Queuing / Counting Automata

Hieu D. Vu
Fort Hays State University
600 Park Street
Hays, KS. 67601
hdvu@fhsu.edu

Abstract - In the software development area, a very important question arise "How the computers process data?" This subject is centered around the concept of Theory of Computing and Automata which were introduced in 1979, and based heavily in Mathematics.

We know that application software can be built using programming languages, but what kind of languages that a computer can understand? How were they created? Were there any rules or grammars to govern the languages? What are Context-Free Languages, Pushdown Automata, and other theoretical machines?

Keywords—Computing theories, Context-Free languages, Context-free grammar, Formal language, Automata, Turing machines, Pushdown automata, Computational problems, Turing machine.

I. INTRODUCTION

Since the development of computers, scientists always try to increase the power of computing for the machines. In reality, the real power of a computer is the data processing capability, and computer scientists can concentrate in either one of the two keys components: hardware that is to build faster more capable processors (CPU-Central Processor Unit), and software or applications that can process large volume of data quickly and accurately.

This research paper will start with the Theory of Computing, Context-Free Languages and Pushdown Automata then introduce other methods for the abstract machine to recognize context-free languages. Queuing Automata is a concept of using queue type data structure instead of stack in pushdown automata. Counting automata is another way to recognize a language using a counter.

II. CONTEXT-FREE LANGUAGES

Context-free languages are languages that have recursive characteristic. This class includes all regular languages and some non-regular, special languages such as the language defined such as L = $\{0^n1^n$: n is greater than or equal 0$\}$. Context-free language can be obtained by either one of the two methods: context-free grammars and push down automata.

II. 1. Context-free Grammars

Context-free grammars which are used to define programming languages in two areas syntactic and compilation.

Definition: A context-free grammar is defined as a 4-tuple G = (V, ∑, R, S), where:

1. V is a finite set of variables such as V = {S, A, B,…} (capital letters)
2. ∑ is a finite set of terminals such as ∑ = {a, b, c,…} (input strings in lower case letters)
3. $V \cap \Sigma = \emptyset = \{\}$ (empty set)
4. S is the start variable. S is an element in set V ($S \in V$)
5. R is a finite set that is a collection of rules. Each rule is of the form A --> w, where $A \in V$ and $w \in (V \cup \Sigma)^*$

Example: Let set R of rules that includes following substitutions:

S --> AB

A --> a

A --> aA

B --> b

B --> bB

We can see, S is the start variable, A and B are variables, and a, b are terminals (input strings). From the rules above, we can construct strings of terminals a and b ($\{a, b\}^*$) using the steps below:

1. Initialize the current string with only the start variable S (starting point).
2. Take any variable in the current string put it in the left-hand side, and using any rule to replace this variable in the current string by the right-hand side of the rule.
3. Go back to step 2 (repeating) until the current string contains only terminals. It is the language that accepted by the grammar (rules).

For illustration, from the starting point, we have:

$$
\begin{aligned}
S &\Rightarrow AB && \text{(rule 1)}\\
&\Rightarrow aAB && \text{(rule 3)}\\
&\Rightarrow aAbB && \text{(rule 5)}\\
&\Rightarrow aaAbB && \text{(rule 3)}\\
&\Rightarrow aaabB && \text{(rule 2)}\\
&\Rightarrow aaabb && \text{(rule 4)}
\end{aligned}
$$

Conclusion, the language in this example is: $L_1 = \{a^m b^n: m \geq 1, n \geq 1\}$ [1]

II.2. Pushdown Automata

- Definition of Automata: is the study of abstract computing machines, before the computers were developed in the late 1930s, as well as the computational problems that can be solved using them. The word automata originally came from the Greek word "αὐτόματα", which means "self-making". An automaton (automata in plural) is an abstract computing device that follows a predefined sequence of operation automatically. [2]

- Pushdown Automata: is a new type of computational model like nondeterministic finite automata but also have a special component called a stack. A stack is a data structure type component that has only one end called stack-top, where we can insert (push) an element into a stack or remove (pop) and element currently is at the top of the stack. Because of that, a stack is classified as an LIFO (Last In, First Out) data structure component. In this pushdown automata, the stack provide additional memory for the finite amount available in the control, they allow pushdown automata to recognize some non-regular languages.

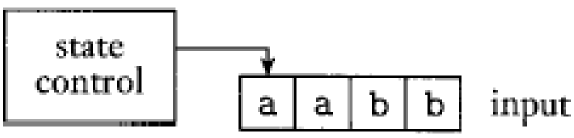


Figure 1: Schematic of a finite automaton

The figure above represents a schema of a finite automaton. The state control (box) represents the states and transition functions, the tape represents input string, and the arrow represents the reading head that points to the next input symbol for reading.

Pushdown automata include a stack as in figure 2.

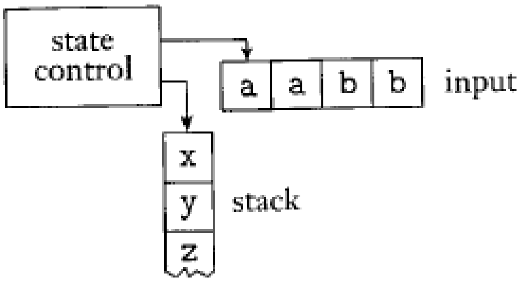


Figure 2: Schematic of a pushdown automaton

A pushdown automaton (PDA) can write (push) symbols in the stack and read (pop) them back later in a LIFO (last in, first out) fashion. When writing a new symbol on the stack, other symbols will be "pushdown" one position, so the new symbol will be pushed in at the top of the stack. All operations (reading, writing) on the stack must be done at the top (only one end).

A stack is proved valuable, because it can hold unlimited amount of data. For example, a finite automaton cannot recognize the language $L_2 = \{0^n1^n \mid n \geq 0\}$ due to the finite memory. A PDA can recognize this language by using stack to store all number 0s it read. They following steps illustrate how a PDA works.

1. Read input symbols, for each 0 is read, push it onto the stack.
2. Repeat step 1 until no more 0s on the input string (until the first 1s is encountered).
3. For each 1s, pop a 0 out of the stack.
4. Repeat step 2 until no more 1s or the end of the input string.

If both the stack and input string become empty, accept the input string as the language

$L_2 = \{0^n1^n\mid n \geq 0\}$. Otherwise, in other cases if the input string became empty or the stack is empty before the other, reject L_2. It is not in the language.

- <u>Definition of Pushdown Automata:</u> A pushdown automaton is a 6-tuple (Q, ∑, Γ, δ, q_0, F) where Q, ∑, Γ, F are finite sets and satisfy the following:
  1. Q is the set of states (q_0, q_1, q_2, …)
  2. ∑ is the input characters (in lower cases)
  3. Γ is the stack alphabets
  4. δ is the transition function: $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon$ ----> $P(Q \times \Gamma_\varepsilon)$
  5. $q_0 \in Q$ is the start state
  6. $F \subseteq Q$ is the set of accept states.

A pushdown automaton M = (Q, ∑, Γ, δ, q_0, F), works (computes) as following. It will accept an input w, if w can be written as $w = w_1w_2w_3\ldots w_n$, where $w_i \in \Sigma_\varepsilon$ $(1 \leq i \leq n)$, and the sequence of states $r_0, r_1, r_2, \ldots, r_n \in Q$, and strings $s_0, s_1, s_2, \ldots s_n \in \Gamma^*$ where string s_i $(0 \leq i \leq n)$ are the content in the stack, exist that satisfy the following conditions:

1. $r_0 = q_0$ and $s_0 = \varepsilon$. It is the starting point for the pushdown automaton (PDA) M, with state q_0 and empty stack (ε).
2. For the first n steps (i = 0, …, n-1), we have $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, where s_i = at and s_{i+1} = bt for some a, b $\in \Gamma_\varepsilon$ and t $\in \Gamma^*$. This condition means that (ADP) M moves properly according to the state, stack, and the next input symbol.
3. r_n = F. This condition states an accept state happen at the input end (string input).

<u>Example:</u>

We use the language in this example $L_3 = \{a^nb^n: n \geq 0\}$ to demonstrate the use of pushdown automata (PDA). Let the pushdown automaton is M_1 = (Q, ∑, Γ, δ, q_1, F), where Q = {q_1, q_2, q_3, q_4}, ∑ = {0, 1}, Γ = {0, $}, F = {$q_1$, q_4), and the transition function δ is given by the table below.

Journal of Multidisciplinary Engineering Science and Technology (JMEST)
ISSN: 2458-9403
Vol. 8 Issue 6, June - 2021

Input:	0			1			ε		
Stack:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2, \$$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

Figure 3: table of transition functions δ

To illustrate the PDA M_1, we can use the state diagram and notation such as "a, b --> c" that means "when the machine read (input) a, it may replace symbol b at the top of the stack with c". All a, b, or c could be the symbol ε. If a (input string character) is ε, the machine may make the transition (b --> c) without reading (any symbol) the input string. If b is ε, the machine may make this transition without reading, popping any symbol from the stack. If c is ε, the machine will not write any symbol on the stack.

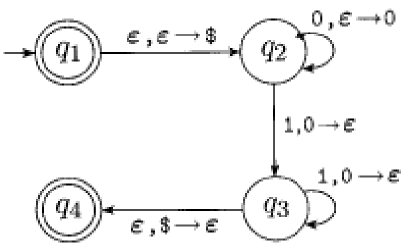


Figure 4: State diagram for the PDA M_1 that recognizes $L_2 = \{0^n1^n | n \geq 0\}$

To find out if the stack is empty? The PDA M_1 initially put a symbol (\$) in the stack to mark the stack is empty when M_1 see it (\$) again. Also the accept state happen only when the machine (M_1) reaches the end of the input string. [3]

III. SIMPLE APPROACH OF PUSHDOWN AUTOMATA

An automaton is an imaginary machine that reads an input string, process the string and accept or reject it. This section will present simple ways by examples to recognize the languages $L_1 = \{a^mb^n | m, n \geq 1\}$, and $L_2 = \{0^n1^n | n \geq 1\}$.

Example 1: Let a finite automaton M_1 be the machine that processes the language

$$L_1 = \{a^mb^n | m, n \geq 1\}.$$

As the reading head reads the input string {a, a, a, …, a (m times), b, b, …, b (n times)}, M_1 moves from initial state q_0 to state q_1 when it read character a(s) and stay in q_1 until when M_1 encounters the first character b, it will move from state q_1 to state q_2 then stays in q_2 for more b(s). When M_1 reach the end of the input string (empty character symbol or ε), it will accept the language $L_1 = \{a^mb^n | m, n \geq 1\}$, and q_2 is the acceptance state. In this finite automaton, we assume there is at least one character a, one character b, and m ≠ n.

Example 2: Let M_2 be the machine to process the language $L_2 = \{a^nb^n | n \geq 1\}$.

Unlike L_1, in this language L_2, the number of characters a(s) and b(s) are equal. In $L_1 = \{a^mb^n | m, n \geq 1\}$, it is not necessary to know or to remember the number of a's, but the following need to remember.

(1) If the first character is b, machine M_2 will reject the string (not in the language)
(2) If character a follows character b, M_2 will reject the string
(3) If character a follows a, and character b follows b, M_2 will accept the string

But, there is a problem with finite automata. It only has a number of finite states therefore cannot remember how many a's in the input string a^nb^n, where n is greater than the number of states of machine M_2. Finite automata does not accept the sets of strings in the languages $L_2 = \{a^nb^n | n \geq$

Journal of Multidisciplinary Engineering Science and Technology (JMEST)
ISSN: 2458-9403
Vol. 8 Issue 6, June - 2021

1}. This problem can be done using pushdown automata. [4]

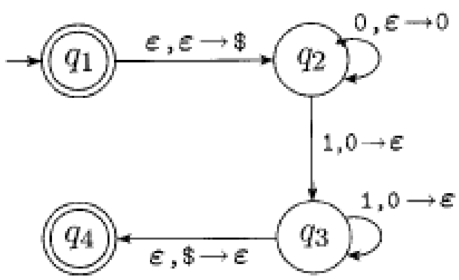
IV. OTHER APPROACHES

In this section, we introduce other automata that recognize the language $L_2 = \{a^n b^n \mid n \geq 1\}$ instead of using pushdown automata.

In the example of pushdown automata in section (II. 2), the pushdown automaton is $M_1 = (Q, \sum, \Gamma, \delta, q_1, F)$, where $Q = \{q_1, q_2, q_3, q_4\}$, $\sum = \{0, 1\}$, $\Gamma = \{0, \$\}$, $F = \{q_1, q_4)$, and the transition function δ is given by the table below.

Input:	0			1			ε		
Stack:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2,$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

And the pushdown automaton M_1 works as illustrated by the following state diagram.



From the starting point, state q_1 the input string is empty and the stack is also empty, then the machine puts a symbol (\$) onto the stack to mark the beginning. Then the head reads the first number 0, the machine will push a 0 onto the stack and move to state q_2. Then it will stay in state q_2 and continue to push (number) 0 onto the stack, as long as the head read (number) 0. When the machine first read a number 1, it will pop a number 0 out of the stack and move to state q_3. Then it will stay in state q_3 and continue to pop (number) 0 out of the stack, as long as a number 1 is read. When the head reaches the end of the tape (input string), which is empty and denoted by symbol ε, and if the symbol \$ is at the top at the stack, the PDA M_1 will move to state q_4 and accept the language $L_2 = \{a^n b^n \mid n \geq 1\}$. Other cases below, M_1 will reject L_2.

a. The head reads number 1 but the (\$) symbol is at the top of the stack (more 1s than 0s)
b. The head reach the end of the input tape, but number 0(s) is still at the top of the stack (more 0s than 1s)

IV.1. Queuing Automata

A queue is a “First In, First Out” (FIFO) data structure, you can visualize a queue of people at a checkout counter in a store. Let the queuing automaton be M_2, we will see how it work in order to recognize language $L_2 = \{a^n b^n \mid n \geq 1\}$.

In a similar fashion of the pushdown automaton M_1 above, we define queuing automaton $M_2 = (Q, \sum, \Gamma, \delta, q_0, F)$, where $Q = \{q_0, q_1, q_2, q_3\}$, $\sum = \{a, b\}$, $\Gamma = \{a\}$, $F = \{q_0, q_3)$, and the transition function δ is given by the table below. The queuing automation M_2 works following these steps:

1. At the starting point, the machine state is q_0, the queue is empty (ε), the input string is also empty (ε).
2. When the head read the first character (a), it will place character (a) at the bottom (tail) of the queue, reset the bottom pointer of the queue (pointer that points to the newest character ‘a’), and move to state q_1. It will stay in this state q_1 and repeat step 2 as long as the head continues to read character (a).
3. When the head encounters the first character (b), it will remove a character

(a) from the top of the queue, reset the top pointer that points to the next character at the top of the queue, and move to state q_2. The theoretical machine M_2 will stay in this state q_2 and repeat step 3 as long as the head reads character (b).

4. When the head reaches the end of the input string (ε), and the queue is also empty (ε), the queuing automaton M_2 will accept language $L_2 = \{a^n b^n \mid n \geq 1\}$.

M_2 will reject other cases, whether the input string or the queue become empty before the other.

We can also use the state diagram to illustrate how queuing automaton M_2 works.

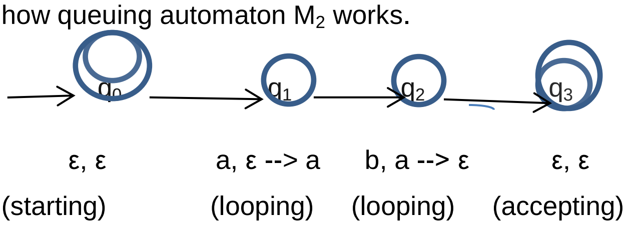


Figure 5: State diagram of machine M_2

IV.2. AUTOMATA WITH COUNTER

This section introduces theoretical machine M_3 that uses a counter instead of a stack or a queue data structures to recognize the language $L_2 = \{a^n b^n \mid n \geq 1\}$ in the example above. The machine M_3 will process an input string (a a a … b b …) using the steps below:

1. Let start with state q_0 and initialize the counter to zero (0).
2. If the machine's head reads the first character of the input string is 'a', then increment the counter and moves to state q_1, otherwise reject the language L_2.
3. Continue to read the next character, if it is another 'a' then increment the counter but stay in state q_1, otherwise go to step 4. Repeating step 3 until character 'b' is encountered.
4. Move to state q_2 and decrement the counter.
5. Continue to read input string until the end of the string (empty or ε). If the character read is 'b' go to step 4, otherwise reject L_2.
6. When the input string is empty and the counter is zero (0) moves to state q_3 an accept the language $L_2 = \{a^n b^n \mid n \geq 1\}$, otherwise reject it.

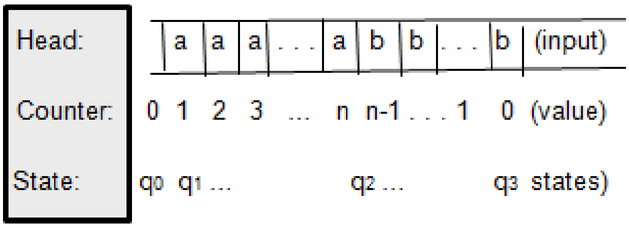


Figure 6: Diagram of M_3: the reading head, counter and states

The theoretical machine M_3 will reject the language L_2 in either cases following:

1. The reading head reaches the end of the input string (empty or ε), but the counter contains a value greater than 0 (more characters 'a' than 'b').
2. The counter contain number 0, but the input string still have more character(s) (more characters 'b' than 'a').

V. CONCLUSION

In 1954, Kleene presented a theorem states that if a language can be defined in one of the three following methods, then it is also defined by the other two. The three methods of defining a language are equivalent.

Kleene's theorem: Any languages can be defined by one of the following equivalent methods:

a. Regular expression, or
b. Finite automaton, or
c. Transition graphs

Can be defined by all three methods. [5]

The Kleene's theorem is considered by many computer scientists that it is the beginning of automata theory. It proved that finite automata can recognize class of languages. These theorems, expressions, automata, and graphs can be processed by theoretical machines. The earliest simple abstract machine for computing was first described by Alan Turing in 1936, and followed by Alonzo Church in 1937 are considered one of the foundational models of computability and in theoretically computer science. [6]

With the advancing of technologies, we continue to explore new concepts for better machines. The machine of the future should be more powerful in computing capability and more flexibility to process different things.

References:

[1]. Maheshwari, Smid, "Introduction to Theory of Computation", Carleton University, Ottawa, Canada. 2014. Pages: 87-88.

[2]. Hopcroft, Motwani, and Ullman, "Introductory to Automata Theory, Languages, and Computation" 2e, Addition-Wesley, 2001. Pages: 1.

[3]. Michael Sipser, "Introduction to the Theory of Computation", Massachusetts Institute of Technology, PWS Publishing Company, Boston, MA. 02116-4324. 1997. Pages: 101-105.

[4]. S.P Eugene Xavier, "Theory of Automata, Formal Languages and Computation", New Age International (P) Ltd., Publishers, 2005. Pages: 159.

[5]. https://assets.ctfassets.net/kdr3qnns3kvk/ohtVAyD8ugH9W04aczUnI/c3122fa3b5346dee2089871cdd7c4eb7/07-Kleenes-Theorem.pdf download 06/17/2021 at 10:22am

[6]. https://plato.stanford.edu/entries/turing-machine/ download 06/17/2021 at 01:10pm