ICT600 Computational Mathematics

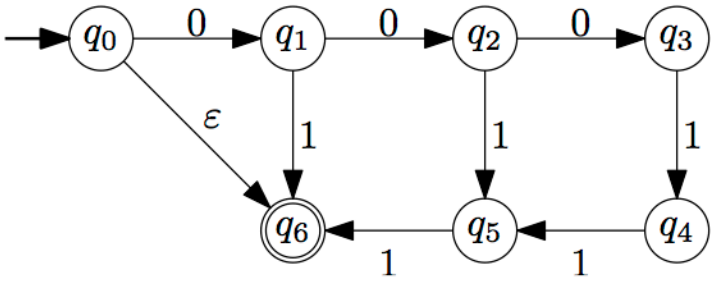
Context-free Grammars and Pushdown Automata

Composed by Cholwich Nattee, D.Eng.

.-**-..-**-..-**-..-**-..-**-.~ ♡ Have Fun!♡ ∽.-**-..-**-..-**-..-**-..-**-.

1 Regular and Non-regular Languages

Let's take the language $A = \{0^n1^n | 0 \leq n \leq 3\}$. We can show that A is a regular language by finding an NFA that recognizes A.



State q_1 shows that the machine has read 1 zero. State q_2 and q_3 similarly shows that the machine has read 2 and 3 zeros, respectively. The machine needs to remember the number of zeros in order to verify the number of ones.

Then, let's consider the language $B = \{0^n1^n | n \geq 0\}$. Since the number of zeros in each word can be infinite. The machine needs the infinite number of states to keep track of the number of zeros. We therefore cannot find an NFA that recognizes B.

Next, we study a method to prove that languages are not regular.

2 Pumping Lemma for Regular Languages

Theorem 1. *If L is a regular language, then there exists a number $p > 0$ (the pumping length) where, for all $s \in L$ and $|s| \geq p$, there exist x, y, and z that $s = xyz$ and satisfy the following conditions:*

(1) for all $i \geq 0$, $xy^iz \in L$, *($y^0 = \varepsilon$)*

(2) $|y| > 0$, and

(3) $|xy| \leq p$.

Example 1. Let $L = \{w | w \in (\Sigma\Sigma\Sigma)^*\}$ where $\Sigma = \{0, 1\}$, and let $s = 010101 \in L$. Show that s can be pumped.

Let P represent "a language L is regular", and Q represent the pumping condition described above. By the pumping lemma, we have

$$P \to Q \equiv \neg Q \to \neg P$$

We can show that a language is *not* regular, by showing that it violates the pumping condition.

A language L does *not* satisfy the pumping condition if

there exists a string $s \in L$ and $|s| \geq p$ such that

for all strings x, y, z that $s = xyz$ with $|xy| \leq p$ and $|y| > 0$.

there exists an $i \geq 0$ such that $xy^iz \notin L$.

To show the condition violation, we can carefully choose a string s that all strings x, y, z satisfies the second and the the third conditions. Then, argue that there exists an i such that $xy^iz \notin L$.

Be careful! The pumping lemma cannot be used to show that a language is regular.

Example 2. Prove that a language $A = \{0^n 1^n | n \geq 0\}$ is not regular.

Example 3. Let $B = \{w | w$ has an equal number of 0s and 1s$\}$. Use the pumping lemma to prove that B is not regular.

Example 4. Let $C = \{0^i 1^j | i > j\}$. Show that C is not regular.

3 Context-Free Grammars

Context-Free Grammar (CFG), similar to regular expression, is a technique to describe languages. It is capable to describe non-regular language.

Definition 1. A context-free grammar is a 4-tuple (V, Σ, R, S), where

(1) V is a finite set called the *variables*,

(2) Σ is a finite set, disjoint from V, called the *terminals*,

(3) R is a finite relation from V to $(V \cup \Sigma)^*$ and each member of R is called a *rule*, and

(4) $S \in V$ is the start variable.

Each string of the language described a grammar can be generated by

1. Write down the start variable.
2. Find a variable that is written down and a rule starting with that variable. Substitute the variable with the right-hand side of that rule.
3. Repeat step 2 until no variables remain.

We can use a *parse tree* to show how the substitutions have been performed.

Definition 2. Let G be a context-free grammar, the language of the grammar, $L(G)$ composes of all strings that can be generated from G.

Definition 3. Any language that can be generated by some context-free grammar is called a *context-free language*.

Example 5. Let $G_1 = (\{S\}, \{\mathtt{a}, \mathtt{b}\}, R, S)$ where R is

$$S \to \mathtt{a}S\mathtt{b}$$
$$S \to SS$$
$$S \to \varepsilon$$

or $S \to \mathtt{a}S\mathtt{b} \mid SS \mid \varepsilon$. Draw a parse tree for each of the following strings.

1. `aaabbb`

2. `aababb`

Example 6. Write context-free grammars for the following languages when $\Sigma = \{0, 1\}$.

1. $\{0^n 1^n | n \geq 0\}$

2. $\{w | w$ contains at least three 1s$\}$

3. $\{w |$length of w is odd$\}$

4. $\{w | w$ is a palindrome$\}$

Definition 4. A context-free grammar is in *Chomsky normal form* if every rule is of the form

$$A \to BC$$
$$A \to a$$

where $a \in \Sigma$, and A, B, $C \in V$, except that B and C may not be the start variable. In additional we permit the rule $S \to \varepsilon$, where S is the start variable.

Theorem 2. *Any context-free language is generated by a context-free grammar in Chomsky normal form.*

To prove Theorem 2, we show steps to convert any grammar G into Chomsky normal form.

1. Add a new start variable S_0.
2. Eliminate all ε rules of the form $A \to \varepsilon$.
3. Eliminate all unit rules of the form $A \to B$.
4. Clean up the remaining rules by replacing

   $$A \to u_1 u_2 \dots u_k, k \geq 3, u_i \in V \cup \Sigma$$

   with $A \to u_1 A_1, A_1 \to u_2 A_2, \dots, A_{k-2} \to u_{k-1} u_k$. When $k = 2$, we replace any terminal u_i with the new variable U_i,.

Example 7. Let G_2 be a context-free grammar with the following rules

$$S \to ASA \mid \mathtt{a}B$$
$$A \to B \mid S$$
$$B \to \mathtt{b} \mid \varepsilon$$

Convert G_2 to Chomsky normal form.

4 Pushdown Automata

A *pushdown automaton* is a computational model composing of a *nondeterministic finite automata* and a *stack* with unlimited space.

Definition 5. A *pushdown automaton* is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets, and

(1) Q is the set of states,

(2) Σ is the input alphabets,

(3) Γ is the stack alphabets,

(4) $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to 2^{Q \times \Gamma_\varepsilon}$ is the transition function,

(5) $q_0 \in Q$ is the start state, and

(6) $F \subseteq Q$ is the set of accept states.

A pushdown automaton starts in the start state and with the empty stack (the alphabet at the top of stack is initially ε). In each step, it reads an alphabet from the input and an alphabet from the stack, then determines the set of the next states according to the transition function. Similar to NFA, the pushdown automaton follows all the possibilities in parallel. It accepts a string when it reads all the alphabets and reaches one of the accept states.

Example 8. Let M_1 be $(Q, \Sigma, \Gamma, \delta, q_1, F)$ where

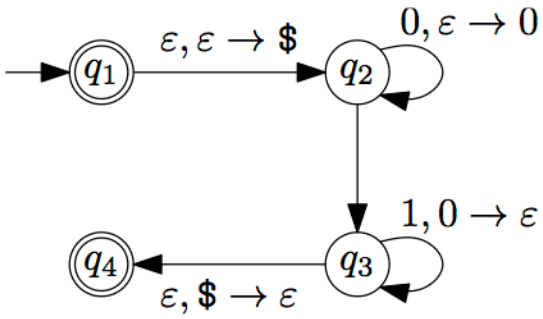
$$
\begin{aligned}
Q &= \{q_1, q_2, q_3, q_4\} \\
\Sigma &= \{0, 1\} \\
\Gamma &= \{0, \$\} \\
F &= \{q_1, q_4\}, \text{ and}
\end{aligned}
$$

δ is given in the following table:

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									

Show how M_1 works given a string 0011.

We can draw a state diagram for M_1 as



Note: "$a, b \to c$" signifies that the machine reads an a from the input, and replaces the symbol b on the top of the stack with a c.

Example 9. Let M_2 be a PDA that recognizes the language $\{ww^{\mathcal{R}} | w \in \{0,1\}^*\}$. Draw a state diagram for M_2.

Theorem 3. *A language is context free if and only if some pushdown automaton recognizes it.*

Here are the informal description of a pushdown automaton that recognizes the same language specified by a CFG.

1. Place the marker symbol \$ and the start variable in the stack.
2. Repeat the following steps
   (a) If the top of stack is a variable symbol A, nondeterministically select one of the rules for A and substitute A by the string on the rhs of the rule.
   (b) If the top of stack is a terminal symbol a, read the next symbol from the input and compare it to a, If they match, repeat. If they do not match, reject on this branch of the nondeterminism.
   (c) If the top of stack is the symbol \$, enter the accept state. Doing so accepts the input if it has all been read.

Example 10. Construct a PDA from the following CFG

$$S \to aTb \mid b$$
$$T \to Ta \mid \varepsilon$$

References

[1] Michael Sipser, *Introduction to the Theory of Computation, 2nd Edition, Thomson Course Technology*, Thomson Course Technology 2006. ISBN 0-534-95097-3.

[2] Richard Cole, *Theory of Computing. What is Feasible, Infeasible, and Impossible Computationally*, Lecture notes.