Queue Automata

Anupam Bhattacharjee
Computer Science & Engineering Department
Bangladesh University of Engineering & Technology
Dhaka-1000, Bangladesh
abrbuet@yahoo.com

Biplob Kumar Debnath
Computer Science & Engineering Department
Bangladesh University of Engineering & Technology
Dhaka-1000, Bangladesh.
bkdebnath@cse.buet.ac.bd

Abstract
In this paper, a queue automaton that is an extension of finite automata is defined. A queue automaton is an automaton having a queue as a store and having three basic operations (enqueue, dequeue and sense empty) on the queue. Then, it is shown that the class of languages accepted by queue automata is the same class of languages accepted by nondeterministic finite automata (NFA) and thus by a deterministic finite automata (DFA). Simulation of nondeterministic finite automata using the defined automata is also shown. Finally, generalization of the new automaton and other automata is described.

Keywords
FA, NFA, DFA, PDA, Regular expression, regular languages, abstract machines, TM.

INTRODUCTION
The field of computer science is established and founded on some theoretical machines which are called abstract machines. Some of the machines, after slight modifications, have been applied to practical fields of computer science. Some have not yet implemented. But, as for the quick hardware change, anytime it may be possible to implement them. In this paper, such an abstract machine Queue Automata (QA) is introduced. A Queue Automaton is an automaton having a queue as a store and having three basic operations (enqueue, dequeue, and sense empty) on the queue. Using a two-queue machine, any context-free languages can be accepted nondeterministically in $O(n \log n)$ time [7]. Queue Automata can be extended to use $n$ queues [8]. A stack machine can be converted to a QA in $O(n \log n)$ time but a single stack machine can’t [8].

So, in case of queue automata, languages that are accepted by pushdown automata (PDA) is also accepted here (for example, $L = \{ wcw^t : w \in \{a,b\}^* \text{ and } c \in \Sigma \}$. But, PDA has very small subset of languages that are accepted by QA. Many languages that are accepted easily in QA, are hard to get accepted in other automata. So, QA makes easier structure in case of many languages [7, 8]. It is very easy to engineer a modern compiler that has the structure of QA along with all the previous compiler techniques. So, QA does not ban any modern compilation technique. Again, any DFA and NFA with the help of QA can be constructed. So, it can be treated as a generalization of these automata. With the help of queue automata we can construct the simulation of any NFA [8].

DEFINING THE AUTOMATON
Let us define a new automaton called Queue based Automaton or simply QA. A queue based automaton is a hex-tuple, $M = (K, \Sigma, \zeta, \delta, s, F)$ where,
$K$ = set of state(s)
$\Sigma$ = alphabet
$\zeta$ = queue content (Queue alphabet)
$\delta$ = transition relation
where, $\delta \subseteq (K \times (\Sigma \cup \{e\}) \times \zeta^*) \times (K \times \zeta^*)$
s = starting state
$F$ = set of final state(s)

BASIC OPERATIONS
In the expression $(p, a, \ldots, \beta, (q, \gamma, \ldots, \theta)) \in \delta$, the automata is now in $p$ state with $a$ at the top and $\beta$ at the bottom of the queue. It may read ‘$a$’ from the input tape (“may read” is mentioned here because if $a = e$ then the input is not consulted), replace the queue content by $\gamma \ldots \theta$ entering state $q$.

Such a pair stated above is called a transition of $M$. If several transitions of $M$ may be simultaneously applicable at any point, the machine is converted to a nondeterministic one.

**Enqueue** operation is defined by the following transitions
$(p, a, e) \Leftrightarrow (p, a, e)$
$(p, q, a) \Leftrightarrow$ enqueues ‘a’ in an empty queue.

**Dequeue** operation is defined as,
$(p, u, a) \Leftrightarrow$ dequeues an ‘a’.
$(p, u, ab) \Leftrightarrow$ dequeues a ‘b’.
$(p, u, ba) \Leftrightarrow$ dequeues an ‘a’.
$(p, u, b) \Leftrightarrow$ dequeues a ‘b’.
$(p, u, e) \Leftrightarrow$ where $q$ may be a halting state.

where $\{p, q\} \subseteq K$ and $\{a, b\} \subseteq \Sigma$. 

...
Configuration
The configuration of a queue automata can be defined as a member of \( K \times \Sigma^* \times \zeta^* \), where the first component is the state of the machine, the second is the input part yet to be read, and the third is the content of the queue. For example, in the configuration \((p, abc, def)\), \(p\) is the present state, \(abc\) is the input to be read, and \(def\) is in the queue. In case of NFA, from some state with some input, there may be several possible next states and again, with some other combination of states and input symbols, there may be no possible move.

Yielding
The operation “yield in one step” is defined by the symbol “\( - \)" as \((p, x, a) (- \) (q, y, \( \rho \)) \) if \((p, a, \beta), (q, \gamma) \) \( \delta \) and \( x = a\gamma\), \( \alpha = \beta n \), and \( \rho = \gamma n \). Let,\( p, a, \) and \( c \) can be empty and where \((p, x, a) \) and \((q, y, \rho)\) are two configurations.

Converting to languages accepted by DFA
Every finite automaton can be viewed as a queue automaton that has no operations on the queue. That means, it enqueues or dequeues in no case.

Converting to languages accepted by NFA
Languages accepted by NFA is the same as those accepted by queue automata.

Closure
The machine is closed under union, concatenation, intersection, complementation and Kleene star. For union operation, let \( M_1 \) \( M_2 \) such that \( L(M_1) \cup L(M_2) \) by QA.

The case can be figured out as,
$M = (K, \Sigma, \zeta, \delta, s, F)$ where,

$K = K_1 \cup K_2 \cup \{s\}$

$F = F_1 \cup F_2$

$\delta = \delta_1 \cup \delta_2$ along with four transitions

1. $(s, a, e), (s, a)$
2. $(s, a, b), (s, a)$
3. $(s, a, a), (s, a)$
4. $(s, a, a), (s, a)$

Simulation of QA

Let us construct a queue to accept $L = \{wcw : w \in \{ab\}^* \text{ and } c \in \Sigma\}$

For example, $ababcabab \in L$ but the string $abcba$ is not in $L$.

Let,

$M = (K, \Sigma, \zeta, \delta, s, F), K = \{s, r, f\}$

$\Sigma = \{a, b, c\}$

$\zeta = \{a, b\}$

$F = \{f\}$

$\delta$ has

1. $(s, a, e), (s, a)$
2. $(s, a, b), (s, b)$
3. $(s, a, c), (r, e))$
4. $(r, a, a), (r, e))$
5. $(r, b, b), (r, e))$
6. $(r, e, e), (f, e, e))$

For example, for $w = abacaba$, it can be stepped as

$(s, abacaba, e) \Rightarrow (s, bacaba, a) \Rightarrow (s, acaba, ba)$

$\Rightarrow (s, caba, aba) \Rightarrow (r, aba, ab)$

$\Rightarrow (r, a, a) \Rightarrow (r, e, e) \Rightarrow (f, e, e)$

As $(s, abacaba, e) \Rightarrow (f, e, e)$ so, $abacaba \in L$.

Again for another example, let us construct an automaton for $L = \{a^m b^n : \{a, b\} \subseteq \Sigma \text{ and } n < m\}$.

Let,

$M = (K, \Sigma, \zeta, \delta, s, F), K = \{s, f\}$

$\Sigma = \{a, b\}$

$\zeta = \{a, b\}$

$F = \{f\}$

$\delta$ has

1. $(s, a, e), (s, a)$
2. $(s, b, a), (s, e)$
3. $(s, b, e), (f, e)$
4. $(f, b, e), (f, e))$

All other possible conditions can be treated as error conditions and can easily be handled by an extra halting state.

For example, for $w = aaabbbb$, it can be stepped as

$(s, aaabbbb, e) \Rightarrow (s, aabbbb, a) \Rightarrow (s, abbbb, aa)$

$\Rightarrow (s, bbbb, aaa) \Rightarrow (s, bbb, aa) \Rightarrow (s, bb, a)$

$\Rightarrow (s, b, e) \Rightarrow (f, e, e)$

As $(s, aaabbbb, e) \Rightarrow (f, e, e)$ so, $aaabbbb \in L$.

So, $L = \{a^m b^n : \{a, b\} \subseteq \Sigma \text{ and } n < m\}$ is accepted by QA.

ACKNOWLEDGMENTS

The Authors are grateful to Dr. M. Kaykobad, Dr. Abul Kashem Mia, Dewan Tanvir Ahmed, Jalal Uddin Mahmud, Md. Eunus Ali, and Uttam Debnath.

REFERENCES


[9] Rosenberg, B., Simulating a stack by queues, University of Miami