a set of stack. A string $x \in \Gamma_i(\updownarrow)^*$ ($i = 1, 2$) is called to be valid in stack i if its changing from the most left to the most right in x leads to the ith stack being empty for prior empty stack i. A string $(A_1, B_1)(A_2, B_2)\ldots(A_k, B_k) \in \Gamma_{1,2}(\updownarrow)^*$ is called to be valid if both $A_1A_2\ldots A_k$ and $B_1B_2\ldots B_k$ are valid in stacks 1 and 2, respectively.*

For example, for $X, Y \in \Gamma$, then both $X(1,\downarrow)Y(1,\downarrow)X(1,\downarrow)X(1,\uparrow)Y(1,\uparrow)X(1,\uparrow)$ and $Y(2,\downarrow)X(2,\downarrow)X(2,\uparrow)Y(2,\downarrow)Y(2,\uparrow)X(2,\uparrow)$ are valid, and therefore

$$\begin{aligned}&(X(1,\downarrow), Y(2,\downarrow))(Y(1,\downarrow), X(2,\downarrow))(X(1,\downarrow), X(2,\uparrow))(X(1,\uparrow), Y(2,\downarrow))\\&(Y(1,\uparrow), Y(2,\uparrow))(X(1,\uparrow), X(2,\uparrow))\end{aligned} \tag{3}$$

is valid. Clearly, $X(1,\downarrow)Y(1,\downarrow)X(1,\downarrow)Y(1,\uparrow)Y(1,\uparrow)X(1,\uparrow)$ is invalid.

Definition 2. *A two-stack machine is defined as $M = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, F)$ where Q is a finite set of states, Σ is a finite input alphabet, Γ is the stack alphabet, Δ is a finite set of tape symbols, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ denotes the set of accepting states, and transition function δ is defined as follows:*

$$\delta : Q \times (\Gamma_{1,2}(\updownarrow) \cup \Sigma \cup \Delta) \to Q. \tag{4}$$

As deterministic finite automata, it is clear that M outputs a state in Q for any inputting string $x \in (\Gamma_{1,2}(\updownarrow) \cup \Sigma \cup \Delta)^*$ with the initial state q_0, and this outputted state is represented by $\delta^*(q_0, x)$, as that in deterministic finite automata.

We still need some notations. Let S_1 and S_2 be two sets. Then, for any string $x \in (S_1 \cup S_2)^*$, $[x]_{S_1}$ denotes the string that is obtained by deleting all elements not in S_1 from x, that is to say, $[x]_{S_1}$ is a string by only keeping the elements in S_1 from x.

The computing procedure of this two-stack machine M can be formally defined and described as follows.

Definition 3. *Given a finite alphabet Σ, and for any input string $x = a_1a_2\cdots a_k \in \Sigma^*$, x is called to be accepted by $M = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, F)$, if there is a string $s \in (\Gamma_{1,2}(\updownarrow) \cup \Sigma \cup \Delta)^*$ satisfying the following conditions:*

1. $[s]_\Sigma = x$.
2. $\delta^*(q_0, s) \in F$.
3. $[s]_{\Gamma_{1,2}(\updownarrow)}$ *is a valid string of stacks changing.*

If $\delta^*(q_0, s) \in Q\backslash F$, then x is called to be rejected by M.

A language $L \subseteq \Sigma^*$ is called to be recognized by a two-stack machine M, if for any $x \in L$, x is accepted by M, and for any $x \in \Sigma^*\backslash L$, x is rejected by M.

Next we present two non-context free languages that can be recognized by two-stack machines.

Example 1. The language $L_{eq} = \{0^n 1^n 2^n : n \in N\}$ can be recognized by a two-stack machine $M(L_{eq}) = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, F)$ illustrated by the following figure of transformation, where

- $Q = \{q_0, q_1, q_2, q_3, q_1^{'}, q_2^{'}, q_3^{'}, q_a\}$;
- $\Sigma = \{0, 1, 2\}$;
- $\Gamma = \{Z_0, X\}$;
- $\Delta = \{t_0\}$, q_0 and q_a ($F = \{q_a\}$) are initial and accepting states, respectively,

and δ is illustrated in Fig.1 as follows.

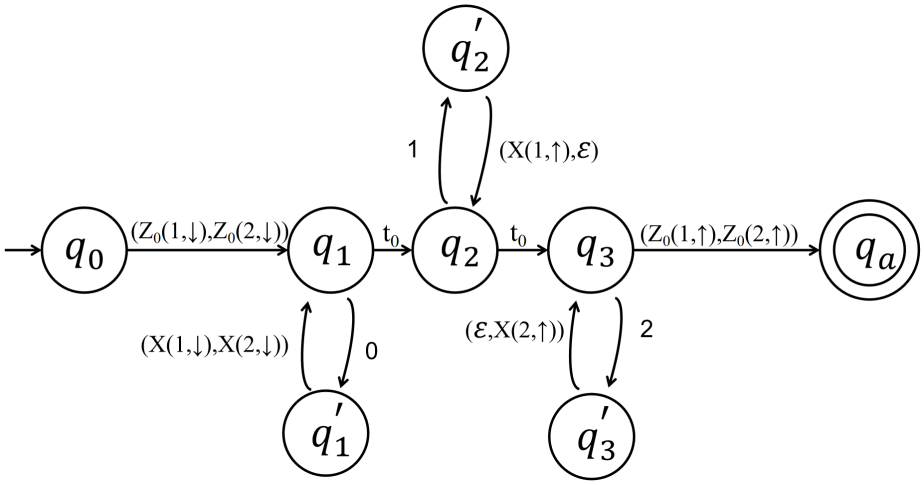

Fig. 1: Two-stack machine $M(L_{eq})$ recognizes L_{eq}.

We give another example in the following.

Example 2. The language $L_w = \{w\#w : w \in \{0,1\}^*\}$ can be recognized by a two-stack machine $M(L_w) = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, F)$ illustrated by the following figure of transformation, where

- $Q = \{q_0, q_1, q_2, q_1^{'}, q_1^{''}, q_2^{'}, q_2^{''}, q_a\}$;
- $\Sigma = \{0, 1, \#\}$;
- $\Gamma = \{Z_0, X\}$;
- $\Delta = \varnothing$, q_0 and q_a ($F = \{q_a\}$) are initial and accepting states, respectively,

and δ is illustrated in Fig.2 as follows.

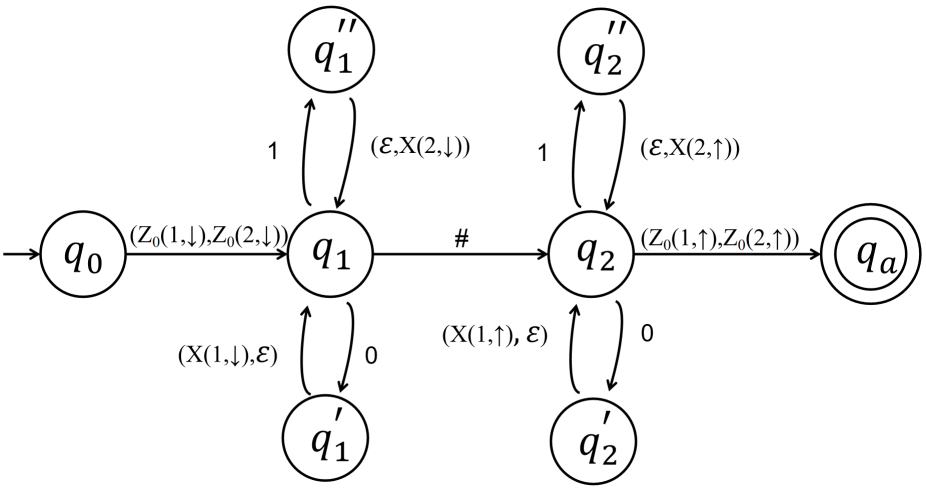

Fig. 2: Two-stack machine $M(L_w)$ recognizes L_w.

Next we show the equivalence between the traditional pushdown automata in [5] and the redefined pushdown automata in [14] that can be thought of as one-stack machines without tape symbols defined above. First let us recall the two definitions.

Definition 4. *[5] A pushdown automaton can be defined as*

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

where

- Q *is finite set of states;*
- Σ *is finite input alphabet;*
- Γ *is finite set of stacks;*
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{Q \times \Gamma^*}$ *is a transform function;*
- $q_0 \in Q$ *is an initial state;*
- $Z_0 \in \Gamma$ *is an initial stack symbol;*
- $F \subseteq Q$ *is a set of accepting states.*

The above model is named as pushdown automata-I (PDA-I) that recognize all context-free languages [5].

Lemma 1. *[5] Let Σ be a finite alphabet. For any language $L \subseteq \Sigma^*$, then L is a context-free language if and only of L is recognized by a pushdown automaton-I.*

Another type of pushdown automata given in [14] is called as pushdown automata-II (PDA-II) and defined as follows:

Definition 5. *A pushdown automaton-II (PDA-II) can be defined as*

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$$

where

- Q *is finite set of states;*
- Σ *is finite input alphabet;*
- Γ *is finite set of stacks;*
- $\delta : Q \times (\Sigma \cup \{\varepsilon\} \cup \Gamma(\updownarrow)\}) \to 2^Q$ *a transform function, where* $\Gamma(\updownarrow) = \{X(\downarrow) : X \in \Gamma\} \cup \{X(\uparrow) : X \in \Gamma\}$;
- $q_0 \in Q$ *is an initial state;*
- $F \subseteq Q$ *is a set of accepting states.*

We describe the computing procedure of PDA-II that is formally defined as follows.

Definition 6. *Given a finite alphabet* Σ*, and for any input string* $x = a_1 a_2 \cdots a_k \in \Sigma^*$*, x is called to be accepted by the PDA-II* $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$*, if there is a string* $s \in (\Gamma(\updownarrow) \cup \Sigma \cup \{\varepsilon\})^*$ *satisfying the following conditions:*

1. $[s]_\Sigma = x$.
2. $\delta^*(q_0, s) \cap F \neq \varnothing$.
3. $[s]_{\Gamma(\updownarrow)}$ *is a valid string of stacks changing.*

Otherwise, x is called to be rejected by M.

The following lemma comes from [14].

Lemma 2. *[14] Let* Σ *be a finite alphabet. For any language* $L \subseteq \Sigma^*$*, then L is a context-free language if and only of L is recognized by a pushdown automaton-II.*

From the above two lemmas it follows that PDA-I and PDA-II are equivalent, that is, any language recognized by PDA-I can be also recognized by PDA-II, and vice versa.

Theorem 1. *Let* Σ *be a finite alphabet. Then for any PDA-I M over* Σ*, there is a PDA-II equivalent to PDA-I (recognizing the same language), and vice versa.*

Proof. It follows from Lemmas 1 and 2. In fact, we can directly prove that PDA-I and PDA-II can simulate to each other by means of a constructivity method. Here we only outline the basic ideas without presenting the details.

Given a PDA-I

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F),$$

we construct a PDA-II as

$$M_N = (Q_N, \Sigma, \Gamma, \mu, q_N, F)$$

where $Q_N = Q \cup \{q_N\} \cup Q'$ for a set of some auxiliary states Q' to be fixed. For any transformation in PDA

$$(p, \gamma) \in \delta(q, a, X),$$

where $q, p \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $X \in \Gamma$, $\gamma \in \Gamma^*$ and $\gamma = X_1X_2...X_n$ with $X_i \in \Gamma$ $(i = 1, 2, \ldots, n)$, we define corresponding transformations in PDA-II

$$q_p{}^{(0)} \in \mu(q, a)$$

$$q_p{}^{(1)} \in \mu(q_p{}^{(0)}, X(\uparrow))$$

$$q_p{}^{(2)} \in \mu(q_p{}^{(1)}, X_n(\downarrow))$$

$$...$$

$$q_p{}^{(n)} \in \mu(q_p{}^{(n-1)}, X_2(\downarrow))$$

$$p \in \mu(q_p{}^{(n)}, X_1(\downarrow))$$

where $q_p{}^{(0)}, q_p{}^{(1)}, ..., q_p{}^{(n)} \in Q'$. In addition, we define

$$q_0 \in \mu(q_N, Z_0(\downarrow)).$$

Then it can be checked that the above M and M_N recognize the same language.

On the contrary, for a given PDA-II $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, we define a PDA-I as $M_N = (Q, \Sigma, \Gamma', \delta', q_0, Z_0, F)$, where $\Gamma' = \Gamma \cup \{Z_0\}$, and δ' is defined as follows:

1. For any $a_{k+1} \in \Sigma \cup \{\varepsilon\}$, $r_{k+1} \in \delta(r_k, a_{k+1})$ and $X \in \Gamma \cup \{Z_0\}$,

$$(r_{k+1}, X) \in \delta'(r_k, a_{k+1}, X).$$

2. For any $a_{k+1} \in \Gamma(\updownarrow)$, and $r_{k+1} \in \delta(r_k, a_{k+1})$,
   (1) if $a_{k+1} = Z_{k+1}(\uparrow)$, then

$$(r_{k+1}, \epsilon) \in \delta'(r_k, \epsilon, Z_{k+1});$$

   (2) if $a_{k+1} = Z_{k+1}(\downarrow)$, then for any $X \in \Gamma$

$$(r_{k+1}, Z_{k+1}X) \in \delta'(r_k, \epsilon, X).$$

   Also it can be checked that the above M and M_N recognize the same language. So, the proof is completed.

In view of the definitions of PDA-I and PDA-II, we know both models (more exactly, their transformation functions) are *nondeterministic*. Concerning PDA-I, it has been proved that PDA-I and deterministic PDA-I are not equivalent [5], since the language $L_{wwr} = \{ww^R : w \in \{0, 1\}^*\}$ can be recognized by PDA-I, but it can not be recognized by any deterministic PDA-I [5].

Now we give the definition of deterministic PDA-II formally as follows.

Definition 7. *A deterministic pushdown automaton-II (DPDA-II) can be defined as*

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$$

where

- Q *is finite set of states;*
- Σ *is finite input alphabet;*
- Γ *is finite set of stacks;*
- $\delta : Q \times (\Sigma \cup \Gamma(\updownarrow)\}) \to Q$ *a transform function, where* $\Gamma(\updownarrow) = \{X(\downarrow) : X \in \Gamma\} \cup \{X(\uparrow) : X \in \Gamma\}$*;*
- $q_0 \in Q$ *is an initial state;*
- $F \subseteq Q$ *is a set of accepting states.*

Clearly, if $\Sigma \cup \Gamma(\updownarrow)$ is thought of as the input alphabet, then deterministic PDA-II and PDA-II M are DFA and nondeterministic finite automata (NFA), respectively.

By using the subset construction method that can construct a DFA being equivalent to any given NFA [5], it is clear that for any PDA-II M, there is a deterministic PDA-II $M^{'}$ equivalent to M. So, we have the following result.

Theorem 2. *Let* Σ *be a finite alphabet. For any PDA-II* $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$*, there is a DPDA-II* $M^{'} = (Q^{'}, \Sigma, \Gamma, \delta^{'}, q_0^{'}, F^{'})$ *that is equivalent to* M*.*

Proof. First, we regard $\Sigma \cup \Gamma(\updownarrow)$ as the same input alphabet of M and $M^{'}$. Then $M^{'}$ follows from the subset construction method that transfers a nondeterministic finite automaton to deterministic finite automaton [5].

If $\Sigma \cup \Gamma(\updownarrow)$ are regarded as the input alphabet of M and $M^{'}$, then we denote $L(\Sigma \cup \Gamma(\updownarrow)) \subseteq (\Sigma \cup \Gamma(\updownarrow))^*$ as the language over $\Sigma \cup \Gamma(\updownarrow)$ recognized by M and $M^{'}$, and clearly it is a regular language over $\Sigma \cup \Gamma(\updownarrow)$. The rest is to show that M and $M^{'}$ also recognize the same language over Σ.

Let $L(\Sigma)$ and $L(\Sigma)^{'}$ denote the languages recognized by M and $M^{'}$, respectively. Our aim is to show $L(\Sigma) = L(\Sigma)^{'}$.

As is shown [14], the language denoted by $L(\Gamma(\updownarrow))$, consisting of all valid strings of stacks in $\Gamma(\updownarrow)^*$, is context-free. Then the extended language, denoted by $L(\Gamma(\updownarrow), \Sigma)$, which is composed by adding any elements from Σ^* to any position in the strings of $L(\Gamma(\updownarrow))$, is also context-free. Formally, $L(\Gamma(\updownarrow), \Sigma)$ can be described mathematically as follows.

$$L(\Gamma(\updownarrow), \Sigma) = \{w : w \in (\Gamma(\updownarrow) \cup \Sigma)^*, [w]_{\Gamma(\updownarrow)} \in L(\Gamma(\updownarrow))\}. \tag{5}$$

Now we have

$$\begin{aligned} L(\Sigma) &= \left[L(\Sigma \cup \Gamma(\updownarrow)) \cap L(\Gamma(\updownarrow), \Sigma)\right]_{\Sigma} \\ &= L(\Sigma)^{'}, \end{aligned} \tag{6}$$

and we have completed the proof.

For a deterministic PDA-II (DPDA-II) M with the input alphabet Σ and the set of stack symbols Γ, if $\Sigma \cup \Gamma(\updownarrow)$ is thought of as the input alphabet, then M is a DFA. From the above proof a corollary follows.

Corollary 1. *For any (deterministic) PDA-II M with the input alphabet Σ and the set of stack symbols Γ, let $L(\Sigma \cup \Gamma(\updownarrow))$ denote the language recognized by M with the input alphabet $\Sigma \cup \Gamma(\updownarrow)$, then the language recognized by M is $[L(\Sigma \cup \Gamma(\updownarrow)) \cap L(\Gamma(\updownarrow), \Sigma)]_{\Sigma}$.*

We give an example to show that the language $L_{wwr} = \{ww^R : w \in \{0,1\}^*\}$ can be recognized PDFA-II in the following (though it is not recognized by DPDA-I [5]).

Example 3. The language $L_{wwr} = \{ww^R : w \in \{0,1\}^*\}$ can be recognized by a PDA-II $M(L_{wwr}) = (Q, \Sigma, \Gamma, \delta, q_0, F)$ illustrated by the following figure of transformations, where

- $Q = \{q_0, q_1, q_2, q_1^{'}, q_1^{''}, q_2^{'}, q_2^{''}, q_a\}$;
- $\Sigma = \{0, 1\}$;
- $\Gamma = \{Z_0, X_0, X_1\}$;
- $q_0 \in Q$ is an initial state, and $F \subseteq Q$ is a subset of accepting states,

and δ is illustrated in Fig.3 as follows.

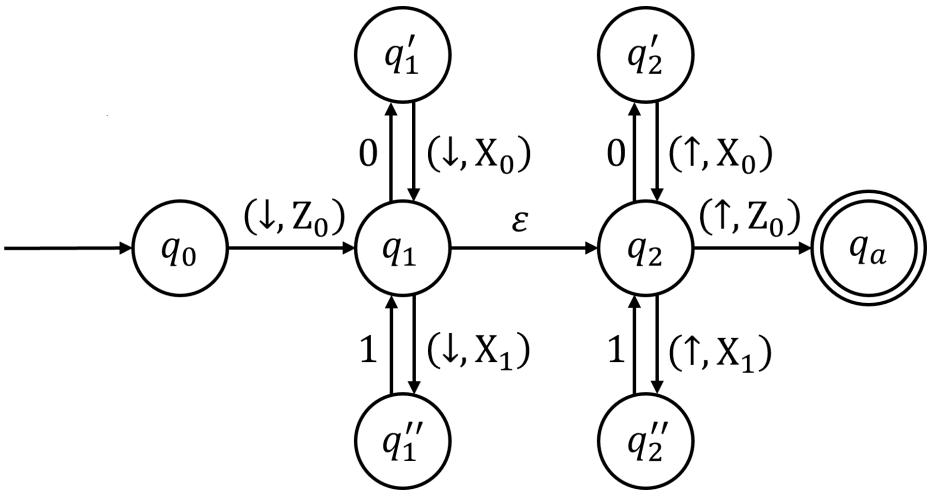

Fig. 3: PDA-II $M(L_{wwr})$ recognizes L_{wwr}.

Here we would like to point out two-stack machines are more general models, and DPDA-II as well as DFA are special two-stack machines.

Remark 1. Let $M = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, F)$ be a two-stack machine. Then we have:

– if only one stack is used and tape symbols are removed, then M reduces to a DPDA-II;
– if two stacks and tape symbols are removed, then M reduces to a DFA.

Finally, we would consider quantum version of these models. In light of the previous definitions, we can define quantum pushdown automata. To avoid confusion with the definitions of quantum pushdown automata in literature [4,12,10], we name it as quantum pushdown automata-II (QPDA-II). We suppose some background concerning quantum automata is known and we can refer to [13].

Suppose Q is finite set of states and denote $|Q|$ as the number of states in Q. Then let $\mathcal{H}_Q$ be the Hilbert space with basis identified with Q.

Definition 8. *A quantum pushdown automaton-II (QPDA-II) can be defined as*

$$M = (Q, \Sigma, \Gamma, \mathcal{U}, q_0, F)$$

where

– Q *is finite set of states;*
– Σ *is finite input alphabet;*
– Γ *is finite set of stacks;*
– $\mathcal{U} = \{U_t : t \in \Sigma \cup \Gamma(\updownarrow)\}$*, where* U_t *is unitary operator on Hilbert space* $\mathcal{H}_Q$*;*
– $|q_0\rangle$ *is an initial state, where* $q_0 \in Q$*;*
– $F \subseteq Q$ *is a set of accepting states.*

For any $x \in \Sigma^*$, denote

$$S_x = \{s \in (\Gamma(\updownarrow) \cup \Sigma)^* : [s]_\Sigma = x, [s]_{\Gamma(\updownarrow)} \in Val(\Gamma(\updownarrow))\}, \tag{7}$$

where $Val(\Gamma(\updownarrow))$ denotes the set consisting of all valid strings of stacks changing.

As usual, the accepting probability of QPDA-II M is defined as follows. For any $x \in \Sigma^*$, the probability $P_a(x)$ of x being accepted by M is

$$P_a(x) = \sup\{\sum_{q \in F} \langle q|U_s|q_0\rangle : s \in S_x\} \tag{8}$$

where $U_s = U_{a_k} U_{a_{k-1}} \cdots U_{a_1}$ if $s = a_1 a_2 \cdots a_k$.

In [11], quantum stack machines are defined. Here we give a different definition, named as quantum 2-stack machines-II (Q2SM-II).

Definition 9. *A quantum two-stack machine is defined as*

$$M = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, F)$$

where where

– Q *is finite set of states;*
– Σ *is finite input alphabet;*
– Γ *is finite set of stacks;*
– Δ *is finite set of tape symbols;*

- $\mathcal{U} = \{U_t : t \in \Sigma \cup \Gamma(\updownarrow) \cup \Delta\}$, *where* U_t *is unitary operator on Hilbert space* $\mathcal{H}_Q$*;*
- $|q_0\rangle$ *is an initial state, where* $q_0 \in Q$*;*
- $F \subseteq Q$ *is a set of accepting states.*

We can also define the accepting probability for Q2SM-II M. For any $x \in \Sigma^*$, denote

$$S_x^{'} = \{s \in (\Gamma_{1,2}(\updownarrow) \cup \Sigma \cup \Delta)^* : [s]_\Sigma = x, [s]_{\Gamma_{1,2}(\updownarrow)} \in Val(\Gamma_{1,2}(\updownarrow))\}, \quad (9)$$

where $Val(\Gamma_{1,2}(\updownarrow))$ denotes the set consisting of all valid strings of stacks changing.

The accepting probability of Q2SM-II M is defined as follows. For any $x \in \Sigma^*$, the probability $P_a(x)$ of x being accepted by M is

$$P_a(x) = \sup\{\sum_{q \in F} \langle q|U_s|q_0\rangle : s \in S_x^{'}\} \quad (10)$$

where $U_s = U_{a_k} U_{a_{k-1}} \cdots U_{a_1}$ if $s = a_1 a_2 \cdots a_k$.

It is seen that Q2SM-II are more general quantum computing models, since both QPDA-II and quantum finite automata (QFA) [9] are special cases of Q2SM-II. Also, QFA are special case of QPDA-II.

The relationships between Q2SM-II and quantum Turing machines (QTMs) [1,2] need to be further considered. Also, how to do simulations by quantum circuits is to be studied furthermore, as QTMs simulated by quantum circuits [16].

References

1. E. Bernstein and U. Vazirani, Quantum complexity theory, SIAM J. Comput. 26 (1997) 1411-1473.
2. D. Deutsh, Quantum theory, the Church-Turing principle and the universal quantum computer, Proc. Roy. Soc. London A 400 (1985) 97-117.
3. P.C. Fischer, Turing machine with restricted memory access, Information and Control 9 (4) (1966) 364-379.
4. M. Golovkins, Quantum Pushdown Automata, in: Proc. 27th Conf. on Current Trends in Theory and Practice of Informatics, Milovy, Lecture Notes in Computer Science, Vol. 1963, Spring-Verlag, Berlin, 2000, pp. 336-346.
5. J.E. Hopcroft and J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addision-Wesley, New York, 1979.
6. O. H. Ibarra, J. Jirasek Jr., I. McQuillan, and L. Prigioniero, Space Complexity of Stack Automata Models, DLT 2020, LNCS 12086, pp. 137-149, 2020.
7. M. Kravtsev, Quantum finite one-counter automata, in: SOFSEM'99, Lecture Notes in Computer Science, Vol.1725, Springer-Verlag, Berlin, 1999, pp.431-440.
8. M.L. Minsky, Recursive unsolvability of Post's problem of 'tag' and other topics in the theory of Turing machines, Annals of Mathematics 74 (3) (1961) 437-455.
9. C. Moore and J.P. Crutchfield, Quantum automata and quantum grammars, Theoret. Comput. Sci. 237 (2000) 275-306. Also quant-ph/9707031, 1997.

10. M. Nakanishi, Quantum Pushdown Automata with a Garbage Tape, arXiv:1402.3449.
11. D.W. Qiu, Simulations of quantum Turing machines by quantum multi-stack machines, quant-ph/0501176, 2005.
12. D.W. Qiu and M.S. Ying, Characterization of quantum automata, Theoret. Comput. Sci. 312 (2004) 479-489.
13. D.W. Qiu, Theoretical Foundations of Quantum Computing, Morgan Kaufmann, 2025.
14. J. Watrous, Introduction to the Theory of Computing, 2020.
15. T. Yamasaki, H. Kobayashi, H. Imai, Quantum versus deterministic counter automata, Theoret. Comput. Sci. 334 (2005) 275-297.
16. A.C. Yao, Quantum circuit complexity, in: Proc. 34th IEEE Symp. on Foundations of Computer science, 1993, pp. 352-361.